

cpp-hocon

0.3.0

Generated by Doxygen 1.9.1



<b>1 C++ HOCON Parser</b>	<b>1</b>
1.1 Caveats	1
1.2 Build Requirements	1
1.3 Pre-Build	2
1.4 Building	2
1.5 Testing	2
<b>2 Namespace Index</b>	<b>3</b>
2.1 Namespace List	3
<b>3 Hierarchical Index</b>	<b>5</b>
3.1 Class Hierarchy	5
<b>4 Class Index</b>	<b>9</b>
4.1 Class List	9
<b>5 File Index</b>	<b>13</b>
5.1 File List	13
<b>6 Namespace Documentation</b>	<b>15</b>
6.1 hocon Namespace Reference	15
6.1.1 Detailed Description	19
6.1.2 Typedef Documentation	19
6.1.2.1 duration	19
6.1.3 Function Documentation	19
6.1.3.1 operator==()	19
<b>7 Class Documentation</b>	<b>21</b>
7.1 hocon::abstract_config_node Class Reference	21
7.1.1 Detailed Description	21
7.1.2 Member Function Documentation	21
7.1.2.1 render()	21
7.2 hocon::abstract_config_node_value Class Reference	22
7.2.1 Detailed Description	22
7.2.2 Member Function Documentation	22
7.2.2.1 render()	22
7.3 hocon::bad_path_exception Struct Reference	23
7.3.1 Detailed Description	23
7.4 hocon::bad_value_exception Struct Reference	23
7.4.1 Detailed Description	24
7.5 hocon::bug_or_broken_exception Struct Reference	24
7.5.1 Detailed Description	24
7.6 hocon::comment Class Reference	25
7.6.1 Detailed Description	25

7.7 hocon::config Class Reference	25
7.7.1 Detailed Description	28
7.7.2 Member Function Documentation	30
7.7.2.1 at_key()	30
7.7.2.2 at_path()	30
7.7.2.3 check_valid()	31
7.7.2.4 entry_set()	32
7.7.2.5 get_duration()	32
7.7.2.6 get_is_null()	33
7.7.2.7 has_path()	33
7.7.2.8 has_path_or_null()	34
7.7.2.9 is_empty()	35
7.7.2.10 is_resolved()	35
7.7.2.11 origin()	35
7.7.2.12 parse_file_any_syntax() [1/2]	35
7.7.2.13 parse_file_any_syntax() [2/2]	36
7.7.2.14 parse_string() [1/2]	36
7.7.2.15 parse_string() [2/2]	37
7.7.2.16 resolve() [1/2]	37
7.7.2.17 resolve() [2/2]	38
7.7.2.18 resolve_with() [1/2]	38
7.7.2.19 resolve_with() [2/2]	39
7.7.2.20 root()	39
7.7.2.21 to_fallback_value()	40
7.7.2.22 with_fallback()	40
7.7.2.23 with_only_path()	41
7.7.2.24 with_value()	41
7.7.2.25 without_path()	42
7.8 hocon::config_boolean Class Reference	42
7.8.1 Detailed Description	44
7.8.2 Member Enumeration Documentation	44
7.8.2.1 type	44
7.8.3 Member Function Documentation	44
7.8.3.1 at_key()	44
7.8.3.2 at_path()	45
7.8.3.3 origin()	45
7.8.3.4 relativized()	46
7.8.3.5 render() [1/2]	47
7.8.3.6 render() [2/2]	47
7.8.3.7 to_fallback_value()	48
7.8.3.8 value_type()	48
7.8.3.9 value_type_name()	48

7.8.3.10 with_fallback()	49
7.8.3.11 with_origin()	49
7.9 hocon::config_concatenation Class Reference	50
7.9.1 Detailed Description	52
7.9.2 Member Enumeration Documentation	52
7.9.2.1 type	52
7.9.3 Member Function Documentation	52
7.9.3.1 at_key()	52
7.9.3.2 at_path()	53
7.9.3.3 has_descendant()	53
7.9.3.4 origin()	53
7.9.3.5 relativized()	54
7.9.3.6 render() [1/2]	54
7.9.3.7 render() [2/2]	54
7.9.3.8 replace_child()	55
7.9.3.9 to_fallback_value()	55
7.9.3.10 value_type()	55
7.9.3.11 value_type_name()	56
7.9.3.12 with_fallback()	56
7.9.3.13 with_origin()	57
7.10 hocon::config_delayed_merge Class Reference	57
7.10.1 Detailed Description	59
7.10.2 Member Enumeration Documentation	59
7.10.2.1 type	59
7.10.3 Member Function Documentation	60
7.10.3.1 at_key()	60
7.10.3.2 at_path()	60
7.10.3.3 has_descendant()	60
7.10.3.4 origin()	61
7.10.3.5 relativized()	61
7.10.3.6 render() [1/2]	61
7.10.3.7 render() [2/2]	62
7.10.3.8 replace_child()	62
7.10.3.9 to_fallback_value()	63
7.10.3.10 value_type()	63
7.10.3.11 value_type_name()	63
7.10.3.12 with_fallback()	64
7.10.3.13 with_origin()	64
7.11 hocon::config_delayed_merge_object Class Reference	65
7.11.1 Detailed Description	67
7.11.2 Member Enumeration Documentation	67
7.11.2.1 type	67

7.11.3 Member Function Documentation	68
7.11.3.1 at_key()	68
7.11.3.2 at_path()	68
7.11.3.3 attempt_peek_with_partial_resolve()	68
7.11.3.4 has_descendant()	69
7.11.3.5 key_set()	69
7.11.3.6 origin()	70
7.11.3.7 relativized()	70
7.11.3.8 render() [1/2]	70
7.11.3.9 render() [2/2]	71
7.11.3.10 replace_child()	71
7.11.3.11 to_config()	71
7.11.3.12 to_fallback_value()	72
7.11.3.13 value_type()	72
7.11.3.14 value_type_name()	72
7.11.3.15 with_fallback()	73
7.11.3.16 with_origin()	73
7.12 hocon::config_document Class Reference	74
7.12.1 Detailed Description	75
7.12.2 Member Function Documentation	75
7.12.2.1 has_path()	75
7.12.2.2 render()	75
7.12.2.3 with_value()	76
7.12.2.4 with_value_text()	76
7.12.2.5 without_path()	77
7.13 hocon::config_double Class Reference	77
7.13.1 Detailed Description	79
7.13.2 Member Enumeration Documentation	79
7.13.2.1 type	80
7.13.3 Member Function Documentation	80
7.13.3.1 at_key()	80
7.13.3.2 at_path()	80
7.13.3.3 origin()	81
7.13.3.4 relativized()	81
7.13.3.5 render() [1/2]	81
7.13.3.6 render() [2/2]	82
7.13.3.7 to_fallback_value()	82
7.13.3.8 value_type()	83
7.13.3.9 value_type_name()	83
7.13.3.10 with_fallback()	83
7.13.3.11 with_origin()	84
7.14 hocon::config_exception Struct Reference	84

7.14.1 Detailed Description	85
7.15 hocon::config_include_context Class Reference	85
7.15.1 Detailed Description	86
7.15.2 Member Function Documentation	86
7.15.2.1 parse_options()	86
7.15.2.2 relative_to()	86
7.16 hocon::config_includer Class Reference	87
7.16.1 Detailed Description	87
7.16.2 Member Function Documentation	87
7.16.2.1 include()	88
7.16.2.2 with_fallback()	88
7.17 hocon::config_includer_file Class Reference	89
7.17.1 Detailed Description	89
7.17.2 Member Function Documentation	89
7.17.2.1 include_file()	89
7.18 hocon::config_int Class Reference	90
7.18.1 Detailed Description	92
7.18.2 Member Enumeration Documentation	92
7.18.2.1 type	92
7.18.3 Member Function Documentation	92
7.18.3.1 at_key()	92
7.18.3.2 at_path()	93
7.18.3.3 origin()	93
7.18.3.4 relativized()	93
7.18.3.5 render() [1/2]	94
7.18.3.6 render() [2/2]	94
7.18.3.7 to_fallback_value()	95
7.18.3.8 value_type()	95
7.18.3.9 value_type_name()	95
7.18.3.10 with_fallback()	96
7.18.3.11 with_origin()	96
7.19 hocon::config_list Class Reference	97
7.19.1 Detailed Description	99
7.19.2 Member Enumeration Documentation	100
7.19.2.1 type	100
7.19.3 Member Function Documentation	100
7.19.3.1 at_key()	100
7.19.3.2 at_path()	100
7.19.3.3 origin()	101
7.19.3.4 relativized()	101
7.19.3.5 render() [1/2]	102
7.19.3.6 render() [2/2]	102

---

7.19.3.7 to_fallback_value()	102
7.19.3.8 value_type()	103
7.19.3.9 value_type_name()	103
7.19.3.10 with_fallback()	104
7.19.3.11 with_origin()	104
7.20 hocon::config_long Class Reference	105
7.20.1 Detailed Description	107
7.20.2 Member Enumeration Documentation	107
7.20.2.1 type	107
7.20.3 Member Function Documentation	107
7.20.3.1 at_key()	107
7.20.3.2 at_path()	108
7.20.3.3 origin()	108
7.20.3.4 relativized()	109
7.20.3.5 render() [1/2]	110
7.20.3.6 render() [2/2]	110
7.20.3.7 to_fallback_value()	111
7.20.3.8 value_type()	111
7.20.3.9 value_type_name()	111
7.20.3.10 with_fallback()	112
7.20.3.11 with_origin()	112
7.21 hocon::config_mergeable Class Reference	113
7.21.1 Detailed Description	114
7.21.2 Member Function Documentation	114
7.21.2.1 to_fallback_value()	114
7.21.2.2 with_fallback()	114
7.22 hocon::config_node Class Reference	115
7.22.1 Detailed Description	115
7.22.2 Member Function Documentation	115
7.22.2.1 render()	116
7.23 hocon::config_node_array Class Reference	116
7.23.1 Detailed Description	116
7.23.2 Member Function Documentation	117
7.23.2.1 render()	117
7.24 hocon::config_node_comment Class Reference	117
7.24.1 Detailed Description	117
7.24.2 Member Function Documentation	118
7.24.2.1 render()	118
7.25 hocon::config_node_complex_value Class Reference	118
7.25.1 Detailed Description	118
7.25.2 Member Function Documentation	119
7.25.2.1 render()	119



7.26 hocon::config_node_concatenation Class Reference	119
7.26.1 Detailed Description	120
7.26.2 Member Function Documentation	120
7.26.2.1 render()	120
7.27 hocon::config_node_field Class Reference	120
7.27.1 Detailed Description	121
7.27.2 Member Function Documentation	121
7.27.2.1 render()	121
7.28 hocon::config_node_include Class Reference	121
7.28.1 Detailed Description	122
7.28.2 Member Function Documentation	122
7.28.2.1 render()	122
7.29 hocon::config_node_object Class Reference	122
7.29.1 Detailed Description	123
7.29.2 Member Function Documentation	123
7.29.2.1 render()	123
7.30 hocon::config_node_path Class Reference	124
7.30.1 Detailed Description	124
7.30.2 Member Function Documentation	124
7.30.2.1 render()	124
7.31 hocon::config_node_root Class Reference	125
7.31.1 Detailed Description	125
7.31.2 Member Function Documentation	125
7.31.2.1 render()	125
7.32 hocon::config_node_simple_value Class Reference	126
7.32.1 Detailed Description	126
7.32.2 Member Function Documentation	126
7.32.2.1 render()	126
7.33 hocon::config_node_single_token Class Reference	127
7.33.1 Detailed Description	127
7.33.2 Member Function Documentation	127
7.33.2.1 render()	127
7.34 hocon::config_null Class Reference	128
7.34.1 Detailed Description	129
7.34.2 Member Enumeration Documentation	129
7.34.2.1 type	130
7.34.3 Member Function Documentation	130
7.34.3.1 at_key()	130
7.34.3.2 at_path()	130
7.34.3.3 origin()	131
7.34.3.4 relativized()	131
7.34.3.5 render() [1/2]	131

7.34.3.6 render() [2/2]	132
7.34.3.7 to_fallback_value()	132
7.34.3.8 value_type()	133
7.34.3.9 value_type_name()	133
7.34.3.10 with_fallback()	133
7.34.3.11 with_origin()	134
7.35 hocon::config_number Class Reference	134
7.35.1 Detailed Description	136
7.35.2 Member Enumeration Documentation	136
7.35.2.1 type	137
7.35.3 Member Function Documentation	137
7.35.3.1 at_key()	137
7.35.3.2 at_path()	137
7.35.3.3 origin()	138
7.35.3.4 relativized()	138
7.35.3.5 render() [1/2]	138
7.35.3.6 render() [2/2]	139
7.35.3.7 to_fallback_value()	139
7.35.3.8 value_type()	140
7.35.3.9 value_type_name()	140
7.35.3.10 with_fallback()	140
7.35.3.11 with_origin()	141
7.36 hocon::config_object Class Reference	141
7.36.1 Detailed Description	144
7.36.2 Member Enumeration Documentation	144
7.36.2.1 type	144
7.36.3 Member Function Documentation	144
7.36.3.1 at_key()	144
7.36.3.2 at_path()	144
7.36.3.3 attempt_peek_with_partial_resolve()	145
7.36.3.4 key_set()	145
7.36.3.5 origin()	146
7.36.3.6 relativized()	146
7.36.3.7 render() [1/2]	146
7.36.3.8 render() [2/2]	147
7.36.3.9 to_config()	147
7.36.3.10 to_fallback_value()	147
7.36.3.11 value_type()	148
7.36.3.12 value_type_name()	148
7.36.3.13 with_fallback()	148
7.36.3.14 with_origin()	149
7.37 hocon::config_origin Class Reference	149

7.37.1 Detailed Description	150
7.37.2 Member Function Documentation	150
7.37.2.1 comments()	150
7.37.2.2 description()	151
7.37.2.3 line_number()	151
7.37.2.4 with_comments()	151
7.37.2.5 with_line_number()	152
7.38 hocon::config_parse_options Class Reference	153
7.38.1 Detailed Description	153
7.38.2 Constructor & Destructor Documentation	154
7.38.2.1 config_parse_options()	154
7.38.3 Member Function Documentation	154
7.38.3.1 append_includer()	154
7.38.3.2 defaults()	154
7.38.3.3 get_allow_missing()	155
7.38.3.4 get_includer()	155
7.38.3.5 get_origin_description()	155
7.38.3.6 get_syntax()	155
7.38.3.7 prepend_includer()	155
7.38.3.8 set_allow_missing()	156
7.38.3.9 set_includer()	156
7.38.3.10 set_origin_description()	157
7.38.3.11 set_syntax()	157
7.39 hocon::config_parseable Class Reference	157
7.39.1 Detailed Description	158
7.39.2 Member Function Documentation	158
7.39.2.1 options()	158
7.39.2.2 origin()	159
7.39.2.3 parse()	159
7.40 hocon::config_reference Class Reference	159
7.40.1 Detailed Description	161
7.40.2 Member Enumeration Documentation	161
7.40.2.1 type	161
7.40.3 Member Function Documentation	161
7.40.3.1 at_key()	161
7.40.3.2 at_path()	162
7.40.3.3 origin()	162
7.40.3.4 relativized()	163
7.40.3.5 render() [1/2]	164
7.40.3.6 render() [2/2]	164
7.40.3.7 to_fallback_value()	165
7.40.3.8 value_type()	165

7.40.3.9	<a href="#">value_type_name()</a>	165
7.40.3.10	<a href="#">with_fallback()</a>	166
7.40.3.11	<a href="#">with_origin()</a>	166
7.41	<a href="#">hocon::config_render_options Class Reference</a>	167
7.41.1	<a href="#">Detailed Description</a>	168
7.41.2	<a href="#">Constructor &amp; Destructor Documentation</a>	168
7.41.2.1	<a href="#">config_render_options()</a>	168
7.41.3	<a href="#">Member Function Documentation</a>	168
7.41.3.1	<a href="#">concise()</a>	168
7.41.3.2	<a href="#">get_comments()</a>	169
7.41.3.3	<a href="#">get_formatted()</a>	169
7.41.3.4	<a href="#">get_json()</a>	169
7.41.3.5	<a href="#">get_origin_comments()</a>	169
7.41.3.6	<a href="#">set_comments()</a>	169
7.41.3.7	<a href="#">set_formatted()</a>	170
7.41.3.8	<a href="#">set_json()</a>	170
7.41.3.9	<a href="#">set_origin_comments()</a>	171
7.42	<a href="#">hocon::config_resolve_options Class Reference</a>	171
7.42.1	<a href="#">Detailed Description</a>	172
7.42.2	<a href="#">Constructor &amp; Destructor Documentation</a>	172
7.42.2.1	<a href="#">config_resolve_options()</a>	172
7.42.3	<a href="#">Member Function Documentation</a>	172
7.42.3.1	<a href="#">get_allow_unresolved()</a>	173
7.42.3.2	<a href="#">get_use_system_environment()</a>	173
7.42.3.3	<a href="#">set_allow_unresolved()</a>	173
7.42.3.4	<a href="#">set_use_system_environment()</a>	174
7.43	<a href="#">hocon::config_string Class Reference</a>	174
7.43.1	<a href="#">Detailed Description</a>	176
7.43.2	<a href="#">Member Enumeration Documentation</a>	176
7.43.2.1	<a href="#">type</a>	176
7.43.3	<a href="#">Member Function Documentation</a>	176
7.43.3.1	<a href="#">at_key()</a>	176
7.43.3.2	<a href="#">at_path()</a>	177
7.43.3.3	<a href="#">origin()</a>	177
7.43.3.4	<a href="#">relativized()</a>	177
7.43.3.5	<a href="#">render()</a> [1/2]	178
7.43.3.6	<a href="#">render()</a> [2/2]	178
7.43.3.7	<a href="#">to_fallback_value()</a>	179
7.43.3.8	<a href="#">value_type()</a>	179
7.43.3.9	<a href="#">value_type_name()</a>	179
7.43.3.10	<a href="#">with_fallback()</a>	180
7.43.3.11	<a href="#">with_origin()</a>	180

7.44 hocon::config_value Class Reference . . . . .	181
7.44.1 Detailed Description . . . . .	183
7.44.2 Member Enumeration Documentation . . . . .	184
7.44.2.1 type . . . . .	184
7.44.3 Member Function Documentation . . . . .	184
7.44.3.1 at_key() . . . . .	184
7.44.3.2 at_path() . . . . .	184
7.44.3.3 origin() . . . . .	185
7.44.3.4 relativized() . . . . .	185
7.44.3.5 render() [1/2] . . . . .	186
7.44.3.6 render() [2/2] . . . . .	186
7.44.3.7 to_fallback_value() . . . . .	186
7.44.3.8 value_type() . . . . .	187
7.44.3.9 value_type_name() . . . . .	187
7.44.3.10 with_fallback() . . . . .	188
7.44.3.11 with_origin() . . . . .	188
7.45 hocon::config_value_factory Class Reference . . . . .	189
7.45.1 Detailed Description . . . . .	189
7.45.2 Member Function Documentation . . . . .	189
7.45.2.1 from_any_ref() . . . . .	189
7.46 hocon::container Class Reference . . . . .	190
7.46.1 Detailed Description . . . . .	190
7.46.2 Member Function Documentation . . . . .	190
7.46.2.1 has_descendant() . . . . .	191
7.46.2.2 replace_child() . . . . .	191
7.47 hocon::default_transformer Class Reference . . . . .	191
7.47.1 Detailed Description . . . . .	191
7.48 hocon::double_slash_comment Class Reference . . . . .	192
7.48.1 Detailed Description . . . . .	192
7.49 hocon::file_name_source Class Reference . . . . .	192
7.49.1 Detailed Description . . . . .	193
7.50 hocon::full_includer Class Reference . . . . .	193
7.50.1 Detailed Description . . . . .	193
7.50.2 Member Function Documentation . . . . .	193
7.50.2.1 include() . . . . .	193
7.50.2.2 include_file() . . . . .	194
7.50.2.3 with_fallback() . . . . .	194
7.51 FwdListIter< T > Class Template Reference . . . . .	195
7.51.1 Detailed Description . . . . .	195
7.52 hocon::generic_exception Struct Reference . . . . .	196
7.52.1 Detailed Description . . . . .	196
7.53 hocon::hash_comment Class Reference . . . . .	196

7.53.1 Detailed Description	197
7.54 hocon::ignored_whitespace Class Reference	197
7.54.1 Detailed Description	197
7.55 hocon::io_exception Struct Reference	197
7.55.1 Detailed Description	198
7.56 hocon::iterator Class Reference	198
7.56.1 Detailed Description	198
7.57 hocon::iterator_wrapper< iter > Class Template Reference	198
7.57.1 Detailed Description	199
7.58 hocon::line Class Reference	199
7.58.1 Detailed Description	199
7.59 List< T > Class Template Reference	200
7.59.1 Detailed Description	200
7.60 hocon::missing_exception Struct Reference	200
7.60.1 Detailed Description	201
7.61 hocon::config_value::modifier Class Reference	201
7.61.1 Detailed Description	201
7.62 hocon::name_source Class Reference	201
7.62.1 Detailed Description	202
7.63 hocon::config_value::no_exceptions_modifier Class Reference	202
7.63.1 Detailed Description	202
7.64 hocon::not_possible_to_resolve_exception Struct Reference	203
7.64.1 Detailed Description	203
7.65 hocon::not_resolved_exception Struct Reference	203
7.65.1 Detailed Description	204
7.66 hocon::null_exception Struct Reference	204
7.66.1 Detailed Description	204
7.67 OutListIter< T > Class Template Reference	205
7.67.1 Detailed Description	205
7.68 hocon::config_document_parser::parse_context Class Reference	205
7.68.1 Detailed Description	205
7.68.2 Member Function Documentation	205
7.68.2.1 parse_single_value()	206
7.69 hocon::config_parser::parse_context Class Reference	206
7.69.1 Detailed Description	206
7.70 hocon::parse_exception Struct Reference	206
7.70.1 Detailed Description	207
7.71 hocon::parseable Class Reference	207
7.71.1 Detailed Description	208
7.71.2 Member Function Documentation	208
7.71.2.1 options()	208
7.71.2.2 origin()	208

7.71.2.3 parse()	208
7.72 hocon::parseable_file Class Reference	209
7.72.1 Detailed Description	210
7.72.2 Member Function Documentation	210
7.72.2.1 options()	210
7.72.2.2 origin()	210
7.72.2.3 parse()	210
7.73 hocon::parseable_not_found Class Reference	211
7.73.1 Detailed Description	212
7.73.2 Member Function Documentation	212
7.73.2.1 options()	212
7.73.2.2 origin()	212
7.73.2.3 parse()	212
7.74 hocon::parseable_resources Class Reference	213
7.74.1 Detailed Description	214
7.74.2 Member Function Documentation	214
7.74.2.1 options()	214
7.74.2.2 origin()	214
7.74.2.3 parse()	214
7.75 hocon::parseable_string Class Reference	215
7.75.1 Detailed Description	216
7.75.2 Member Function Documentation	216
7.75.2.1 options()	216
7.75.2.2 origin()	216
7.75.2.3 parse()	216
7.76 hocon::path Class Reference	217
7.76.1 Detailed Description	218
7.76.2 Member Function Documentation	218
7.76.2.1 has_funky_chars()	218
7.76.2.2 parent()	218
7.76.2.3 remainder()	218
7.76.2.4 render()	218
7.76.2.5 to_string()	218
7.77 hocon::path_builder Class Reference	219
7.77.1 Detailed Description	219
7.77.2 Member Function Documentation	219
7.77.2.1 result()	219
7.78 hocon::path_parser Class Reference	219
7.78.1 Detailed Description	219
7.79 hocon::problem Class Reference	220
7.79.1 Detailed Description	220
7.80 hocon::problem_exception Class Reference	220

7.80.1 Detailed Description	221
7.81 hocon::relative_name_source Class Reference	221
7.81.1 Detailed Description	221
7.82 hocon::replaceable_merge_stack Class Reference	221
7.82.1 Detailed Description	222
7.82.2 Member Function Documentation	222
7.82.2.1 has_descendant()	222
7.82.2.2 replace_child()	222
7.83 hocon::resolve_context Class Reference	223
7.83.1 Detailed Description	223
7.84 hocon::resolve_result< V > Struct Template Reference	223
7.84.1 Detailed Description	223
7.85 hocon::resolve_source Class Reference	224
7.85.1 Detailed Description	224
7.86 hocon::resolve_source::result_with_path Struct Reference	224
7.86.1 Detailed Description	224
7.87 hocon::simple_config_document Class Reference	225
7.87.1 Detailed Description	225
7.87.2 Member Function Documentation	225
7.87.2.1 has_path()	225
7.87.2.2 render()	226
7.87.2.3 with_value()	226
7.87.2.4 with_value_text()	226
7.87.2.5 without_path()	227
7.88 hocon::simple_config_list Class Reference	228
7.88.1 Detailed Description	230
7.88.2 Member Enumeration Documentation	230
7.88.2.1 type	230
7.88.3 Member Function Documentation	230
7.88.3.1 at_key()	230
7.88.3.2 at_path()	230
7.88.3.3 has_descendant()	231
7.88.3.4 origin()	231
7.88.3.5 relativized()	231
7.88.3.6 render() [1/2]	232
7.88.3.7 render() [2/2]	232
7.88.3.8 replace_child()	233
7.88.3.9 to_fallback_value()	233
7.88.3.10 value_type()	233
7.88.3.11 value_type_name()	234
7.88.3.12 with_fallback()	234
7.88.3.13 with_origin()	235



7.89 hocon::simple_config_object Class Reference . . . . .	235
7.89.1 Detailed Description . . . . .	238
7.89.2 Member Enumeration Documentation . . . . .	238
7.89.2.1 type . . . . .	238
7.89.3 Member Function Documentation . . . . .	238
7.89.3.1 at_key() . . . . .	238
7.89.3.2 at_path() . . . . .	239
7.89.3.3 attempt_peek_with_partial_resolve() . . . . .	239
7.89.3.4 has_descendant() . . . . .	240
7.89.3.5 key_set() . . . . .	240
7.89.3.6 origin() . . . . .	240
7.89.3.7 relativized() . . . . .	240
7.89.3.8 render() [1/2] . . . . .	241
7.89.3.9 render() [2/2] . . . . .	241
7.89.3.10 replace_child() . . . . .	242
7.89.3.11 to_config() . . . . .	242
7.89.3.12 to_fallback_value() . . . . .	242
7.89.3.13 value_set() . . . . .	243
7.89.3.14 value_type() . . . . .	243
7.89.3.15 value_type_name() . . . . .	243
7.89.3.16 with_fallback() . . . . .	244
7.89.3.17 with_only_path_or_null() . . . . .	244
7.89.3.18 with_origin() . . . . .	245
7.90 hocon::simple_config_origin Class Reference . . . . .	245
7.90.1 Detailed Description . . . . .	246
7.90.2 Constructor & Destructor Documentation . . . . .	246
7.90.2.1 simple_config_origin() . . . . .	246
7.90.3 Member Function Documentation . . . . .	246
7.90.3.1 comments() . . . . .	247
7.90.3.2 description() . . . . .	247
7.90.3.3 line_number() . . . . .	247
7.90.3.4 with_comments() . . . . .	248
7.90.3.5 with_line_number() . . . . .	248
7.91 hocon::simple_include_context Class Reference . . . . .	249
7.91.1 Detailed Description . . . . .	249
7.91.2 Member Function Documentation . . . . .	249
7.91.2.1 parse_options() . . . . .	249
7.91.2.2 relative_to() . . . . .	250
7.92 hocon::simple_includer Class Reference . . . . .	250
7.92.1 Detailed Description . . . . .	251
7.92.2 Member Function Documentation . . . . .	251
7.92.2.1 include() . . . . .	251

7.92.2.2 <code>include_file()</code> . . . . .	251
7.92.2.3 <code>with_fallback()</code> . . . . .	252
7.93 <code>hocon::single_token_iterator</code> Class Reference . . . . .	252
7.93.1 Detailed Description . . . . .	253
7.94 <code>hocon::substitution</code> Class Reference . . . . .	253
7.94.1 Detailed Description . . . . .	253
7.95 <code>hocon::substitution_expression</code> Class Reference . . . . .	254
7.95.1 Detailed Description . . . . .	254
7.96 <code>hocon::token</code> Class Reference . . . . .	254
7.96.1 Detailed Description . . . . .	255
7.97 <code>hocon::token_iterator</code> Class Reference . . . . .	255
7.97.1 Detailed Description . . . . .	255
7.98 <code>hocon::token_list_iterator</code> Class Reference . . . . .	255
7.98.1 Detailed Description . . . . .	256
7.99 <code>hocon::tokens</code> Class Reference . . . . .	256
7.99.1 Detailed Description . . . . .	256
7.99.2 Member Function Documentation . . . . .	256
7.99.2.1 <code>start_token()</code> . . . . .	257
7.100 <code>hocon::unmergeable</code> Class Reference . . . . .	257
7.100.1 Detailed Description . . . . .	257
7.101 <code>hocon::unquoted_text</code> Class Reference . . . . .	257
7.101.1 Detailed Description . . . . .	258
7.102 <code>hocon::unresolved_substitution_exception</code> Struct Reference . . . . .	258
7.102.1 Detailed Description . . . . .	258
7.103 <code>hocon::unsupported_exception</code> Struct Reference . . . . .	259
7.103.1 Detailed Description . . . . .	259
7.104 <code>hocon::validation_failed_exception</code> Struct Reference . . . . .	259
7.104.1 Detailed Description . . . . .	260
7.105 <code>hocon::validation_problem</code> Struct Reference . . . . .	260
7.105.1 Detailed Description . . . . .	260
7.106 <code>hocon::value</code> Class Reference . . . . .	260
7.106.1 Detailed Description . . . . .	261
7.107 <code>hocon::wrong_type_exception</code> Struct Reference . . . . .	261
7.107.1 Detailed Description . . . . .	261
<b>8 File Documentation</b> . . . . .	<b>263</b>
8.1 <code>hocon/version.h</code> File Reference . . . . .	263
8.1.1 Detailed Description . . . . .	263
8.1.2 Macro Definition Documentation . . . . .	263
8.1.2.1 <code>CPP_HOCON_VERSION</code> . . . . .	263
8.1.2.2 <code>CPP_HOCON_VERSION_MAJOR</code> . . . . .	264
8.1.2.3 <code>CPP_HOCON_VERSION_MINOR</code> . . . . .	264

---

8.1.2.4 CPP_HOCON_VERSION_PATCH . . . . .	264
8.1.2.5 CPP_HOCON_VERSION_WITH_COMMIT . . . . .	264
<b>Index</b>	<b>265</b>



# C++ HOCON Parser

The library provides C++ support for the HOCON configuration file format.

## 1.1 Caveats

- Include requires the location specifier, i.e. `include "foo"` won't work but `include file("foo")` will. URL is not yet implemented, and classpath won't be supported as it makes less sense outside of the JVM.
- Unicode testing is absent so support is unknown. There are likely things that won't work.

- OSX or Linux
- GCC  $\geq$  4.8 or Clang  $\geq$  3.4 (with libc++)
- CMake  $\geq$  3.2.2
- Boost Libraries  $\geq$  1.54
- Leatherman

## 1.3 Pre-Build

Prepare the cmake release environment:

```
$ mkdir release
$ cd release
$ cmake ..
```

Optionally, also prepare the debug environment:

```
$ mkdir debug
$ cd debug
$ cmake -DCMAKE_BUILD_TYPE=Debug ..
```

## 1.4 Building

1. Enter your build environment of choice, i.e. `cd release` or `cd debug`
2. `make`
3. (optional) install with `make install`

## 1.5 Testing

Run tests with `make test`.

## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

<a href="#">hocon</a>	Factory for creating <a href="#">config_document</a> instances . . . . .	<a href="#">15</a>
-----------------------	--	--------------------





## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

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

hocon::config_document . . . . .	74
hocon::simple_config_document . . . . .	225
hocon::config_include_context . . . . .	85
hocon::simple_include_context . . . . .	249
hocon::config_includer . . . . .	87
hocon::full_includer . . . . .	193
hocon::simple_includer . . . . .	250
hocon::config_includer_file . . . . .	89
hocon::full_includer . . . . .	193
hocon::simple_includer . . . . .	250
hocon::config_mergeable . . . . .	113
hocon::config . . . . .	25
hocon::config_value . . . . .	181
hocon::config_boolean . . . . .	42
hocon::config_concatenation . . . . .	50
hocon::config_delayed_merge . . . . .	57
hocon::config_list . . . . .	97
hocon::simple_config_list . . . . .	228
hocon::config_null . . . . .	128
hocon::config_number . . . . .	134
hocon::config_double . . . . .	77
hocon::config_int . . . . .	90
hocon::config_long . . . . .	105
hocon::config_object . . . . .	141
hocon::config_delayed_merge_object . . . . .	65
hocon::simple_config_object . . . . .	235
hocon::config_reference . . . . .	159
hocon::config_string . . . . .	174
hocon::config_node . . . . .	115
hocon::abstract_config_node . . . . .	21
hocon::abstract_config_node_value . . . . .	22
hocon::config_node_complex_value . . . . .	118
hocon::config_node_array . . . . .	116

hocon::config_node_concatenation . . . . .	119
hocon::config_node_object . . . . .	122
hocon::config_node_root . . . . .	125
hocon::config_node_field . . . . .	120
hocon::config_node_simple_value . . . . .	126
hocon::config_node_include . . . . .	121
hocon::config_node_path . . . . .	124
hocon::config_node_single_token . . . . .	127
hocon::config_node_comment . . . . .	117
hocon::config_origin . . . . .	149
hocon::simple_config_origin . . . . .	245
hocon::config_parse_options . . . . .	153
hocon::config_parseable . . . . .	157
hocon::parseable . . . . .	207
hocon::parseable_file . . . . .	209
hocon::parseable_not_found . . . . .	211
hocon::parseable_resources . . . . .	213
hocon::parseable_string . . . . .	215
hocon::config_render_options . . . . .	167
hocon::config_resolve_options . . . . .	171
hocon::config_value_factory . . . . .	189
hocon::container . . . . .	190
hocon::config_concatenation . . . . .	50
hocon::replaceable_merge_stack . . . . .	221
hocon::config_delayed_merge . . . . .	57
hocon::config_delayed_merge_object . . . . .	65
hocon::simple_config_list . . . . .	228
hocon::simple_config_object . . . . .	235
hocon::default_transformer . . . . .	191
std::enable_shared_from_this	
hocon::config . . . . .	25
hocon::config_value . . . . .	181
hocon::parseable . . . . .	207
hocon::simple_config_origin . . . . .	245
hocon::simple_includer . . . . .	250
hocon::substitution_expression . . . . .	254
std::exception	
std::runtime_error	
hocon::config_exception . . . . .	84
hocon::bad_path_exception . . . . .	23
hocon::bad_value_exception . . . . .	23
hocon::bug_or_broken_exception . . . . .	24
hocon::not_possible_to_resolve_exception . . . . .	203
hocon::not_resolved_exception . . . . .	203
hocon::generic_exception . . . . .	196
hocon::io_exception . . . . .	197
hocon::missing_exception . . . . .	200
hocon::null_exception . . . . .	204
hocon::parse_exception . . . . .	206
hocon::unresolved_substitution_exception . . . . .	258
hocon::validation_failed_exception . . . . .	259
hocon::wrong_type_exception . . . . .	261
hocon::problem_exception . . . . .	220
hocon::unsupported_exception . . . . .	259
hocon::iterator . . . . .	198
hocon::iterator_wrapper< iter > . . . . .	198

hocon::single_token_iterator . . . . .	252
hocon::token_iterator . . . . .	255
hocon::token_list_iterator . . . . .	255
std::iterator	
FwdListIter< T > . . . . .	195
OutListIter< T > . . . . .	205
List< T > . . . . .	200
List< shared_string > . . . . .	200
hocon::config_value::modifier . . . . .	201
hocon::config_value::no_exceptions_modifier . . . . .	202
hocon::name_source . . . . .	201
hocon::file_name_source . . . . .	192
hocon::relative_name_source . . . . .	221
hocon::config_document_parser::parse_context . . . . .	205
hocon::config_parser::parse_context . . . . .	206
hocon::path . . . . .	217
hocon::path_builder . . . . .	219
hocon::path_parser . . . . .	219
hocon::resolve_context . . . . .	223
hocon::resolve_result< V > . . . . .	223
hocon::resolve_result< shared_value > . . . . .	223
hocon::resolve_source . . . . .	224
hocon::resolve_source::result_with_path . . . . .	224
hocon::token . . . . .	254
hocon::comment . . . . .	25
hocon::double_slash_comment . . . . .	192
hocon::hash_comment . . . . .	196
hocon::ignored_whitespace . . . . .	197
hocon::line . . . . .	199
hocon::problem . . . . .	220
hocon::substitution . . . . .	253
hocon::unquoted_text . . . . .	257
hocon::value . . . . .	260
hocon::tokens . . . . .	256
hocon::unmergeable . . . . .	257
hocon::config_concatenation . . . . .	50
hocon::config_delayed_merge . . . . .	57
hocon::config_delayed_merge_object . . . . .	65
hocon::config_reference . . . . .	159
hocon::validation_problem . . . . .	260



## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">hocon::abstract_config_node</a> . . . . .	21
<a href="#">hocon::abstract_config_node_value</a> This is used to classify certain <a href="#">abstract_config_node</a> subclasses . . . . .	22
<a href="#">hocon::bad_path_exception</a> Exception indicating that a path expression was invalid . . . . .	23
<a href="#">hocon::bad_value_exception</a> Exception indicating that a value was messed up, for example you may have asked for a duration and the value can't be sensibly parsed as a duration . . . . .	23
<a href="#">hocon::bug_or_broken_exception</a> Exception indicating that there's a bug in something (possibly the library itself) or the runtime environment is broken . . . . .	24
<a href="#">hocon::comment</a> . . . . .	25
<a href="#">hocon::config</a> An immutable map from config paths to config values . . . . .	25
<a href="#">hocon::config_boolean</a> . . . . .	42
<a href="#">hocon::config_concatenation</a> A ConfigConcatenation represents a list of values to be concatenated (see the spec) . . . . .	50
<a href="#">hocon::config_delayed_merge</a> . . . . .	57
<a href="#">hocon::config_delayed_merge_object</a> . . . . .	65
<a href="#">hocon::config_document</a> Represents an individual HOCON or JSON file, preserving all formatting and syntax details . . . . .	74
<a href="#">hocon::config_double</a> . . . . .	77
<a href="#">hocon::config_exception</a> All exceptions thrown by the library are subclasses of <a href="#">config_exception</a> . . . . .	84
<a href="#">hocon::config_include_context</a> Context provided to a <a href="#">config_includer</a> ; this interface is only useful inside a . . . . .	85
<a href="#">hocon::config_includer</a> Implement this interface and provide an instance to <a href="#">config_parse_options.set_includer()</a> to cus- tomize handling of . . . . .	87
<a href="#">hocon::config_includer_file</a> Implement this <i>in addition to</i> <a href="#">config_includer</a> if you want to support inclusion of files with the . . . . .	89
<a href="#">hocon::config_int</a> . . . . .	90
<a href="#">hocon::config_list</a> Subtype of ConfigValue representing a list value, as in JSON's . . . . .	97
<a href="#">hocon::config_long</a> . . . . .	105

<a href="#">hocon::config_mergeable</a>	113
<a href="#">hocon::config_node</a>	
A node in the syntax tree for a HOCON or JSON document	115
<a href="#">hocon::config_node_array</a>	116
<a href="#">hocon::config_node_comment</a>	117
<a href="#">hocon::config_node_complex_value</a>	118
<a href="#">hocon::config_node_concatenation</a>	119
<a href="#">hocon::config_node_field</a>	
A field represents a key-value pair of the format "key : value", where key is a quoted or unquoted string, and value can be any node type	120
<a href="#">hocon::config_node_include</a>	
Represents an include statement of the form "include include_kind(include_path)"	121
<a href="#">hocon::config_node_object</a>	122
<a href="#">hocon::config_node_path</a>	124
<a href="#">hocon::config_node_root</a>	125
<a href="#">hocon::config_node_simple_value</a>	126
<a href="#">hocon::config_node_single_token</a>	127
<a href="#">hocon::config_null</a>	
This exists because sometimes null is not the same as missing	128
<a href="#">hocon::config_number</a>	134
<a href="#">hocon::config_object</a>	141
<a href="#">hocon::config_origin</a>	
Represents the origin (such as filename and line number) of a <a href="#">config_value</a> for use in error messages	149
<a href="#">hocon::config_parse_options</a>	
A set of options related to parsing	153
<a href="#">hocon::config_parseable</a>	
An opaque handle to something that can be parsed, obtained from <a href="#">config_include_context</a>	157
<a href="#">hocon::config_reference</a>	159
<a href="#">hocon::config_render_options</a>	167
<a href="#">hocon::config_resolve_options</a>	
A set of options related to resolving substitutions	171
<a href="#">hocon::config_string</a>	174
<a href="#">hocon::config_value</a>	
An immutable value, following the <a href="#">JSON</a> type schema	181
<a href="#">hocon::config_value_factory</a>	189
<a href="#">hocon::container</a>	
An AbstractConfigValue which contains other values	190
<a href="#">hocon::default_transformer</a>	191
<a href="#">hocon::double_slash_comment</a>	192
<a href="#">hocon::file_name_source</a>	192
<a href="#">hocon::full_includer</a>	193
<a href="#">FwdListIter&lt; T &gt;</a>	195
<a href="#">hocon::generic_exception</a>	
Exception that doesn't fall into any other category	196
<a href="#">hocon::hash_comment</a>	196
<a href="#">hocon::ignored_whitespace</a>	197
<a href="#">hocon::io_exception</a>	
Exception indicating that there was an IO error	197
<a href="#">hocon::iterator</a>	198
<a href="#">hocon::iterator_wrapper&lt; iter &gt;</a>	198
<a href="#">hocon::line</a>	199
<a href="#">List&lt; T &gt;</a>	200
<a href="#">hocon::missing_exception</a>	
Exception indicates that the setting was never set to anything, not even null	200
<a href="#">hocon::config_value::modifier</a>	201
<a href="#">hocon::name_source</a>	201
<a href="#">hocon::config_value::no_exceptions_modifier</a>	202

<a href="#">hocon::not_possible_to_resolve_exception</a>	203
<a href="#">hocon::not_resolved_exception</a>	
Exception indicating that you tried to use a function that requires substitutions to be resolved, but substitutions have not been resolved (that is, <a href="#">config#resolve</a> was not called)	203
<a href="#">hocon::null_exception</a>	
Exception indicates that the setting was treated as missing because it was set to null	204
<a href="#">OutListIter&lt; T &gt;</a>	205
<a href="#">hocon::config_document_parser::parse_context</a>	205
<a href="#">hocon::config_parser::parse_context</a>	206
<a href="#">hocon::parse_exception</a>	
Exception indicating that there was a parse error	206
<a href="#">hocon::parseable</a>	207
<a href="#">hocon::parseable_file</a>	209
<a href="#">hocon::parseable_not_found</a>	211
<a href="#">hocon::parseable_resources</a>	213
<a href="#">hocon::parseable_string</a>	215
<a href="#">hocon::path</a>	217
<a href="#">hocon::path_builder</a>	219
<a href="#">hocon::path_parser</a>	219
<a href="#">hocon::problem</a>	220
<a href="#">hocon::problem_exception</a>	220
<a href="#">hocon::relative_name_source</a>	221
<a href="#">hocon::replaceable_merge_stack</a>	221
<a href="#">hocon::resolve_context</a>	223
<a href="#">hocon::resolve_result&lt; V &gt;</a>	223
<a href="#">hocon::resolve_source</a>	224
<a href="#">hocon::resolve_source::result_with_path</a>	224
<a href="#">hocon::simple_config_document</a>	225
<a href="#">hocon::simple_config_list</a>	228
<a href="#">hocon::simple_config_object</a>	235
<a href="#">hocon::simple_config_origin</a>	245
<a href="#">hocon::simple_include_context</a>	249
<a href="#">hocon::simple_includer</a>	250
<a href="#">hocon::single_token_iterator</a>	252
<a href="#">hocon::substitution</a>	253
<a href="#">hocon::substitution_expression</a>	254
<a href="#">hocon::token</a>	254
<a href="#">hocon::token_iterator</a>	255
<a href="#">hocon::token_list_iterator</a>	255
<a href="#">hocon::tokens</a>	256
<a href="#">hocon::unmergeable</a>	
Interface that tags a ConfigValue that is not mergeable until after substitutions are resolved	257
<a href="#">hocon::unquoted_text</a>	257
<a href="#">hocon::unresolved_substitution_exception</a>	
Exception indicating that a substitution did not resolve to anything	258
<a href="#">hocon::unsupported_exception</a>	259
<a href="#">hocon::validation_failed_exception</a>	
Exception indicating that <a href="#">config#check_valid</a> found validity problems	259
<a href="#">hocon::validation_problem</a>	
Information about a problem that occurred in <a href="#">config#check_valid</a>	260
<a href="#">hocon::value</a>	260
<a href="#">hocon::wrong_type_exception</a>	
Exception indicating that the type of a value does not match the type you requested	261





## Chapter 5

# File Index

### 5.1 File List

Here is a list of all documented files with brief descriptions:

hocon/config.hpp	??
hocon/config_exception.hpp	??
hocon/config_include_context.hpp	??
hocon/config_includer.hpp	??
hocon/config_includer_file.hpp	??
hocon/config_list.hpp	??
hocon/config_mergeable.hpp	??
hocon/config_object.hpp	??
hocon/config_origin.hpp	??
hocon/config_parse_options.hpp	??
hocon/config_parseable.hpp	??
hocon/config_render_options.hpp	??
hocon/config_resolve_options.hpp	??
hocon/config_syntax.hpp	??
hocon/config_value.hpp	??
hocon/config_value_factory.hpp	??
hocon/export.h	??
hocon/functional_list.hpp	??
hocon/path.hpp	??
hocon/program_options.hpp	??
hocon/types.hpp	??
hocon/version.h	??
Declares macros for the cpp-hocon library version	263
hocon/parser/config_document.hpp	??
hocon/parser/config_document_factory.hpp	??
hocon/parser/config_node.hpp	??
internal/config_document_parser.hpp	??
internal/config_parser.hpp	??
internal/config_util.hpp	??
internal/container.hpp	??
internal/default_transformer.hpp	??
internal/full_includer.hpp	??
internal/parseable.hpp	??
internal/path_builder.hpp	??
internal/path_parser.hpp	??

internal/ <b>replaceable_merge_stack.hpp</b>	??
internal/ <b>resolve_context.hpp</b>	??
internal/ <b>resolve_result.hpp</b>	??
internal/ <b>resolve_source.hpp</b>	??
internal/ <b>simple_config_document.hpp</b>	??
internal/ <b>simple_config_origin.hpp</b>	??
internal/ <b>simple_include_context.hpp</b>	??
internal/ <b>simple_includer.hpp</b>	??
internal/ <b>substitution_expression.hpp</b>	??
internal/ <b>token.hpp</b>	??
internal/ <b>tokenizer.hpp</b>	??
internal/ <b>tokens.hpp</b>	??
internal/ <b>unmergeable.hpp</b>	??
internal/nodes/ <b>abstract_config_node.hpp</b>	??
internal/nodes/ <b>abstract_config_node_value.hpp</b>	??
internal/nodes/ <b>config_node_array.hpp</b>	??
internal/nodes/ <b>config_node_comment.hpp</b>	??
internal/nodes/ <b>config_node_complex_value.hpp</b>	??
internal/nodes/ <b>config_node_concatenation.hpp</b>	??
internal/nodes/ <b>config_node_field.hpp</b>	??
internal/nodes/ <b>config_node_include.hpp</b>	??
internal/nodes/ <b>config_node_object.hpp</b>	??
internal/nodes/ <b>config_node_path.hpp</b>	??
internal/nodes/ <b>config_node_root.hpp</b>	??
internal/nodes/ <b>config_node_simple_value.hpp</b>	??
internal/nodes/ <b>config_node_single_token.hpp</b>	??
internal/values/ <b>config_boolean.hpp</b>	??
internal/values/ <b>config_concatenation.hpp</b>	??
internal/values/ <b>config_delayed_merge.hpp</b>	??
internal/values/ <b>config_delayed_merge_object.hpp</b>	??
internal/values/ <b>config_double.hpp</b>	??
internal/values/ <b>config_int.hpp</b>	??
internal/values/ <b>config_long.hpp</b>	??
internal/values/ <b>config_null.hpp</b>	??
internal/values/ <b>config_number.hpp</b>	??
internal/values/ <b>config_reference.hpp</b>	??
internal/values/ <b>config_string.hpp</b>	??
internal/values/ <b>simple_config_list.hpp</b>	??
internal/values/ <b>simple_config_object.hpp</b>	??

## Chapter 6

# Namespace Documentation

### 6.1 hocon Namespace Reference

Factory for creating [config\\_document](#) instances.

#### Classes

- class [config](#)  
*An immutable map from config paths to config values.*
- struct [config\\_exception](#)  
*All exceptions thrown by the library are subclasses of [config\\_exception](#).*
- struct [wrong\\_type\\_exception](#)  
*Exception indicating that the type of a value does not match the type you requested.*
- struct [missing\\_exception](#)  
*Exception indicates that the setting was never set to anything, not even null.*
- struct [null\\_exception](#)  
*Exception indicates that the setting was treated as missing because it was set to null.*
- struct [bad\\_value\\_exception](#)  
*Exception indicating that a value was messed up, for example you may have asked for a duration and the value can't be sensibly parsed as a duration.*
- struct [bad\\_path\\_exception](#)  
*Exception indicating that a path expression was invalid.*
- struct [bug\\_or\\_broken\\_exception](#)  
*Exception indicating that there's a bug in something (possibly the library itself) or the runtime environment is broken.*
- struct [io\\_exception](#)  
*Exception indicating that there was an IO error.*
- struct [parse\\_exception](#)  
*Exception indicating that there was a parse error.*
- struct [unresolved\\_substitution\\_exception](#)  
*Exception indicating that a substitution did not resolve to anything.*
- struct [not\\_resolved\\_exception](#)  
*Exception indicating that you tried to use a function that requires substitutions to be resolved, but substitutions have not been resolved (that is, [config#resolve](#) was not called).*
- struct [not\\_possible\\_to\\_resolve\\_exception](#)
- struct [validation\\_problem](#)

- Information about a problem that occurred in `config#check_valid`.

  - struct `validation_failed_exception`

Exception indicating that `config#check_valid` found validity problems.
  - struct `generic_exception`

Exception that doesn't fall into any other category.
  - class `config_include_context`

Context provided to a `config_includer`; this interface is only useful inside a.
  - class `config_includer`

Implement this interface and provide an instance to `config_parse_options.set_includer()` to customize handling of.
  - class `config_includer_file`

Implement this in addition to `config_includer` if you want to support inclusion of files with the.
  - class `config_list`

Subtype of `ConfigValue` representing a list value, as in JSON's.
  - class `config_mergeable`
  - class `config_object`
  - class `config_origin`

Represents the origin (such as filename and line number) of a `config_value` for use in error messages.
  - class `config_parse_options`

A set of options related to parsing.
  - class `config_parseable`

An opaque handle to something that can be parsed, obtained from `config_include_context`.
  - class `config_render_options`
  - class `config_resolve_options`

A set of options related to resolving substitutions.
  - struct `resolve_result`
  - class `config_value`

An immutable value, following the `JSON` type schema.
  - class `config_value_factory`
  - class `config_document`

Represents an individual HOCON or JSON file, preserving all formatting and syntax details.
  - class `config_node`

A node in the syntax tree for a HOCON or JSON document.
  - class `path`
  - class `container`

An `AbstractConfigValue` which contains other values.
  - class `default_transformer`
  - class `full_includer`
  - class `abstract_config_node`
  - class `abstract_config_node_value`

This is used to classify certain `abstract_config_node` subclasses.
  - class `config_node_array`
  - class `config_node_comment`
  - class `config_node_complex_value`
  - class `config_node_concatenation`
  - class `config_node_field`

A field represents a key-value pair of the format "key : value", where key is a quoted or unquoted string, and value can be any node type.
  - class `config_node_include`

Represents an include statement of the form "include include\_kind(include\_path)".
  - class `config_node_object`
  - class `config_node_path`
  - class `config_node_root`

- class [config\\_node\\_simple\\_value](#)
- class [config\\_node\\_single\\_token](#)
- class [parseable](#)
- class [parseable\\_file](#)
- class [parseable\\_string](#)
- class [parseable\\_resources](#)
- class [parseable\\_not\\_found](#)
- class [path\\_builder](#)
- class [path\\_parser](#)
- class [replaceable\\_merge\\_stack](#)
- class [resolve\\_context](#)
- class [resolve\\_source](#)
- class [simple\\_config\\_document](#)
- class [simple\\_config\\_origin](#)
- class [simple\\_include\\_context](#)
- class [simple\\_includer](#)
- class [name\\_source](#)
- class [relative\\_name\\_source](#)
- class [file\\_name\\_source](#)
- class [substitution\\_expression](#)
- struct [unsupported\\_exception](#)
- class [token](#)
- class [problem\\_exception](#)
- class [iterator](#)
- class [iterator\\_wrapper](#)
- class [token\\_iterator](#)
- class [single\\_token\\_iterator](#)
- class [token\\_list\\_iterator](#)
- class [value](#)
- class [line](#)
- class [unquoted\\_text](#)
- class [ignored\\_whitespace](#)
- class [problem](#)
- class [comment](#)
- class [double\\_slash\\_comment](#)
- class [hash\\_comment](#)
- class [substitution](#)
- class [tokens](#)
- class [unmergeable](#)

*Interface that tags a ConfigValue that is not mergeable until after substitutions are resolved.*

- class [config\\_boolean](#)
- class [config\\_concatenation](#)

*A ConfigConcatenation represents a list of values to be concatenated (see the spec).*

- class [config\\_delayed\\_merge](#)
- class [config\\_delayed\\_merge\\_object](#)
- class [config\\_double](#)
- class [config\\_int](#)
- class [config\\_long](#)
- class [config\\_null](#)

*This exists because sometimes null is not the same as missing.*

- class [config\\_number](#)
- class [config\\_reference](#)
- class [config\\_string](#)
- class [simple\\_config\\_list](#)
- class [simple\\_config\\_object](#)

## Typedefs

- using `duration` = `std::pair< int64_t, int >`  
*A duration represented as a 64-bit integer of seconds plus a 32-bit number of nanoseconds representing a fraction of a second.*
- using `shared_config` = `std::shared_ptr< const config >`
- using `shared_object` = `std::shared_ptr< const config_object >`
- using `shared_origin` = `std::shared_ptr< const config_origin >`
- using `shared_value` = `std::shared_ptr< const config_value >`
- using `shared_list` = `std::shared_ptr< const config_list >`
- typedef `boost::recursive_variant< boost::blank, std::string, int64_t, double, int, bool, std::vector< boost::recursive_variant_ >, std::unordered_map< std::string, boost::recursive_variant_ > >::type` `unwrapped_value`
- using `shared_container` = `std::shared_ptr< const container >`
- using `shared_node` = `std::shared_ptr< const abstract_config_node >`
- using `shared_node_list` = `std::vector< shared_node >`
- using `shared_string` = `std::shared_ptr< const std::string >`
- using `shared_includer` = `std::shared_ptr< const config_includer >`
- using `shared_include_context` = `std::shared_ptr< const config_include_context >`
- using `shared_parseable` = `std::shared_ptr< const config_parseable >`
- using `shared_node_value` = `std::shared_ptr< const abstract_config_node_value >`
- using `shared_node_array` = `std::shared_ptr< const config_node_array >`
- using `shared_node_concatenation` = `std::shared_ptr< const config_node_concatenation >`
- using `shared_node_object` = `std::shared_ptr< const config_node_object >`
- using `shared_token` = `std::shared_ptr< const token >`
- using `token_list` = `std::vector< shared_token >`

## Enumerations

- enum class `time_unit` { `NANOSECONDS`, `MICROSECONDS`, `MILLISECONDS`, `SECONDS`, `MINUTES`, `HOURS`, `DAYS` }
- enum class `resolve_status` { `RESOLVED`, `UNRESOLVED` }
- enum class `config_include_kind` { `URL`, `FILE`, `CLASSPATH`, `HEURISTIC` }
- enum class `origin_type` { `GENERIC`, `FILE`, `RESOURCE` }
- enum class `token_type` { `START`, `END`, `COMMA`, `EQUALS`, `COLON`, `OPEN_CURLY`, `CLOSE_CURLY`, `OPEN_SQUARE`, `CLOSE_SQUARE`, `VALUE`, `NEWLINE`, `UNQUOTED_TEXT`, `IGNORED_WHITESPACE`, `SUBSTITUTION`, `PROBLEM`, `COMMENT`, `PLUS_EQUALS` }
- enum class `config_string_type` { `QUOTED`, `UNQUOTED` }

## Functions

- bool `operator==` (`config_document` const &lhs, `config_document` const &rhs)  
*Config documents compare via rendered strings.*
- bool `is_whitespace` (char codepoint)
- bool `is_whitespace_not_newline` (char codepoint)
- bool `is_C0_control` (char c)
- std::string `render_json_string` (std::string const &s)
- std::string `render_string_unquoted_if_possible` (std::string const &s)
- template<typename T >  
static `resolve_result`< `shared_value` > `make_resolve_result` (`resolve_context` context, T value)

### 6.1.1 Detailed Description

Factory for creating [config\\_document](#) instances.

The root namespace for cpp-hocon.

### 6.1.2 Typedef Documentation

#### 6.1.2.1 duration

```
using hocon::duration = typedef std::pair<int64_t, int>
```

A duration represented as a 64-bit integer of seconds plus a 32-bit number of nanoseconds representing a fraction of a second.

Definition at line 21 of file types.hpp.

### 6.1.3 Function Documentation

#### 6.1.3.1 operator==()

```
bool hocon::operator== (
    config_document const & lhs,
    config_document const & rhs )
```

Config documents compare via rendered strings.



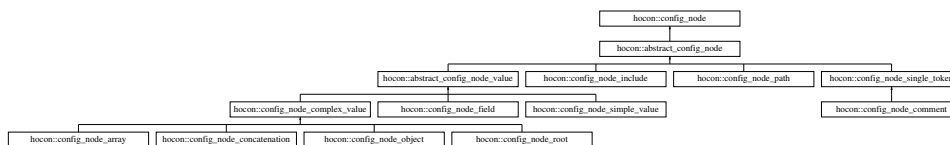


## Chapter 7

# Class Documentation

### 7.1 hocon::abstract\_config\_node Class Reference

Inheritance diagram for hocon::abstract\_config\_node:



#### Public Member Functions

- std::string [render](#) () const  
*The original text of the input which was used to form this particular node.*
- bool **operator==** (const [abstract\\_config\\_node](#) &other) const
- virtual token\_list **get\_tokens** () const =0

#### 7.1.1 Detailed Description

Definition at line 9 of file abstract\_config\_node.hpp.

#### 7.1.2 Member Function Documentation

##### 7.1.2.1 render()

```
std::string hocon::abstract_config_node::render ( ) const [virtual]
```

The original text of the input which was used to form this particular node.

##### Returns

the original text used to form this node as a String

Implements [hocon::config\\_node](#).

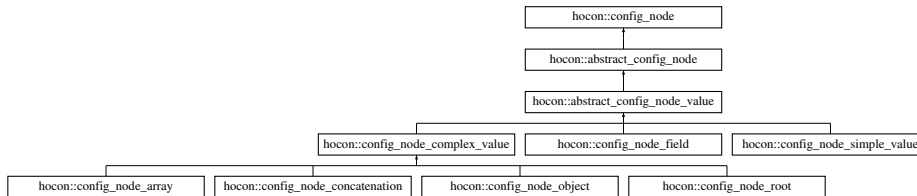
The documentation for this class was generated from the following file:

- internal/nodes/abstract\_config\_node.hpp

## 7.2 hocon::abstract\_config\_node\_value Class Reference

This is used to classify certain [abstract\\_config\\_node](#) subclasses.

Inheritance diagram for hocon::abstract\_config\_node\_value:



### Public Member Functions

- std::string [render](#) () const  
*The original text of the input which was used to form this particular node.*
- bool **operator==** (const [abstract\\_config\\_node](#) &other) const
- virtual token\_list **get\_tokens** () const =0

### 7.2.1 Detailed Description

This is used to classify certain [abstract\\_config\\_node](#) subclasses.

Definition at line 8 of file `abstract_config_node_value.hpp`.

### 7.2.2 Member Function Documentation

#### 7.2.2.1 render()

```
std::string hocon::abstract_config_node::render ( ) const [virtual], [inherited]
```

The original text of the input which was used to form this particular node.

#### Returns

the original text used to form this node as a String

Implements [hocon::config\\_node](#).

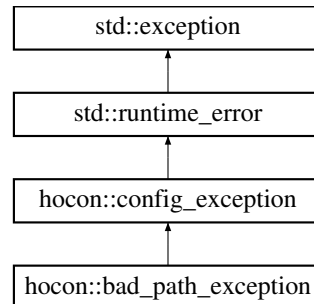
The documentation for this class was generated from the following file:

- `internal/nodes/abstract_config_node_value.hpp`

## 7.3 hocon::bad\_path\_exception Struct Reference

Exception indicating that a path expression was invalid.

Inheritance diagram for hocon::bad\_path\_exception:



### Public Member Functions

- **bad\_path\_exception** ([config\\_origin](#) const &origin, std::string const &[path](#), std::string const &message)
- **bad\_path\_exception** (std::string const &[path](#), std::string const &message)

#### 7.3.1 Detailed Description

Exception indicating that a path expression was invalid.

Try putting double quotes around path elements that contain "special" characters.

Definition at line 71 of file `config_exception.hpp`.

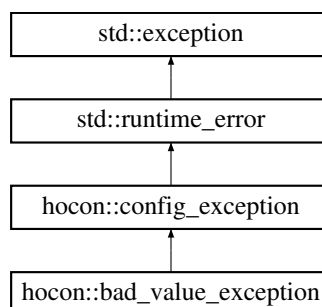
The documentation for this struct was generated from the following file:

- `hocon/config_exception.hpp`

## 7.4 hocon::bad\_value\_exception Struct Reference

Exception indicating that a value was messed up, for example you may have asked for a duration and the value can't be sensibly parsed as a duration.

Inheritance diagram for hocon::bad\_value\_exception:



## Public Member Functions

- **bad\_value\_exception** ([config\\_origin](#) const &origin, std::string const &[path](#), std::string const &message)
- **bad\_value\_exception** (std::string const &[path](#), std::string const &message)

### 7.4.1 Detailed Description

Exception indicating that a value was messed up, for example you may have asked for a duration and the value can't be sensibly parsed as a duration.

Definition at line 59 of file `config_exception.hpp`.

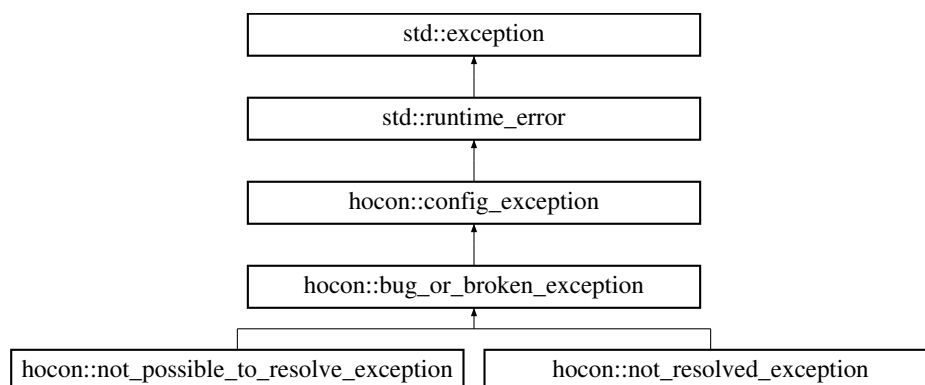
The documentation for this struct was generated from the following file:

- `hocon/config_exception.hpp`

## 7.5 hocon::bug\_or\_broken\_exception Struct Reference

Exception indicating that there's a bug in something (possibly the library itself) or the runtime environment is broken.

Inheritance diagram for `hocon::bug_or_broken_exception`:



## Public Member Functions

- **bug\_or\_broken\_exception** (std::string const &message)

### 7.5.1 Detailed Description

Exception indicating that there's a bug in something (possibly the library itself) or the runtime environment is broken.

This exception should never be handled; instead, something should be fixed to keep the exception from occurring. This exception can be thrown by any method in the library.

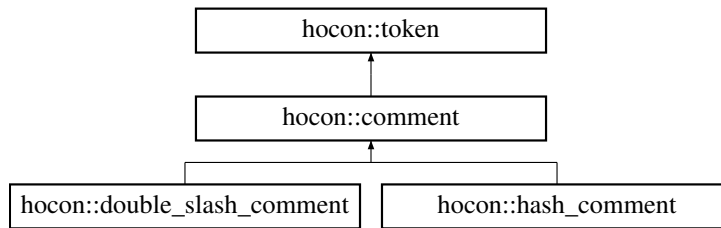
Definition at line 85 of file `config_exception.hpp`.

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

- `hocon/config_exception.hpp`

## 7.6 hocon::comment Class Reference

Inheritance diagram for hocon::comment:



### Public Member Functions

- **comment** (shared\_origin origin, std::string text)
- std::string **text** () const
- std::string **to\_string** () const override
- bool **operator==** (const token &other) const override
- virtual token\_type **get\_token\_type** () const
- virtual std::string **token\_text** () const
- virtual shared\_origin const & **origin** () const
- int **line\_number** () const

### 7.6.1 Detailed Description

Definition at line 69 of file tokens.hpp.

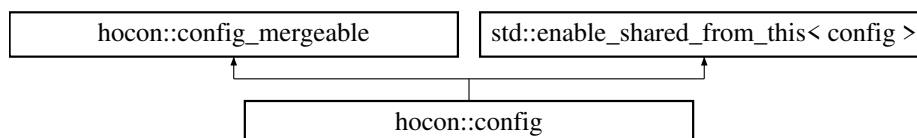
The documentation for this class was generated from the following file:

- internal/tokens.hpp

## 7.7 hocon::config Class Reference

An immutable map from config paths to config values.

Inheritance diagram for hocon::config:



## Public Member Functions

- virtual shared\_object [root](#) () const  
*Gets the.*
- virtual shared\_origin [origin](#) () const  
*Gets the origin of the.*
- std::shared\_ptr< const [config\\_mergeable](#) > [with\\_fallback](#) (std::shared\_ptr< const [config\\_mergeable](#) > other) const override  
*Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*
- shared\_value [to\\_fallback\\_value](#) () const override  
*Converts a config to its root object and a [config\\_value](#) to itself.*
- virtual shared\_config [resolve](#) () const  
*Returns a replacement config with all substitutions (the `{foo.bar}` syntax, see [the spec](#)) resolved.*
- virtual shared\_config [resolve](#) ([config\\_resolve\\_options](#) options) const  
*Like [config#resolve\(\)](#) but allows you to specify non-default options.*
- virtual bool [is\\_resolved](#) () const  
*Checks whether the config is completely resolved.*
- virtual shared\_config [resolve\\_with](#) (shared\_config source) const  
*Like [config#resolve\(\)](#) except that substitution values are looked up in the given source, rather than in this instance.*
- virtual shared\_config [resolve\\_with](#) (shared\_config source, [config\\_resolve\\_options](#) options) const  
*Like [config#resolve\\_with\(config\)](#) but allows you to specify non-default options.*
- virtual void [check\\_valid](#) (shared\_config reference, std::vector< std::string > restrict\_to\_paths) const  
*Validates this config against a reference config, throwing an exception if it is invalid.*
- virtual bool [has\\_path](#) (std::string const &[path](#)) const  
*Checks whether a value is present and non-null at the given path.*
- virtual bool [has\\_path\\_or\\_null](#) (std::string const &[path](#)) const  
*Checks whether a value is present at the given path, even if the value is null.*
- virtual bool [is\\_empty](#) () const  
*Returns true if the.*
- virtual std::set< std::pair< std::string, std::shared\_ptr< const [config\\_value](#) > > > [entry\\_set](#) () const  
*Returns the set of path-value pairs, excluding any null values, found by recursing [the root object](#).*
- virtual bool [get\\_is\\_null](#) (std::string const &[path](#)) const  
*Checks whether a value is set to null at the given path, but throws an exception if the value is entirely unset.*
- virtual bool [get\\_bool](#) (std::string const &[path](#)) const
- virtual int [get\\_int](#) (std::string const &[path](#)) const
- virtual int64\_t [get\\_long](#) (std::string const &[path](#)) const
- virtual double [get\\_double](#) (std::string const &[path](#)) const
- virtual std::string [get\\_string](#) (std::string const &[path](#)) const
- virtual std::shared\_ptr< const [config\\_object](#) > [get\\_object](#) (std::string const &[path](#)) const
- virtual shared\_config [get\\_config](#) (std::string const &[path](#)) const
- virtual unwrapped\_value [get\\_any\\_ref](#) (std::string const &[path](#)) const
- virtual std::shared\_ptr< const [config\\_value](#) > [get\\_value](#) (std::string const &[path](#)) const
- template<typename T >  
std::vector< T > [get\\_homogeneous\\_unwrapped\\_list](#) (std::string const &[path](#)) const
- virtual shared\_list [get\\_list](#) (std::string const &[path](#)) const
- virtual std::vector< bool > [get\\_bool\\_list](#) (std::string const &[path](#)) const
- virtual std::vector< int > [get\\_int\\_list](#) (std::string const &[path](#)) const
- virtual std::vector< int64\_t > [get\\_long\\_list](#) (std::string const &[path](#)) const
- virtual std::vector< double > [get\\_double\\_list](#) (std::string const &[path](#)) const
- virtual std::vector< std::string > [get\\_string\\_list](#) (std::string const &[path](#)) const
- virtual std::vector< shared\_object > [get\\_object\\_list](#) (std::string const &[path](#)) const
- virtual std::vector< shared\_config > [get\\_config\\_list](#) (std::string const &[path](#)) const

- virtual int64\_t [get\\_duration](#) (std::string const &[path](#), time\_unit unit) const  
*Gets a value as an integer number of the specified units.*
- virtual shared\_config [with\\_only\\_path](#) (std::string const &[path](#)) const  
*Clone the config with only the given path (and its children) retained; all sibling paths are removed.*
- virtual shared\_config [without\\_path](#) (std::string const &[path](#)) const  
*Clone the config with the given path removed.*
- virtual shared\_config [at\\_path](#) (std::string const &[path](#)) const  
*Places the config inside another.*
- virtual shared\_config [at\\_key](#) (std::string const &key) const  
*Places the config inside a.*
- virtual shared\_config [with\\_value](#) (std::string const &[path](#), std::shared\_ptr< const [config\\_value](#) > [value](#)) const  
*Returns a.*
- bool **operator==** ([config](#) const &other) const
- **config** (shared\_object object)
- template<> std::vector< int64\_t > **get\_homogeneous\_unwrapped\_list** (std::string const &[path](#)) const

## Static Public Member Functions

- static shared\_config [parse\\_file\\_any\\_syntax](#) (std::string file\_basename, [config\\_parse\\_options](#) options)  
*Parses a file with a flexible extension.*
- static shared\_config [parse\\_file\\_any\\_syntax](#) (std::string file\_basename)  
*Like parseFileAnySyntax(File,ConfigParseOptions) but always uses default parse options.*
- static shared\_config [parse\\_string](#) (std::string s, [config\\_parse\\_options](#) options)  
*Parses a string (which should be valid HOCON or JSON by default, or the syntax specified in the options otherwise).*
- static shared\_config [parse\\_string](#) (std::string s)  
*Parses a string (which should be valid HOCON or JSON).*
- static shared\_object **env\_variables\_as\_config\_object** ()

## Protected Member Functions

- shared\_value **find** (std::string const &path\_expression, [config\\_value::type](#) expected) const
- shared\_value **find** ([path](#) path\_expression, [config\\_value::type](#) expected, [path](#) original\_path) const
- shared\_value **find** ([path](#) path\_expression, [config\\_value::type](#) expected) const
- shared\_config **at\_key** (shared\_origin [origin](#), std::string const &key) const

## Static Protected Member Functions

- static shared\_includer **default\_includer** ()

## Friends

- class **config\_object**
- class **config\_value**
- class **config\_parseable**
- class **parseable**

### 7.7.1 Detailed Description

An immutable map from config paths to config values.

Paths are dot-separated expressions such as `foo.bar.baz`. Values are as in JSON (booleans, strings, numbers, lists, or objects), represented by `config_value` instances. Values accessed through the `config` interface are never null.

`config`

is an immutable object and thus safe to use from multiple threads. There's never a need for "defensive copies."

Fundamental operations on a

`config`

include getting configuration values, *resolving* substitutions with `config#resolve()`, and merging configs using `config#with_fallback(config_mergeable)`.

All operations return a new immutable

`config`

rather than modifying the original instance.

#### Examples

You can find an example app and library [on GitHub](#). Also be sure to read the [package overview](#) which describes the big picture as shown in those examples.

#### Paths, keys, and config vs. `config_object`

`config` is a view onto a tree of `config_object`; the corresponding object tree can be found through `config#root()`. `config_object` is a map from config *keys*, rather than paths, to config values. Think of `config_object` as a JSON object and `config` as a configuration API.

The API tries to consistently use the terms "key" and "path." A key is a key in a JSON object; it's just a string that's the key in a map. A "path" is a parseable expression with a syntax and it refers to a series of keys. Path expressions are described in the [spec for Human-Optimized Config Object Notation](#). In brief, a path is period-separated so "a.b.c" looks for key c in object b in object a in the root object. Sometimes double quotes are needed around special characters in path expressions.

The API for a

`config`

is in terms of path expressions, while the API for a

`config_object`

is in terms of keys. Conceptually,

`config`

is a one-level map from *paths* to values, while a

`config_object`

is a tree of nested maps from *keys* to values.

Use `config_util#join_path` and `config_util#split_path` to convert between path expressions and individual path elements (keys).

Another difference between

`config`

and

`config_object`



is that conceptually,  
`config_value`

s with a `value_type()` of `NULL` exist in a  
`config_object`

, while a  
`config`

treats null values as if they were missing. (With the exception of two methods: `config#has_path_or_null` and `config#get_is_null` let you detect `null` values.)

### Getting configuration values

The "getters" on a  
`config`

all work in the same way. They never return null, nor do they return a  
`config_value`

with `value_type()` of `NULL`. Instead, they throw `config_exception` if the value is completely absent or set to null. If the value is set to null, a subtype of  
`config_exception.missing`

called `config_exception.null` will be thrown. `config_exception.wrong_type` will be thrown anytime you ask for a type and the value has an incompatible type. Reasonable type conversions are performed for you though.

### Iteration

If you want to iterate over the contents of a  
`config`

, you can get its  
`config_object`

with `root()`, and then iterate over the  
`config_object`

(which implements `java.util.Map`). Or, you can use `entry_set()` which recurses the object tree for you and builds up a `set` of all path-value pairs where the value is not null.

### Resolving substitutions

*Substitutions* are the `${foo.bar}` syntax in config files, described in the [specification](#). Resolving substitutions replaces these references with real values.

Before using a  
`config`

it's necessary to call `config#resolve()` to handle substitutions (though `config_factory#load()` and similar methods will do the resolve for you already).

### Merging

The full `config` for your application can be constructed using the associative operation `config#with_fallback(config_mergeable)`. If you use `config_factory#load()` (recommended), it merges system properties over the top of `application.conf` over the top of `reference.conf`, using `with_fallback`. You can add in additional sources of configuration in the same way (usually, custom layers should go either just above or just below `application.conf`, keeping `reference.conf` at the bottom and system properties at the top).

### Serialization

Convert a `config` to a JSON or HOCON string by calling [config\\_object#render\(\)](#) on the root object, `my_↵ config.root().render()`. There's also a variant `config_object#render(config_render_options)` which allows you to control the format of the rendered string. (See [config\\_render\\_options](#).) Note that `config` does not remember the formatting of the original file, so if you load, modify, and re-save a config file, it will be substantially reformatted.

As an alternative to [config\\_object#render\(\)](#), the `to_string()` method produces a debug-output-oriented representation (which is not valid JSON).

### This is an interface but don't implement it yourself

*Do not implement*  
`config`

; it should only be implemented by the config library. Arbitrary implementations will not work because the library internals assume a specific concrete implementation. Also, this interface is likely to grow new methods over time, so third-party implementations will break.

Definition at line 172 of file `config.hpp`.

## 7.7.2 Member Function Documentation

### 7.7.2.1 at\_key()

```
virtual shared_config hocon::config::at_key (
    std::string const & key ) const [virtual]
```

Places the config inside a.  
`config`

at the given key. See also [at\\_path\(\)](#). Note that a key is NOT a path expression (see `config_util#join_path` and `config_util#split_path`).

#### Parameters

<i>key</i>	key to store this config at.
------------	------------------------------

#### Returns

a  
`config`

instance containing this config at the given key.

### 7.7.2.2 at\_path()

```
virtual shared_config hocon::config::at_path (
    std::string const & path ) const [virtual]
```

Places the config inside another.  
`config`

at the given path.

Note that path expressions have a syntax and sometimes require quoting (see `config_util#join_path` and `config_util#split_path`).

#### Parameters

<i>path</i>	path expression to store this config at.
-------------	--

#### Returns

`a`  
`config`  
 instance containing this config at the given path.

#### 7.7.2.3 check\_valid()

```
virtual void hocon::config::check_valid (
    shared_config reference,
    std::vector< std::string > restrict_to_paths ) const [virtual]
```

Validates this config against a reference config, throwing an exception if it is invalid.

The purpose of this method is to "fail early" with a comprehensive list of problems; in general, anything this method can find would be detected later when trying to use the config, but it's often more user-friendly to fail right away when loading the config.

Using this method is always optional, since you can "fail late" instead.

You must restrict validation to paths you "own" (those whose meaning are defined by your code module). If you validate globally, you may trigger errors about paths that happen to be in the config but have nothing to do with your module. It's best to allow the modules owning those paths to validate them. Also, if every module validates only its own stuff, there isn't as much redundant work being done.

If no paths are specified in `check_valid()`'s parameter list, validation is for the entire config.

If you specify paths that are not in the reference config, those paths are ignored. (There's nothing to validate.)

Here's what validation involves:

- All paths found in the reference config must be present in this config or an exception will be thrown.
- Some changes in type from the reference config to this config will cause an exception to be thrown. Not all potential type problems are detected, in particular it's assumed that strings are compatible with everything except objects and lists. This is because string types are often "really" some other type (system properties always start out as strings, or a string like "5ms" could be used with `get_milliseconds`). Also, it's allowed to set any type to null or override null with any type.

- Any unresolved substitutions in this config will cause a validation failure; both the reference config and this config should be resolved before validation. If the reference config is unresolved, it's a bug in the caller of this method.

If you want to allow a certain setting to have a flexible type (or otherwise want validation to be looser for some settings), you could either remove the problematic setting from the reference config provided to this method, or you could intercept the validation exception and screen out certain problems. Of course, this will only work if all other callers of this method are careful to restrict validation to their own paths, as they should be.

If validation fails, the thrown exception contains a list of all problems found. The exception will have all the problem concatenated into one huge string.

Again, `check_valid()` can't guess every domain-specific way a setting can be invalid, so some problems may arise later when attempting to use the config. `check_valid()` is limited to reporting generic, but common, problems such as missing settings and blatant type incompatibilities.

#### Parameters

<i>reference</i>	a reference configuration
<i>restrictToPaths</i>	only validate values underneath these paths that your code module owns and understands

#### 7.7.2.4 entry\_set()

```
virtual std::set<std::pair<std::string, std::shared_ptr<const config_value> > > hocon<
::config::entry_set ( ) const [virtual]
```

Returns the set of path-value pairs, excluding any null values, found by recursing [the root object](#).

Note that this is very different from `root().entry_set()` which returns the set of immediate-child keys in the root object and includes null values.

Entries contain *path expressions* meaning there may be quoting and escaping involved. Parse path expressions with `config_util#split_path`.

Because a `config` is conceptually a single-level map from paths to values, there will not be any `config_object` values in the entries (that is, all entries represent leaf nodes). Use `config_object` rather than `config` if you want a tree. (OK, this is a slight lie: `config` entries may contain `config_list` and the lists may contain objects. But no objects are directly included as entry values.)

#### Returns

set of paths with non-null values, built up by recursing the entire tree of `config_object` and creating an entry for each leaf value.

#### 7.7.2.5 get\_duration()

```
virtual int64_t hocon::config::get_duration (
    std::string const & path,
    time_unit unit ) const [virtual]
```

Gets a value as an integer number of the specified units.

If the result would have a fractional part, the number is truncated. Correctly handles durations within the range  $\pm 2^{63}$  seconds.

## Parameters

<i>path</i>	the path to the time value
<i>unit</i>	the units of the number returned

## Returns

a 64-bit integer representing the value converted to the requested units

## 7.7.2.6 get\_is\_null()

```
virtual bool hocon::config::get_is_null (
    std::string const & path ) const [virtual]
```

Checks whether a value is set to null at the given path, but throws an exception if the value is entirely unset.

This method will not throw if {} returned true for the same path, so to avoid any possible exception check `has_path_or_null()` if possible. Note that path expressions have a syntax and sometimes require quoting (see `config_util#join_path` and `config_util#split_path`).

## Parameters

<i>path</i>	the path expression
-------------	---------------------

## Returns

true if the value exists and is null, false if it exists and is not null

## 7.7.2.7 has\_path()

```
virtual bool hocon::config::has_path (
    std::string const & path ) const [virtual]
```

Checks whether a value is present and non-null at the given path.

This differs in two ways from

`Map.containsKey()`

as implemented by [config\\_object](#): it looks for a path expression, not a key; and it returns false for null values, while `contains_key()`

returns true indicating that the object contains a null value for the key.

If a path exists according to `has_path(string)`, then `get_value(string)` will never throw an exception. However, the typed getters will still throw if the value is not convertible to the requested type.

Note that path expressions have a syntax and sometimes require quoting (see `config_util#join_path` and `config_util#split_path`).

**Parameters**

<i>path</i>	the path expression
-------------	---------------------

**Returns**

true if a non-null value is present at the path

**7.7.2.8 has\_path\_or\_null()**

```
virtual bool hocon::config::has_path_or_null (
    std::string const & path ) const [virtual]
```

Checks whether a value is present at the given path, even if the value is null.

Most of the getters on `config` will throw if you try to get a null value, so if you plan to call `get_value(string)`, `get_int(string)`, or another getter you may want to use plain `has_path(string)` rather than this method.

To handle all three cases (unset, null, and a non-null value) the code might look like:

```
if (config.has_path_or_null(path)) {
    if (config.get_is_null(path)) {
        // handle null setting
    } else {
        // get and use non-null setting
    }
} else {
    // handle entirely unset path
}
```

However, the usual thing is to allow entirely unset paths to be a bug that throws an exception (because you set a default in your `reference.conf`), so in that case it's OK to call `get_is_null(string)` without checking `has_path_or_null` first.

Note that path expressions have a syntax and sometimes require quoting (see `config_util#join_path` and `config_util#split_path`).

**Parameters**

<i>path</i>	the path expression
-------------	---------------------

**Returns**

true if a value is present at the path, even if the value is null

### 7.7.2.9 is\_empty()

```
virtual bool hocon::config::is_empty ( ) const [virtual]
```

Returns true if the.

`Config`

's root object contains no key-value pairs.

#### Returns

true if the configuration is empty

### 7.7.2.10 is\_resolved()

```
virtual bool hocon::config::is_resolved ( ) const [virtual]
```

Checks whether the config is completely resolved.

After a successful call to [config#resolve\(\)](#) it will be completely resolved, but after calling `config#resolve(config_resolve_options)` with `allow_unresolved` set in the options, it may or may not be completely resolved. A newly-loaded config may or may not be completely resolved depending on whether there were substitutions present in the file.

#### Returns

true if there are no unresolved substitutions remaining in this configuration.

### 7.7.2.11 origin()

```
virtual shared_origin hocon::config::origin ( ) const [virtual]
```

Gets the origin of the.

`Config`

, which may be a file, or a file with a line number, or just a descriptive phrase.

#### Returns

the origin of the

`Config`

for use in error messages

### 7.7.2.12 parse\_file\_any\_syntax() [1/2]

```
static shared_config hocon::config::parse_file_any_syntax (
    std::string file_basename ) [static]
```

Like `parseFileAnySyntax(File, ConfigParseOptions)` but always uses default parse options.

**Parameters**

<i>fileBaseline</i>	a filename with or without extension
---------------------	--------------------------------------

**Returns**

the parsed configuration

**7.7.2.13 parse\_file\_any\_syntax() [2/2]**

```
static shared_config hocon::config::parse_file_any_syntax (
    std::string file_basename,
    config_parse_options options ) [static]
```

Parses a file with a flexible extension.

If the *fileBaseline* already ends in a known extension, this method parses it according to that extension (the file's syntax must match its extension). If the *fileBaseline* does not end in an extension, it parses files with all known extensions and merges whatever is found.

In the current implementation, the extension ".conf" forces ConfigSyntax#CONF, ".json" forces ConfigSyntax#JSON. When merging files, ".conf" falls back to ".json".

Future versions of the implementation may add additional syntaxes or additional extensions. However, the ordering (fallback priority) of the three current extensions will remain the same.

If *options* forces a specific syntax, this method only parses files with an extension matching that syntax.

If *options.getAllowMissing()* is true, then no files have to exist; if false, then at least one file has to exist.

**Parameters**

<i>fileBaseline</i>	a filename with or without extension
<i>options</i>	parse options

**Returns**

the parsed configuration

**7.7.2.14 parse\_string() [1/2]**

```
static shared_config hocon::config::parse_string (
    std::string s ) [static]
```

Parses a string (which should be valid HOCON or JSON).



## Parameters

<i>s</i>	string to parse
----------	-----------------

## Returns

the parsed configuration

7.7.2.15 `parse_string()` [2/2]

```
static shared_config hocon::config::parse_string (
    std::string s,
    config_parse_options options ) [static]
```

Parses a string (which should be valid HOCON or JSON by default, or the syntax specified in the options otherwise).

## Parameters

<i>s</i>	string to parse
<i>options</i>	parse options

## Returns

the parsed configuration

7.7.2.16 `resolve()` [1/2]

```
virtual shared_config hocon::config::resolve ( ) const [virtual]
```

Returns a replacement config with all substitutions (the `${foo.bar}` syntax, see [the spec](#)) resolved.

Substitutions are looked up using this `config` as the root object, that is, a substitution `${foo.bar}` will be replaced with the result of `get_value("foo.bar")`.

This method uses `config_resolve_options()`, there is another variant `config#resolve(config_resolve_options)` which lets you specify non-default options.

A given `config` must be resolved before using it to retrieve config values, but ideally should be resolved one time for your entire stack of fallbacks (see [config#with\\_fallback](#)). Otherwise, some substitutions that could have resolved with all fallbacks available may not resolve, which will be potentially confusing for your application's users.

`resolve()` should be invoked on root config objects, rather than on a subtree (a subtree is the result of something like `config.get_config("foo")`). The problem with `resolve()` on a subtree is that substitutions are relative to the root of the config and the subtree will have no way to get values from the root. For example, if you did `config.get_config("foo").resolve()` on the below config file, it would not work:

```
common-value = 10
foo {
  whatever = ${common-value}
}
```

Many methods on `config_factory` such as `config_factory#load()` automatically resolve the loaded `config` on the loaded stack of config files.

Resolving an already-resolved config is a harmless no-op, but again, it is best to resolve an entire stack of fallbacks (such as all your config files combined) rather than resolving each one individually.

#### Returns

an immutable object with substitutions resolved

#### 7.7.2.17 `resolve()` [2/2]

```
virtual shared_config hocon::config::resolve (
    config_resolve_options options ) const [virtual]
```

Like [config#resolve\(\)](#) but allows you to specify non-default options.

#### Parameters

<i>options</i>	resolve options
----------------	-----------------

#### Returns

the resolved `config` (may be only partially resolved if options are set to allow unresolved)

#### 7.7.2.18 `resolve_with()` [1/2]

```
virtual shared_config hocon::config::resolve_with (
    shared_config source ) const [virtual]
```

Like [config#resolve\(\)](#) except that substitution values are looked up in the given source, rather than in this instance.

This is a special-purpose method which doesn't make sense to use in most cases; it's only needed if you're constructing some sort of app-specific custom approach to configuration. The more usual approach if you have a source of substitution values would be to merge that source into your config stack using `config#withFallback` and then resolve.

Note that this method does NOT look in this instance for substitution values. If you want to do that, you could either merge this instance into your value source using [config#with\\_fallback](#), or you could resolve multiple times with multiple sources (using `config_resolve_options#setAllowUnresolved(boolean)` so the partial resolves don't fail).

**Parameters**

<i>source</i>	configuration to pull values from
---------------	-----------------------------------

**Returns**

an immutable object with substitutions resolved

**7.7.2.19 resolve\_with() [2/2]**

```
virtual shared_config hocon::config::resolve_with (
    shared_config source,
    config_resolve_options options ) const [virtual]
```

Like `config#resolve_with(config)` but allows you to specify non-default options.

**Parameters**

<i>source</i>	source configuration to pull values from
<i>options</i>	resolve options

**Returns**

the resolved `config` (may be only partially resolved if options are set to allow unresolved)

**7.7.2.20 root()**

```
virtual shared_object hocon::config::root ( ) const [virtual]
```

Gets the.

`Config`

as a tree of `ConfigObject`. This is a constant-time operation (it is not proportional to the number of values in the `Config`

).

**Returns**

the root object in the configuration

### 7.7.2.21 to\_fallback\_value()

```
shared_value hocon::config::to_fallback_value ( ) const [override], [virtual]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

### 7.7.2.22 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

#### Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

**Returns**

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

**7.7.2.23 with\_only\_path()**

```
virtual shared_config hocon::config::with_only_path (
    std::string const & path ) const [virtual]
```

Clone the config with only the given path (and its children) retained; all sibling paths are removed.

Note that path expressions have a syntax and sometimes require quoting (see [config\\_util#join\\_path](#) and [config\\_util#split\\_path](#)).

**Parameters**

<i>path</i>	path to keep
-------------	--------------

**Returns**

a copy of the config minus all paths except the one specified

**7.7.2.24 with\_value()**

```
virtual shared_config hocon::config::with_value (
    std::string const & path,
    std::shared_ptr< const config_value > value ) const [virtual]
```

Returns a.

*config*

based on this one, but with the given path set to the given value. Does not modify this instance (since it's immutable). If the path already has a value, that value is replaced. To remove a value, use [withoutPath\(\)](#).

Note that path expressions have a syntax and sometimes require quoting (see [config\\_util#join\\_path](#) and [config\\_util#split\\_path](#)).

**Parameters**

<i>path</i>	path expression for the value's new location
<i>value</i>	value at the new path

**Returns**

the new instance with the new map entry

**7.7.2.25 without\_path()**

```
virtual shared_config hocon::config::without_path (
    std::string const & path ) const [virtual]
```

Clone the config with the given path removed.

Note that path expressions have a syntax and sometimes require quoting (see `config_util#join_path` and `config_util#split_path`).

**Parameters**

<i>path</i>	path expression to remove
-------------	---------------------------

**Returns**

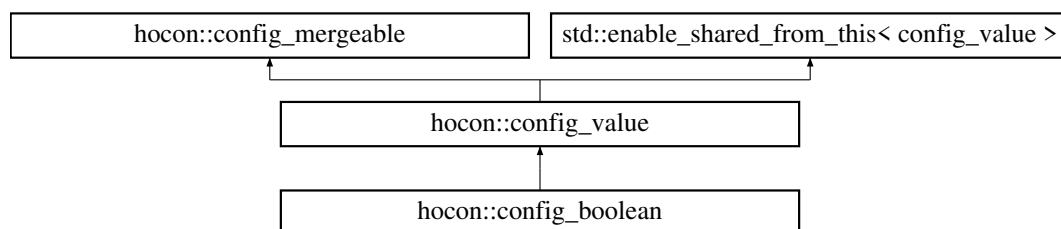
a copy of the config minus the specified path

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

- `hocon/config.hpp`

**7.8 hocon::config\_boolean Class Reference**

Inheritance diagram for `hocon::config_boolean`:

**Public Types**

- enum class `type` {  
**OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
**CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

*The type of a configuration value (following the `JSON` type schema).*

## Public Member Functions

- **config\_boolean** (shared\_origin [origin](#), bool [value](#))
- [config\\_value::type value\\_type](#) () const override  
*The type of the value; matches the JSON type schema.*
- std::string **transform\_to\_string** () const override
- unwrapped\_value **unwrapped** () const override
- bool **bool\_value** () const
- bool **operator==** ([config\\_value](#) const &other) const override
- virtual shared\_origin const & [origin](#) () const  
*The origin of the value (file, line number, etc.), for debugging and error messages.*
- char const \* [value\\_type\\_name](#) () const  
*The printable name of the value type.*
- virtual std::string [render](#) () const  
*Renders the config value as a HOCON string.*
- virtual std::string [render](#) ([config\\_render\\_options](#) options) const  
*Renders the config value to a string, using the provided options.*
- shared\_config [at\\_key](#) (std::string const &key) const  
*Places the value inside a [config](#) at the given key.*
- shared\_config [at\\_path](#) (std::string const &path\_expression) const  
*Places the value inside a [config](#) at the given path.*
- virtual shared\_value [with\\_origin](#) (shared\_origin [origin](#)) const  
*Returns a.*
- virtual shared\_value [relativized](#) (std::string prefix) const  
*This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.*
- virtual resolve\_status **get\_resolve\_status** () const
- std::shared\_ptr< const [config\\_mergeable](#) > [with\\_fallback](#) (std::shared\_ptr< const [config\\_mergeable](#) > other) const override  
*Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*

## Static Public Member Functions

- static char const \* **type\_name** ([type](#) t)

## Protected Member Functions

- shared\_value **new\_copy** (shared\_origin) const override
- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &[at\\_key](#), [config\\_render\\_options](#) options) const
- virtual void **render** (std::string &result, int indent, bool at\_root, [config\\_render\\_options](#) options) const
- shared\_config **at\_key** (shared\_origin [origin](#), std::string const &key) const
- shared\_config **at\_path** (shared\_origin [origin](#), [path](#) raw\_path) const
- virtual [resolve\\_result](#)< shared\_value > **resolve\_substitutions** ([resolve\\_context](#) const &context, [resolve\\_source](#) const &source) const
- void **require\_not\_ignoring\_fallbacks** () const
- virtual bool **ignores\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const [unmergeable](#) > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const [unmergeable](#) > fallback) const

- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- virtual shared\_value **construct\_delayed\_merge** (shared\_origin [origin](#), std::vector< shared\_value > stack) const
- shared\_value [to\\_fallback\\_value](#) () const override

*Converts a config to its root object and a [config\\_value](#) to itself.*

## Static Protected Member Functions

- static void **indent** (std::string &result, int indent, [config\\_render\\_options](#) const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** ([config\\_value](#) const &other, std::function< bool(T const &)> checker)

### 7.8.1 Detailed Description

Definition at line 7 of file config\_boolean.hpp.

### 7.8.2 Member Enumeration Documentation

#### 7.8.2.1 type

```
enum hocon::config_value::type [strong], [inherited]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file config\_value.hpp.

### 7.8.3 Member Function Documentation

#### 7.8.3.1 at\_key()

```
shared_config hocon::config_value::at_key (
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also [config\\_value#at\\_path\(string\)](#).



## Parameters

<i>key</i>	key to store this value at.
------------	-----------------------------

## Returns

a  
config  
instance containing this value at the given key.

## 7.8.3.2 at\_path()

```
shared_config hocon::config_value::at_path (  
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

## Parameters

<i>path</i>	path to store this value at.
-------------	------------------------------

## Returns

a  
config  
instance containing this value at the given path.

## 7.8.3.3 origin()

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

## Returns

where the value came from

#### 7.8.3.4 relativized()

```
virtual shared_value hocon::config_value::relativized (
    std::string prefix ) const    [inline], [virtual], [inherited]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at `foo.bar` in the parent, and the included file as a substitution `${a.b.c}`, the included substitution now needs to be `${foo.bar.a.b.c}` because we resolve substitutions globally only after parsing everything.

## Parameters

<i>prefix</i>	
---------------	--

## Returns

value relativized to the given path or the same value if nothing to do

Reimplemented in [hocon::config\\_concatenation](#), and [hocon::simple\\_config\\_list](#).

Definition at line 181 of file config\_value.hpp.

### 7.8.3.5 render() [1/2]

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to

```
render(config_render_options())
```

.

## Returns

the rendered value

### 7.8.3.6 render() [2/2]

```
virtual std::string hocon::config_value::render (
    config_render_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

**Parameters**

<i>options</i>	the rendering options
----------------	-----------------------

**Returns**

the rendered value

**7.8.3.7 to\_fallback\_value()**

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],  
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

**7.8.3.8 value\_type()**

```
config_value::type hocon::config_boolean::value_type ( ) const [override], [virtual]
```

The type of the value; matches the JSON type schema.

**Returns**

value's type

Implements [hocon::config\\_value](#).

**7.8.3.9 value\_type\_name()**

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

**Returns**

value's type's name

Definition at line 92 of file config\_value.hpp.

### 7.8.3.10 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

#### Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

#### Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

### 7.8.3.11 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```

Returns a.  
`config_value`

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single `config_value`.

#### Parameters

<code>origin</code>	the origin set on the returned value
---------------------	--------------------------------------

#### Returns

the new `config_value` with the given origin

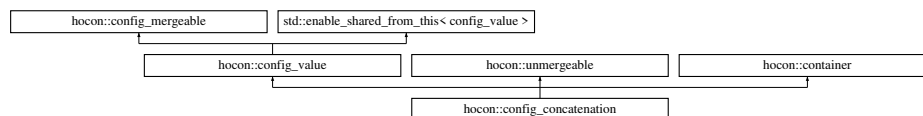
The documentation for this class was generated from the following file:

- `internal/values/config_boolean.hpp`

## 7.9 hocon::config\_concatenation Class Reference

A ConfigConcatenation represents a list of values to be concatenated (see the spec).

Inheritance diagram for `hocon::config_concatenation`:



### Public Types

- enum class `type` {  
**OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
**CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

*The type of a configuration value (following the `JSON` type schema).*

### Public Member Functions

- `config_concatenation` (shared\_origin `origin`, std::vector< shared\_value > pieces)
- `config_value::type value_type` () const override  
*The type of the value; matches the JSON type schema.*
- std::vector< shared\_value > `unmerged_values` () const override
- resolve\_status `get_resolve_status` () const override
- shared\_value `replace_child` (shared\_value const &child, shared\_value replacement) const override  
*Replace a child of this value.*
- bool `has_descendant` (shared\_value const &descendant) const override  
*Super-expensive full traversal to see if descendant is anywhere underneath this container.*
- `resolve_result`< shared\_value > `resolve_substitutions` (`resolve_context` const &context, `resolve_source` const &source) const override

- shared\_value **relativized** (std::string prefix) const override  
*This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.*
- unwrapped\_value **unwrapped** () const override
- bool **operator==** (config\_value const &other) const override
- virtual shared\_origin const & **origin** () const  
*The origin of the value (file, line number, etc.), for debugging and error messages.*
- char const \* **value\_type\_name** () const  
*The printable name of the value type.*
- virtual std::string **render** () const  
*Renders the config value as a HOCON string.*
- virtual std::string **render** (config\_render\_options options) const  
*Renders the config value to a string, using the provided options.*
- shared\_config **at\_key** (std::string const &key) const  
*Places the value inside a config at the given key.*
- shared\_config **at\_path** (std::string const &path\_expression) const  
*Places the value inside a config at the given path.*
- virtual shared\_value **with\_origin** (shared\_origin origin) const  
*Returns a.*
- std::shared\_ptr< const config\_mergeable > **with\_fallback** (std::shared\_ptr< const config\_mergeable > other) const override  
*Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*
- virtual std::string **transform\_to\_string** () const

## Static Public Member Functions

- static std::vector< shared\_value > **consolidate** (std::vector< shared\_value > pieces)
- static shared\_value **concatenate** (std::vector< shared\_value > pieces)
- static char const \* **type\_name** (type t)

## Protected Member Functions

- shared\_value **new\_copy** (shared\_origin origin) const override
- bool **ignores\_fallbacks** () const override
- void **render** (std::string &result, int indent, bool at\_root, config\_render\_options options) const override
- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &at\_key, config\_render\_options options) const
- shared\_config **at\_key** (shared\_origin origin, std::string const &key) const
- shared\_config **at\_path** (shared\_origin origin, path raw\_path) const
- void **require\_not\_ignoring\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const unmergeable > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const unmergeable > fallback) const
- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- virtual shared\_value **construct\_delayed\_merge** (shared\_origin origin, std::vector< shared\_value > stack) const
- shared\_value **to\_fallback\_value** () const override  
*Converts a config to its root object and a config\_value to itself.*

## Static Protected Member Functions

- static void **indent** (std::string &result, int indent, [config\\_render\\_options](#) const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** ([config\\_value](#) const &other, std::function< bool(T const &)> checker)

### 7.9.1 Detailed Description

A ConfigConcatenation represents a list of values to be concatenated (see the spec).

It only has to exist if at least one value is an unresolved substitution, otherwise we could go ahead and collapse the list into a single value.

Right now this is always a list of strings and `${}` references, but in the future should support a list of ConfigList. We may also support concatenations of objects, but ConfigDelayedMerge should be used for that since a concat of objects really will merge, not concatenate.

Definition at line 24 of file `config_concatenation.hpp`.

### 7.9.2 Member Enumeration Documentation

#### 7.9.2.1 type

```
enum hocon::config\_value::type [strong], [inherited]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file `config_value.hpp`.

### 7.9.3 Member Function Documentation

#### 7.9.3.1 at\_key()

```
shared_config hocon::config_value::at_key (  
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also `config_value#at_path(string)`.



## Parameters

<i>key</i>	key to store this value at.
------------	-----------------------------

## Returns

a  
config  
instance containing this value at the given key.

## 7.9.3.2 at\_path()

```
shared_config hocon::config_value::at_path (
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

## Parameters

<i>path</i>	path to store this value at.
-------------	------------------------------

## Returns

a  
config  
instance containing this value at the given path.

## 7.9.3.3 has\_descendant()

```
bool hocon::config_concatenation::has_descendant (
    shared_value const & descendant ) const [override], [virtual]
```

Super-expensive full traversal to see if descendant is anywhere underneath this container.

Implements [hocon::container](#).

## 7.9.3.4 origin()

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

## Returns

where the value came from

### 7.9.3.5 relativized()

```
shared_value hocon::config_concatenation::relativized (
    std::string prefix ) const [override], [virtual]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at `foo.bar` in the parent, and the included file as a substitution `${a.b.c}`, the included substitution now needs to be `${foo.bar.a.b.c}` because we resolve substitutions globally only after parsing everything.

#### Parameters

<i>prefix</i>	
---------------	--

#### Returns

value relativized to the given path or the same value if nothing to do

Reimplemented from [hocon::config\\_value](#).

### 7.9.3.6 render() [1/2]

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to  
[render](#)([config\\_render\\_options](#)())

.

#### Returns

the rendered value

### 7.9.3.7 render() [2/2]

```
virtual std::string hocon::config_value::render (
    config_render_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

## Parameters

<i>options</i>	the rendering options
----------------	-----------------------

## Returns

the rendered value

**7.9.3.8 replace\_child()**

```
shared_value hocon::config_concatenation::replace_child (
    shared_value const & child,
    shared_value replacement ) const [override], [virtual]
```

Replace a child of this value.

CAUTION if replacement is null, delete the child, which may also delete the parent, or make the parent into a non-container.

Implements [hocon::container](#).

**7.9.3.9 to\_fallback\_value()**

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

**7.9.3.10 value\_type()**

```
config\_value::type hocon::config_concatenation::value_type ( ) const [override], [virtual]
```

The type of the value; matches the JSON type schema.

## Returns

value's type

Implements [hocon::config\\_value](#).

### 7.9.3.11 value\_type\_name()

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

#### Returns

value's type's name

Definition at line 92 of file config\_value.hpp.

### 7.9.3.12 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

## Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

## Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

## 7.9.3.13 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```

Returns a.  
`config_value`

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single [config\\_value](#).

## Parameters

<i>origin</i>	the origin set on the returned value
---------------	--------------------------------------

## Returns

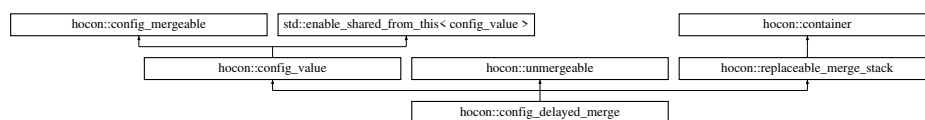
the new [config\\_value](#) with the given origin

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

- `internal/values/config_concatenation.hpp`

## 7.10 hocon::config\_delayed\_merge Class Reference

Inheritance diagram for `hocon::config_delayed_merge`:



## Public Types

- enum class [type](#) {  
**OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
**CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

*The type of a configuration value (following the [JSON](#) type schema).*

## Public Member Functions

- **config\_delayed\_merge** (shared\_origin [origin](#), std::vector< shared\_value > stack)
- [config\\_value::type value\\_type](#) () const override  
*The type of the value; matches the JSON type schema.*
- shared\_value **make\_replacement** ([resolve\\_context](#) const &context, int skipping) const override
- std::vector< shared\_value > **unmerged\_values** () const override
- unwrapped\_value **unwrapped** () const override
- [resolve\\_result](#)< shared\_value > **resolve\_substitutions** ([resolve\\_context](#) const &context, [resolve\\_source](#) const &source) const override
- resolve\_status **get\_resolve\_status** () const override
- bool **operator==** ([config\\_value](#) const &other) const override
- shared\_value [replace\\_child](#) (shared\_value const &child, shared\_value replacement) const override  
*Replace a child of this value.*
- bool [has\\_descendant](#) (shared\_value const &descendant) const override  
*Super-expensive full traversal to see if descendant is anywhere underneath this container.*
- virtual shared\_origin const & [origin](#) () const  
*The origin of the value (file, line number, etc.), for debugging and error messages.*
- char const \* [value\\_type\\_name](#) () const  
*The printable name of the value type.*
- virtual std::string [render](#) () const  
*Renders the config value as a HOCON string.*
- virtual std::string [render](#) ([config\\_render\\_options](#) options) const  
*Renders the config value to a string, using the provided options.*
- shared\_config [at\\_key](#) (std::string const &key) const  
*Places the value inside a [config](#) at the given key.*
- shared\_config [at\\_path](#) (std::string const &path\_expression) const  
*Places the value inside a [config](#) at the given path.*
- virtual shared\_value [with\\_origin](#) (shared\_origin [origin](#)) const  
*Returns a.*
- virtual shared\_value [relativized](#) (std::string prefix) const  
*This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.*
- std::shared\_ptr< const [config\\_mergeable](#) > [with\\_fallback](#) (std::shared\_ptr< const [config\\_mergeable](#) > other) const override  
*Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*
- virtual std::string **transform\_to\_string** () const

## Static Public Member Functions

- static shared\_value **make\_replacement** ([resolve\\_context](#) const &context, std::vector< shared\_value > stack, int skipping)
- static [resolve\\_result](#)< shared\_value > **resolve\_substitutions** (std::shared\_ptr< const [replaceable\\_merge\\_stack](#) > replaceable, const std::vector< shared\_value > &\_stack, [resolve\\_context](#) const &context, [resolve\\_source](#) const &source)
- static void **render** (std::vector< shared\_value > const &stack, std::string &s, int indent\_value, bool at\_root, std::string const &[at\\_key](#), [config\\_render\\_options](#) options)
- static char const \* **type\_name** (type t)

## Protected Member Functions

- shared\_value **new\_copy** (shared\_origin) const override
- bool **ignores\_fallbacks** () const override
- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &at\_key, config\_render\_options options) const override
- virtual void **render** (std::string &result, int indent, bool at\_root, config\_render\_options options) const override
- shared\_config **at\_key** (shared\_origin origin, std::string const &key) const
- shared\_config **at\_path** (shared\_origin origin, path raw\_path) const
- void **require\_not\_ignoring\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const unmergeable > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const unmergeable > fallback) const
- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- virtual shared\_value **construct\_delayed\_merge** (shared\_origin origin, std::vector< shared\_value > stack) const
- shared\_value **to\_fallback\_value** () const override

*Converts a config to its root object and a config\_value to itself.*

## Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config\_render\_options const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** (config\_value const &other, std::function< bool(T const &)> checker)

### 7.10.1 Detailed Description

Definition at line 11 of file config\_delayed\_merge.hpp.

### 7.10.2 Member Enumeration Documentation

#### 7.10.2.1 type

```
enum hocon::config_value::type [strong], [inherited]
```

The type of a configuration value (following the JSON type schema).

Definition at line 56 of file config\_value.hpp.

### 7.10.3 Member Function Documentation

#### 7.10.3.1 at\_key()

```
shared_config hocon::config_value::at_key (
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also `config_value#at_path(string)`.

##### Parameters

<i>key</i>	key to store this value at.
------------	-----------------------------

##### Returns

a  
`config`

instance containing this value at the given key.

#### 7.10.3.2 at\_path()

```
shared_config hocon::config_value::at_path (
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

##### Parameters

<i>path</i>	path to store this value at.
-------------	------------------------------

##### Returns

a  
`config`

instance containing this value at the given path.

#### 7.10.3.3 has\_descendant()

```
bool hocon::config_delayed_merge::has_descendant (
    shared_value const & descendant ) const [override], [virtual]
```



Super-expensive full traversal to see if descendant is anywhere underneath this container.

Implements [hocon::container](#).

#### 7.10.3.4 origin()

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

##### Returns

where the value came from

#### 7.10.3.5 relativized()

```
virtual shared_value hocon::config_value::relativized (
    std::string prefix ) const [inline], [virtual], [inherited]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at foo.bar in the parent, and the included file as a substitution \${a.b.c}, the included substitution now needs to be \${foo.bar.a.b.c} because we resolve substitutions globally only after parsing everything.

##### Parameters

<i>prefix</i>	
---------------	--

##### Returns

value relativized to the given path or the same value if nothing to do

Reimplemented in [hocon::config\\_concatenation](#), and [hocon::simple\\_config\\_list](#).

Definition at line 181 of file config\_value.hpp.

#### 7.10.3.6 render() [1/2]

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to  
`render(config_render_options())`

.

#### Returns

the rendered value

### 7.10.3.7 render() [2/2]

```
virtual std::string hocon::config_value::render (
    config_render_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

#### Parameters

<i>options</i>	the rendering options
----------------	-----------------------

#### Returns

the rendered value

### 7.10.3.8 replace\_child()

```
shared_value hocon::config_delayed_merge::replace_child (
    shared_value const & child,
    shared_value replacement ) const [override], [virtual]
```

Replace a child of this value.

CAUTION if replacement is null, delete the child, which may also delete the parent, or make the parent into a non-container.

Implements [hocon::container](#).

### 7.10.3.9 to\_fallback\_value()

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],  
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

### 7.10.3.10 value\_type()

```
config_value::type hocon::config_delayed_merge::value_type ( ) const [override], [virtual]
```

The type of the value; matches the JSON type schema.

#### Returns

value's type

Implements [hocon::config\\_value](#).

### 7.10.3.11 value\_type\_name()

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

#### Returns

value's type's name

Definition at line 92 of file config\_value.hpp.

### 7.10.3.12 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

#### Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

#### Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

### 7.10.3.13 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```

Returns a.  
config\_value

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single [config\\_value](#).

#### Parameters

<i>origin</i>	the origin set on the returned value
---------------	--------------------------------------

#### Returns

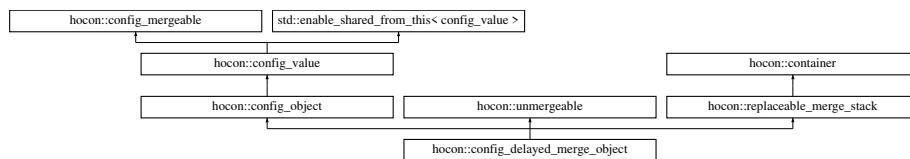
the new [config\\_value](#) with the given origin

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

- internal/values/config\_delayed\_merge.hpp

## 7.11 hocon::config\_delayed\_merge\_object Class Reference

Inheritance diagram for hocon::config\_delayed\_merge\_object:



### Public Types

- using **iterator** = std::unordered\_map< std::string, shared\_value >::const\_iterator
- enum class **type** {  
  **OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
  **CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

*The type of a configuration value (following the [JSON](#) type schema).*

### Public Member Functions

- **config\_delayed\_merge\_object** (shared\_origin [origin](#), std::vector< shared\_value > const &stack)
- [resolve\\_result](#)< shared\_value > **resolve\_substitutions** ([resolve\\_context](#) const &context, [resolve\\_source](#) const &source) const override
- std::vector< shared\_value > **unmerged\_values** () const override
- shared\_value **make\_replacement** ([resolve\\_context](#) const &context, int skipping) const override
- shared\_object **with\_value** ([path](#) raw\_path, shared\_value [value](#)) const override
- shared\_object **with\_value** (std::string key, shared\_value [value](#)) const override
- resolve\_status **get\_resolve\_status** () const override
- std::vector< std::string > **key\_set** () const override
- *Construct a list of keys in the \_value map.*
- bool **is\_empty** () const override

- `size_t size ()` const override
- `shared_value operator[] (std::string const &key)` const override
- `shared_value get (std::string const &key)` const override
- `iterator begin ()` const override
- `iterator end ()` const override
- `unwrapped_value unwrapped ()` const override
- `bool operator== (config_value const &other)` const override
- `shared_value replace_child (shared_value const &child, shared_value replacement)` const override  
*Replace a child of this value.*
- `bool has_descendant (shared_value const &descendant)` const override  
*Super-expensive full traversal to see if descendant is anywhere underneath this container.*
- `virtual std::shared_ptr< const config > to_config ()` const  
*Converts this object to a Config instance, enabling you to use path expressions to find values in the object.*
- `config_value::type value_type ()` const override  
*The type of the value; matches the JSON type schema.*
- `virtual shared_origin const & origin ()` const  
*The origin of the value (file, line number, etc.), for debugging and error messages.*
- `char const * value_type_name ()` const  
*The printable name of the value type.*
- `virtual std::string render ()` const  
*Renders the config value as a HOCON string.*
- `virtual std::string render (config_render_options options)` const  
*Renders the config value to a string, using the provided options.*
- `shared_config at_key (std::string const &key)` const  
*Places the value inside a config at the given key.*
- `shared_config at_path (std::string const &path_expression)` const  
*Places the value inside a config at the given path.*
- `virtual shared_value with_origin (shared_origin origin)` const  
*Returns a.*
- `virtual shared_value relativized (std::string prefix)` const  
*This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.*
- `std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other)` const override  
*Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*
- `virtual std::string transform_to_string ()` const

## Static Public Member Functions

- `static char const * type_name (type t)`

## Protected Member Functions

- `shared_value attempt_peek_with_partial_resolve (std::string const &key)` const override  
*Look up the key on an only-partially-resolved object, with no transformation or type conversion of any kind; if 'this' is not resolved then try to look up the key anyway if possible.*
- `std::unordered_map< std::string, shared_value > const & entry_set ()` const override
- `shared_object without_path (path raw_path)` const override
- `shared_object with_only_path (path raw_path)` const override
- `shared_object with_only_path_or_null (path raw_path)` const override

- shared\_object **new\_copy** (resolve\_status const &status, shared\_origin [origin](#)) const override
- bool **ignores\_fallbacks** () const override
- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &at\_key, [config\\_render\\_options](#) options) const override
- virtual void **render** (std::string &result, int indent, bool at\_root, [config\\_render\\_options](#) options) const override
- shared\_value **peek\_path** ([path](#) desired\_path) const
- shared\_value **peek\_assuming\_resolved** (std::string const &key, [path](#) original\_path) const
- shared\_value **new\_copy** (shared\_origin [origin](#)) const override
- shared\_value **construct\_delayed\_merge** (shared\_origin [origin](#), std::vector< shared\_value > stack) const override
- shared\_config **at\_key** (shared\_origin [origin](#), std::string const &key) const
- shared\_config **at\_path** (shared\_origin [origin](#), [path](#) raw\_path) const
- void **require\_not\_ignoring\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const [unmergeable](#) > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const [unmergeable](#) > fallback) const
- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- shared\_value **to\_fallback\_value** () const override

*Converts a config to its root object and a [config\\_value](#) to itself.*

## Static Protected Member Functions

- static shared\_value **peek\_path** (const [config\\_object](#) \*self, [path](#) desired\_path)
- static shared\_origin **merge\_origins** (std::vector< shared\_value > const &stack)
- static void **indent** (std::string &result, int indent, [config\\_render\\_options](#) const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** ([config\\_value](#) const &other, std::function< bool(T const &)> checker)

### 7.11.1 Detailed Description

Definition at line 10 of file `config_delayed_merge_object.hpp`.

### 7.11.2 Member Enumeration Documentation

#### 7.11.2.1 type

```
enum hocon::config\_value::type [strong], [inherited]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file `config_value.hpp`.

### 7.11.3 Member Function Documentation

#### 7.11.3.1 at\_key()

```
shared_config hocon::config_value::at_key (
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also `config_value#at_path(string)`.

##### Parameters

<i>key</i>	key to store this value at.
------------	-----------------------------

##### Returns

a  
`config`

instance containing this value at the given key.

#### 7.11.3.2 at\_path()

```
shared_config hocon::config_value::at_path (
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

##### Parameters

<i>path</i>	path to store this value at.
-------------	------------------------------

##### Returns

a  
`config`

instance containing this value at the given path.

#### 7.11.3.3 attempt\_peek\_with\_partial\_resolve()

```
shared_value hocon::config_delayed_merge_object::attempt_peek_with_partial_resolve (
    std::string const & key ) const [override], [protected], [virtual]
```



Look up the key on an only-partially-resolved object, with no transformation or type conversion of any kind; if 'this' is not resolved then try to look up the key anyway if possible.

#### Parameters

<i>key</i>	key to look up
------------	----------------

#### Returns

the value of the key, or null if known not to exist

#### Exceptions

<a href="#"><i>config_exception</i></a>	if can't figure out key's value (or existence) without more resolving
---	---

Implements [hocon::config\\_object](#).

#### 7.11.3.4 has\_descendant()

```
bool hocon::config_delayed_merge_object::has_descendant (
    shared_value const & descendant ) const [override], [virtual]
```

Super-expensive full traversal to see if descendant is anywhere underneath this container.

Implements [hocon::container](#).

#### 7.11.3.5 key\_set()

```
std::vector<std::string> hocon::config_delayed_merge_object::key_set ( ) const [inline],
[override], [virtual]
```

Construct a list of keys in the `_value` map.

Use a vector rather than set, because most of the time we just want to iterate over them.

Implements [hocon::config\\_object](#).

Definition at line 23 of file `config_delayed_merge_object.hpp`.

### 7.11.3.6 origin()

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

#### Returns

where the value came from

### 7.11.3.7 relativized()

```
virtual shared_value hocon::config_value::relativized (
    std::string prefix ) const [inline], [virtual], [inherited]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at `foo.bar` in the parent, and the included file as a substitution `$(a.b.c)`, the included substitution now needs to be `$(foo.bar.a.b.c)` because we resolve substitutions globally only after parsing everything.

#### Parameters

<i>prefix</i>	
---------------	--

#### Returns

value relativized to the given path or the same value if nothing to do

Reimplemented in [hocon::config\\_concatenation](#), and [hocon::simple\\_config\\_list](#).

Definition at line 181 of file `config_value.hpp`.

### 7.11.3.8 render() [1/2]

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to  
[render](#)(`config_render_options()`)

.

#### Returns

the rendered value

### 7.11.3.9 render() [2/2]

```
virtual std::string hocon::config_value::render (
    config_render_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

#### Parameters

<i>options</i>	the rendering options
----------------	-----------------------

#### Returns

the rendered value

### 7.11.3.10 replace\_child()

```
shared_value hocon::config_delayed_merge_object::replace_child (
    shared_value const & child,
    shared_value replacement ) const [override], [virtual]
```

Replace a child of this value.

CAUTION if replacement is null, delete the child, which may also delete the parent, or make the parent into a non-container.

Implements [hocon::container](#).

### 7.11.3.11 to\_config()

```
virtual std::shared_ptr<const config> hocon::config_object::to_config ( ) const [virtual],
[inherited]
```

Converts this object to a Config instance, enabling you to use path expressions to find values in the object.

This is a constant-time operation (it is not proportional to the size of the object).

#### Returns

a Config with this object as its root

### 7.11.3.12 to\_fallback\_value()

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],  
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

### 7.11.3.13 value\_type()

```
config_value::type hocon::config_object::value_type ( ) const [override], [virtual], [inherited]
```

The type of the value; matches the JSON type schema.

#### Returns

value's type

Implements [hocon::config\\_value](#).

### 7.11.3.14 value\_type\_name()

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

#### Returns

value's type's name

Definition at line 92 of file config\_value.hpp.

### 7.11.3.15 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

#### Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

#### Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

### 7.11.3.16 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```

Returns a.  
`config_value`

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single `config_value`.

#### Parameters

<code>origin</code>	the origin set on the returned value
---------------------	--------------------------------------

#### Returns

the new `config_value` with the given origin

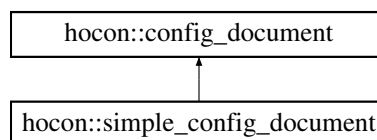
The documentation for this class was generated from the following file:

- `internal/values/config_delayed_merge_object.hpp`

## 7.12 hocon::config\_document Class Reference

Represents an individual HOCON or JSON file, preserving all formatting and syntax details.

Inheritance diagram for `hocon::config_document`:



### Public Member Functions

- `virtual std::unique_ptr< config_document > with_value_text (std::string path, std::string newValue) const =0`  
Returns a new `config_document` that is a copy of the current `config_document`, but with the desired value set at the desired path.
- `virtual std::unique_ptr< config_document > with_value (std::string path, std::shared_ptr< config_value > new_value) const =0`  
Returns a new `config_document` that is a copy of the current `config_document`, but with the desired value set at the desired path.
- `virtual std::unique_ptr< config_document > without_path (std::string path) const =0`  
Returns a new `config_document` that is a copy of the current `config_document`, but with all values at the desired path removed.
- `virtual bool has_path (std::string const &path) const =0`  
Returns a boolean indicating whether or not a `config_document` has a value at the desired path.
- `virtual std::string render () const =0`  
The original text of the input, modified if necessary with any replaced or added values.

### 7.12.1 Detailed Description

Represents an individual HOCON or JSON file, preserving all formatting and syntax details.

This can be used to replace individual values and exactly render the original text of the input.

Because this object is immutable, it is safe to use from multiple threads and there's no need for "defensive copies."

*Do not implement interface*

`config_document`

; it should only be implemented by the config library. Arbitrary implementations will not work because the library internals assume a specific concrete implementation. Also, this interface is likely to grow new methods over time, so third-party implementations will break.

Definition at line 26 of file `config_document.hpp`.

### 7.12.2 Member Function Documentation

#### 7.12.2.1 has\_path()

```
virtual bool hocon::config_document::has_path (
    std::string const & path ) const    [pure virtual]
```

Returns a boolean indicating whether or not a [config\\_document](#) has a value at the desired path.

null counts as a value for purposes of this check.

##### Parameters

<i>path</i>	the path to check
-------------	-------------------

##### Returns

true if the path exists in the document, otherwise false

Implemented in [hocon::simple\\_config\\_document](#).

#### 7.12.2.2 render()

```
virtual std::string hocon::config_document::render ( ) const    [pure virtual]
```

The original text of the input, modified if necessary with any replaced or added values.

##### Returns

the modified original text

Implemented in [hocon::simple\\_config\\_document](#).

### 7.12.2.3 with\_value()

```
virtual std::unique_ptr<config_document> hocon::config_document::with_value (
    std::string path,
    std::shared_ptr< config_value > new_value ) const [pure virtual]
```

Returns a new [config\\_document](#) that is a copy of the current [config\\_document](#), but with the desired value set at the desired path.

Works like `with_value_text(string, string)`, but takes a [config\\_value](#) instead of a string.

#### Parameters

<i>path</i>	the path at which to set the desired value
<i>new_value</i>	the value to set at the desired path, represented as a ConfigValue. The rendered text of the ConfigValue will be inserted into the <a href="#">config_document</a> .

#### Returns

a copy of the [config\\_document](#) with the desired value at the desired path

Implemented in [hocon::simple\\_config\\_document](#).

### 7.12.2.4 with\_value\_text()

```
virtual std::unique_ptr<config_document> hocon::config_document::with_value_text (
    std::string path,
    std::string newValue ) const [pure virtual]
```

Returns a new [config\\_document](#) that is a copy of the current [config\\_document](#), but with the desired value set at the desired path.

If the path exists, it will remove all duplicates before the final occurrence of the path, and replace the value at the final occurrence of the path. If the path does not exist, it will be added. If the document has an array as the root value, an exception will be thrown.

#### Parameters

<i>path</i>	the path at which to set the desired value
<i>new_value</i>	the value to set at the desired path, represented as a string. This string will be parsed into a <a href="#">config_node</a> using the same options used to parse the entire document, and the text will be inserted as-is into the document. Leading and trailing comments, whitespace, or newlines are not allowed, and if present an exception will be thrown. If a concatenation is passed in for <code>newValue</code> but the document was parsed with JSON, the first value in the concatenation will be parsed and inserted into the <a href="#">config_document</a> .

#### Returns

a copy of the [config\\_document](#) with the desired value at the desired path



Implemented in [hocon::simple\\_config\\_document](#).

### 7.12.2.5 without\_path()

```
virtual std::unique_ptr<config_document> hocon::config_document::without_path (
    std::string path ) const [pure virtual]
```

Returns a new [config\\_document](#) that is a copy of the current [config\\_document](#), but with all values at the desired path removed.

If the path does not exist in the document, a copy of the current document will be returned. If there is an array at the root, an exception will be thrown.

#### Parameters

<i>path</i>	the path to remove from the document
-------------	--------------------------------------

#### Returns

a copy of the [config\\_document](#) with the desired value removed from the document.

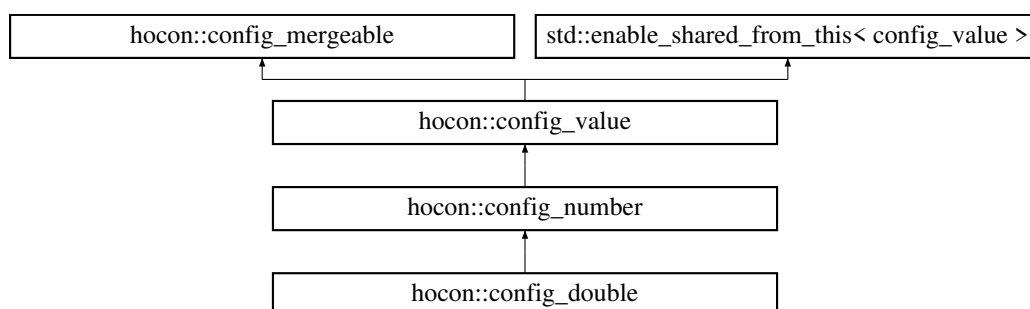
Implemented in [hocon::simple\\_config\\_document](#).

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

- [hocon/parser/config\\_document.hpp](#)

## 7.13 hocon::config\_double Class Reference

Inheritance diagram for hocon::config\_double:



### Public Types

- enum class [type](#) {  
**OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
**CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

*The type of a configuration value (following the [JSON](#) type schema).*

## Public Member Functions

- **config\_double** (shared\_origin [origin](#), double [value](#), std::string original\_text)
- std::string **transform\_to\_string** () const override
- unwrapped\_value **unwrapped** () const override
- int64\_t **long\_value** () const override
- double **double\_value** () const override
- [config\\_value::type](#) **value\_type** () const override
 

*The type of the value; matches the JSON type schema.*
- bool **is\_whole** () const
- bool **operator==** (const [config\\_number](#) &other) const
- bool **operator==** ([config\\_value](#) const &other) const override
- bool **operator!=** (const [config\\_number](#) &other) const
- int **int\_value\_range\_checked** (std::string const &path) const
- virtual shared\_origin const & **origin** () const
 

*The origin of the value (file, line number, etc.), for debugging and error messages.*
- char const \* **value\_type\_name** () const
 

*The printable name of the value type.*
- virtual std::string **render** () const
 

*Renders the config value as a HOCON string.*
- virtual std::string **render** ([config\\_render\\_options](#) options) const
 

*Renders the config value to a string, using the provided options.*
- shared\_config **at\_key** (std::string const &key) const
 

*Places the value inside a [config](#) at the given key.*
- shared\_config **at\_path** (std::string const &path\_expression) const
 

*Places the value inside a [config](#) at the given path.*
- virtual shared\_value **with\_origin** (shared\_origin [origin](#)) const
 

*Returns a.*
- virtual shared\_value **relativized** (std::string prefix) const
 

*This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.*
- virtual resolve\_status **get\_resolve\_status** () const
- std::shared\_ptr< const [config\\_mergeable](#) > **with\_fallback** (std::shared\_ptr< const [config\\_mergeable](#) > other) const override
 

*Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*

## Static Public Member Functions

- static std::shared\_ptr< [config\\_number](#) > **new\_number** (shared\_origin [origin](#), int64\_t [value](#), std::string original\_text)
- static std::shared\_ptr< [config\\_number](#) > **new\_number** (shared\_origin [origin](#), double [value](#), std::string original\_text)
- static char const \* **type\_name** ([type](#) t)

## Protected Member Functions

- shared\_value **new\_copy** (shared\_origin) const override
- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &at\_key, config\_render\_options options) const
- virtual void **render** (std::string &result, int indent, bool at\_root, config\_render\_options options) const
- shared\_config **at\_key** (shared\_origin origin, std::string const &key) const
- shared\_config **at\_path** (shared\_origin origin, path raw\_path) const
- virtual resolve\_result< shared\_value > **resolve\_substitutions** (resolve\_context const &context, resolve\_source const &source) const
- void **require\_not\_ignoring\_fallbacks** () const
- virtual bool **ignores\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const unmergeable > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const unmergeable > fallback) const
- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- virtual shared\_value **construct\_delayed\_merge** (shared\_origin origin, std::vector< shared\_value > stack) const
- shared\_value **to\_fallback\_value** () const override

*Converts a config to its root object and a config\_value to itself.*

## Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config\_render\_options const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** (config\_value const &other, std::function< bool(T const &)> checker)

## Protected Attributes

- std::string **\_original\_text**

### 7.13.1 Detailed Description

Definition at line 10 of file config\_double.hpp.

### 7.13.2 Member Enumeration Documentation

### 7.13.2.1 type

```
enum hocon::config_value::type [strong], [inherited]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file config\_value.hpp.

## 7.13.3 Member Function Documentation

### 7.13.3.1 at\_key()

```
shared_config hocon::config_value::at_key (  
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also `config_value#at_path(string)`.

#### Parameters

<i>key</i>	key to store this value at.
------------	-----------------------------

#### Returns

**a**  
`config`

instance containing this value at the given key.

### 7.13.3.2 at\_path()

```
shared_config hocon::config_value::at_path (  
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

#### Parameters

<i>path</i>	path to store this value at.
-------------	------------------------------

**Returns**

**a**  
config

instance containing this value at the given path.

**7.13.3.3 origin()**

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

**Returns**

where the value came from

**7.13.3.4 relativized()**

```
virtual shared_value hocon::config_value::relativized (
    std::string prefix ) const [inline], [virtual], [inherited]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at foo.bar in the parent, and the included file as a substitution \${a.b.c}, the included substitution now needs to be \${foo.bar.a.b.c} because we resolve substitutions globally only after parsing everything.

**Parameters**

<i>prefix</i>	
---------------	--

**Returns**

value relativized to the given path or the same value if nothing to do

Reimplemented in [hocon::config\\_concatenation](#), and [hocon::simple\\_config\\_list](#).

Definition at line 181 of file config\_value.hpp.

**7.13.3.5 render() [1/2]**

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to  
[render](#)([config\\_render\\_options](#)())

.

#### Returns

the rendered value

#### 7.13.3.6 [render\(\)](#) [2/2]

```
virtual std::string hocon::config_value::render (
    config\_render\_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

#### Parameters

<i>options</i>	the rendering options
----------------	-----------------------

#### Returns

the rendered value

#### 7.13.3.7 [to\\_fallback\\_value\(\)](#)

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

### 7.13.3.8 value\_type()

```
config_value::type hocon::config_number::value_type ( ) const [override], [virtual], [inherited]
```

The type of the value; matches the JSON type schema.

#### Returns

value's type

Implements [hocon::config\\_value](#).

### 7.13.3.9 value\_type\_name()

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

#### Returns

value's type's name

Definition at line 92 of file config\_value.hpp.

### 7.13.3.10 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

## Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

## Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

**7.13.3.11 with\_origin()**

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```

Returns a.  
`config_value`

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single [config\\_value](#).

## Parameters

<i>origin</i>	the origin set on the returned value
---------------	--------------------------------------

## Returns

the new [config\\_value](#) with the given origin

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

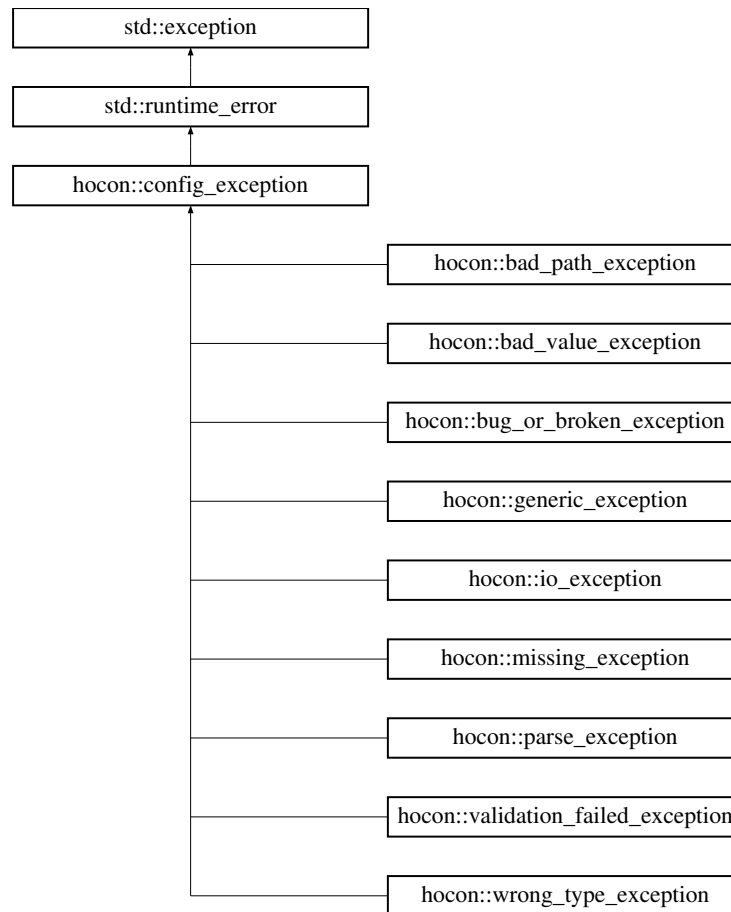
- `internal/values/config_double.hpp`

**7.14 hocon::config\_exception Struct Reference**

All exceptions thrown by the library are subclasses of [config\\_exception](#).

Inheritance diagram for `hocon::config_exception`:





### Public Member Functions

- **config\_exception** ([config\\_origin](#) const &origin, std::string const &message)
- **config\_exception** (std::string const &message)
- **config\_exception** (std::string const &message, std::exception const &e)

#### 7.14.1 Detailed Description

All exceptions thrown by the library are subclasses of [config\\_exception](#).

Definition at line 14 of file `config_exception.hpp`.

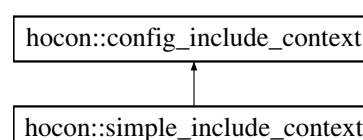
The documentation for this struct was generated from the following file:

- `hocon/config_exception.hpp`

## 7.15 hocon::config\_include\_context Class Reference

Context provided to a [config\\_includer](#); this interface is only useful inside a.

Inheritance diagram for `hocon::config_include_context`:



## Public Member Functions

- virtual shared\_parseable [relative\\_to](#) (std::string file\_name) const =0  
*Tries to find a name relative to whatever is doing the including, for example in the same directory as the file doing the including.*
- virtual [config\\_parse\\_options](#) [parse\\_options](#) () const =0  
*Parse options to use (if you use another method to get a [config\\_parseable](#) then use [config\\_parseable#options\(\)](#) instead though).*
- void **set\_cur\_dir** (std::string dir) const
- std::string **get\_cur\_dir** () const

## Protected Attributes

- std::shared\_ptr< std::string > **\_cur\_dir**

### 7.15.1 Detailed Description

Context provided to a [config\\_includer](#); this interface is only useful inside a.

implementation, and is not intended for apps to implement.

*Do not implement this interface*; it should only be implemented by the config library. Arbitrary implementations will not work because the library internals assume a specific concrete implementation. Also, this interface is likely to grow new methods over time, so third-party implementations will break.

Definition at line 21 of file `config_include_context.hpp`.

### 7.15.2 Member Function Documentation

#### 7.15.2.1 [parse\\_options\(\)](#)

```
virtual config\_parse\_options hocon::config_include_context::parse_options ( ) const [pure virtual]
```

Parse options to use (if you use another method to get a [config\\_parseable](#) then use [config\\_parseable#options\(\)](#) instead though).

#### Returns

the parse options

Implemented in [hocon::simple\\_include\\_context](#).

#### 7.15.2.2 [relative\\_to\(\)](#)

```
virtual shared_parseable hocon::config_include_context::relative_to (
    std::string file_name ) const [pure virtual]
```

Tries to find a name relative to whatever is doing the including, for example in the same directory as the file doing the including.

Returns null if it can't meaningfully create a relative name. The returned parseable may not exist; this function is not required to do any IO, just compute what the name would be.

The passed-in filename has to be a complete name (with extension), not just a basename. (Include statements in config files are allowed to give just a basename.)

## Parameters

<i>filename</i>	the name to make relative to the resource doing the including
-----------------	---

## Returns

parseable item relative to the resource doing the including, or null

Implemented in [hocon::simple\\_include\\_context](#).

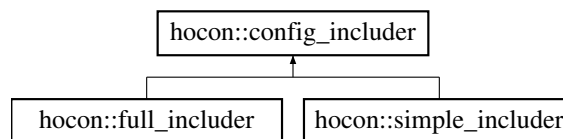
The documentation for this class was generated from the following file:

- [hocon/config\\_include\\_context.hpp](#)

## 7.16 hocon::config\_includer Class Reference

Implement this interface and provide an instance to [config\\_parse\\_options.set\\_includer\(\)](#) to customize handling of.

Inheritance diagram for hocon::config\_includer:



### Public Member Functions

- virtual shared\_includer [with\\_fallback](#) (shared\_includer fallback) const =0  
*Returns a new includer that falls back to the given includer.*
- virtual shared\_object [include](#) (shared\_include\_context context, std::string what) const =0  
*Parses another item to be included.*

#### 7.16.1 Detailed Description

Implement this interface and provide an instance to [config\\_parse\\_options.set\\_includer\(\)](#) to customize handling of. [include](#)

statements in config files. You may also want to implement [config\\_includer\\_file](#) and [config\\_includer\\_URL](#), or not.

Definition at line 15 of file [config\\_includer.hpp](#).

#### 7.16.2 Member Function Documentation

### 7.16.2.1 include()

```
virtual shared_object hocon::config_includer::include (
    shared_include_context context,
    std::string what ) const [pure virtual]
```

Parses another item to be included.

The returned object typically would not have substitutions resolved. You can throw a [config\\_exception](#) here to abort parsing, or return an empty object, but may not return null.

This method is used for a "heuristic" include statement that does not specify file, or URL resource. If the include statement does specify, then the same class implementing [config\\_includer](#) must also implement [config\\_includer\\_file](#) or [config\\_includer\\_URL](#) as needed, or a default includer will be used.

#### Parameters

<i>context</i>	some info about the include context
<i>what</i>	the include statement's argument

#### Returns

a non-null [config\\_object](#)

Implemented in [hocon::simple\\_includer](#).

### 7.16.2.2 with\_fallback()

```
virtual shared_includer hocon::config_includer::with_fallback (
    shared_includer fallback ) const [pure virtual]
```

Returns a new includer that falls back to the given includer.

This is how you can obtain the default includer; it will be provided as a fallback. It's up to your includer to chain to it if you want to. You might want to merge any files found by the fallback includer with any objects you load yourself.

It's important to handle the case where you already have the fallback with a "return this", i.e. this method should not create a new object if the fallback is the same one you already have. The same fallback may be added repeatedly.

#### Parameters

<i>fallback</i>	the previous includer for chaining
-----------------	------------------------------------

#### Returns

a new includer

Implemented in [hocon::simple\\_includer](#).

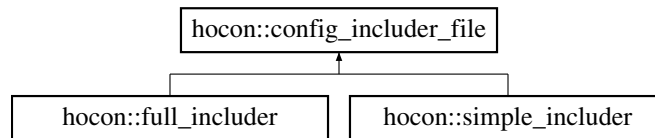
The documentation for this class was generated from the following file:

- [hocon/config\\_includer.hpp](#)

## 7.17 hocon::config\_includer\_file Class Reference

Implement this *in addition to* [config\\_includer](#) if you want to support inclusion of files with the.

Inheritance diagram for hocon::config\_includer\_file:



### Public Member Functions

- virtual shared\_object [include\\_file](#) (shared\_include\_context context, std::string what) const =0  
*Parses another item to be included.*

#### 7.17.1 Detailed Description

Implement this *in addition to* [config\\_includer](#) if you want to support inclusion of files with the.

```
include_file("filename")
```

syntax. If you do not implement this but do implement [config\\_includer](#), attempts to load files will use the default includer.

Definition at line 14 of file config\_includer\_file.hpp.

#### 7.17.2 Member Function Documentation

##### 7.17.2.1 include\_file()

```
virtual shared_object hocon::config_includer_file::include_file (
    shared_include_context context,
    std::string what ) const [pure virtual]
```

Parses another item to be included.

The returned object typically would not have substitutions resolved. You can throw a [config\\_exception](#) here to abort parsing, or return an empty object, but may not return null.

##### Parameters

<i>context</i>	some info about the include context
<i>what</i>	the include statement's argument (a file path)

**Returns**

a non-null [config\\_object](#)

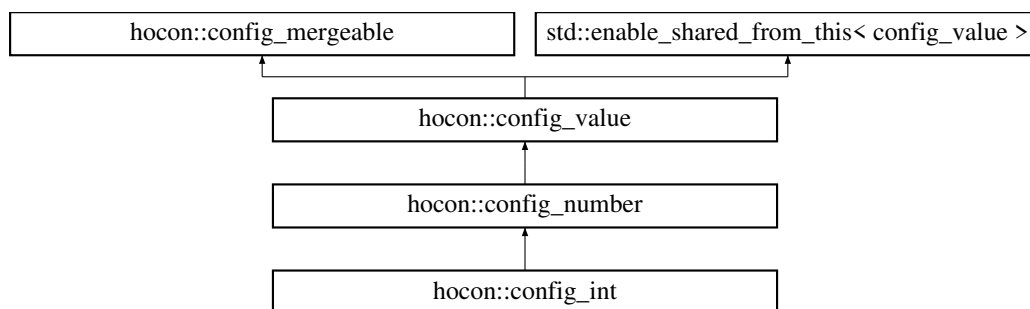
Implemented in [hocon::simple\\_includer](#).

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

- [hocon/config\\_includer\\_file.hpp](#)

## 7.18 hocon::config\_int Class Reference

Inheritance diagram for `hocon::config_int`:



### Public Types

- enum class [type](#) {  
**OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
**CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

*The type of a configuration value (following the [JSON](#) type schema).*

### Public Member Functions

- **config\_int** (shared\_origin [origin](#), int [value](#), std::string original\_text)
- std::string **transform\_to\_string** () const override
- unwrapped\_value **unwrapped** () const override
- int64\_t **long\_value** () const override
- double **double\_value** () const override
- [config\\_value::type](#) **value\_type** () const override

*The type of the value; matches the JSON type schema.*

- bool **is\_whole** () const
- bool **operator==** (const [config\\_number](#) &other) const
- bool **operator==** ([config\\_value](#) const &other) const override
- bool **operator!=** (const [config\\_number](#) &other) const
- int **int\_value\_range\_checked** (std::string const &[path](#)) const
- virtual shared\_origin const & **origin** () const

*The origin of the value (file, line number, etc.), for debugging and error messages.*

- char const \* **value\_type\_name** () const

*The printable name of the value type.*

- virtual std::string **render** () const  
*Renders the config value as a HOCON string.*
- virtual std::string **render** (config\_render\_options options) const  
*Renders the config value to a string, using the provided options.*
- shared\_config **at\_key** (std::string const &key) const  
*Places the value inside a **config** at the given key.*
- shared\_config **at\_path** (std::string const &path\_expression) const  
*Places the value inside a **config** at the given path.*
- virtual shared\_value **with\_origin** (shared\_origin origin) const  
*Returns a.*
- virtual shared\_value **relativized** (std::string prefix) const  
*This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.*
- virtual resolve\_status **get\_resolve\_status** () const
- std::shared\_ptr< const **config\_mergeable** > **with\_fallback** (std::shared\_ptr< const **config\_mergeable** > other) const override  
*Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*

## Static Public Member Functions

- static std::shared\_ptr< **config\_number** > **new\_number** (shared\_origin origin, int64\_t value, std::string original\_text)
- static std::shared\_ptr< **config\_number** > **new\_number** (shared\_origin origin, double value, std::string original\_text)
- static char const \* **type\_name** (type t)

## Protected Member Functions

- shared\_value **new\_copy** (shared\_origin) const override
- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &at\_key, config\_render\_options options) const
- virtual void **render** (std::string &result, int indent, bool at\_root, config\_render\_options options) const
- shared\_config **at\_key** (shared\_origin origin, std::string const &key) const
- shared\_config **at\_path** (shared\_origin origin, path raw\_path) const
- virtual **resolve\_result**< shared\_value > **resolve\_substitutions** (**resolve\_context** const &context, **resolve\_source** const &source) const
- void **require\_not\_ignoring\_fallbacks** () const
- virtual bool **ignores\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const **unmergeable** > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const **unmergeable** > fallback) const
- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- virtual shared\_value **construct\_delayed\_merge** (shared\_origin origin, std::vector< shared\_value > stack) const
- shared\_value **to\_fallback\_value** () const override

*Converts a config to its root object and a **config\_value** to itself.*

## Static Protected Member Functions

- static void **indent** (std::string &result, int indent, [config\\_render\\_options](#) const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** ([config\\_value](#) const &other, std::function< bool(T const &)> checker)

## Protected Attributes

- std::string **\_original\_text**

### 7.18.1 Detailed Description

Definition at line 7 of file config\_int.hpp.

### 7.18.2 Member Enumeration Documentation

#### 7.18.2.1 type

```
enum hocon::config\_value::type [strong], [inherited]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file config\_value.hpp.

### 7.18.3 Member Function Documentation

#### 7.18.3.1 at\_key()

```
shared_config hocon::config_value::at_key (
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also [config\\_value#at\\_path\(string\)](#).

#### Parameters

<i>key</i>	key to store this value at.
------------	-----------------------------



**Returns**

a  
config

instance containing this value at the given key.

**7.18.3.2 at\_path()**

```
shared_config hocon::config_value::at_path (
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

**Parameters**

<i>path</i>	path to store this value at.
-------------	------------------------------

**Returns**

a  
config

instance containing this value at the given path.

**7.18.3.3 origin()**

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

**Returns**

where the value came from

**7.18.3.4 relativized()**

```
virtual shared_value hocon::config_value::relativized (
    std::string prefix ) const [inline], [virtual], [inherited]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at `foo.bar` in the parent, and the included file as a substitution `$(a.b.c)`, the included substitution now needs to be `$(foo.bar.a.b.c)` because we resolve substitutions globally only after parsing everything.

## Parameters

<i>prefix</i>	
---------------	--

## Returns

value relativized to the given path or the same value if nothing to do

Reimplemented in [hocon::config\\_concatenation](#), and [hocon::simple\\_config\\_list](#).

Definition at line 181 of file `config_value.hpp`.

**7.18.3.5 render() [1/2]**

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to

```
render(config\_render\_options() )
```

.

## Returns

the rendered value

**7.18.3.6 render() [2/2]**

```
virtual std::string hocon::config_value::render (
    config\_render\_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

## Parameters

<i>options</i>	the rendering options
----------------	-----------------------

## Returns

the rendered value

**7.18.3.7 to\_fallback\_value()**

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],  
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

**7.18.3.8 value\_type()**

```
config_value::type hocon::config_number::value_type ( ) const [override], [virtual], [inherited]
```

The type of the value; matches the JSON type schema.

## Returns

value's type

Implements [hocon::config\\_value](#).

**7.18.3.9 value\_type\_name()**

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

## Returns

value's type's name

Definition at line 92 of file config\_value.hpp.

### 7.18.3.10 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

#### Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

#### Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

### 7.18.3.11 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```

Returns a.  
config\_value

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single config\_value.

#### Parameters

<i>origin</i>	the origin set on the returned value
---------------	--------------------------------------

#### Returns

the new config\_value with the given origin

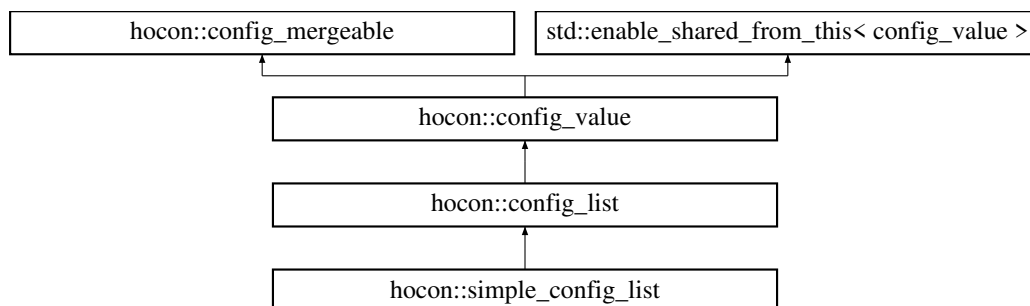
The documentation for this class was generated from the following file:

- internal/values/config\_int.hpp

## 7.19 hocon::config\_list Class Reference

Subtype of ConfigValue representing a list value, as in JSON's.

Inheritance diagram for hocon::config\_list:



### Public Types

- using **iterator** = std::vector< shared\_value >::const\_iterator
- enum class **type** {  
  **OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
  **CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

*The type of a configuration value (following the **JSON** type schema).*

## Public Member Functions

- **config\_list** (shared\_origin [origin](#))
- virtual bool **is\_empty** () const =0
- virtual size\_t **size** () const =0
- virtual shared\_value **operator[]** (size\_t index) const =0
- virtual shared\_value **get** (size\_t index) const =0
- virtual iterator **begin** () const =0
- virtual iterator **end** () const =0
- virtual unwrapped\_value **unwrapped** () const =0
- virtual shared\_origin const & [origin](#) () const  
*The origin of the value (file, line number, etc.), for debugging and error messages.*
- virtual [type](#) **value\_type** () const =0  
*The type of the value; matches the JSON type schema.*
- char const \* [value\\_type\\_name](#) () const  
*The printable name of the value type.*
- virtual std::string [render](#) () const  
*Renders the config value as a HOCON string.*
- virtual std::string [render](#) ([config\\_render\\_options](#) options) const  
*Renders the config value to a string, using the provided options.*
- shared\_config [at\\_key](#) (std::string const &key) const  
*Places the value inside a [config](#) at the given key.*
- shared\_config [at\\_path](#) (std::string const &path\_expression) const  
*Places the value inside a [config](#) at the given path.*
- virtual shared\_value [with\\_origin](#) (shared\_origin [origin](#)) const  
*Returns a.*
- virtual shared\_value [relativized](#) (std::string prefix) const  
*This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.*
- virtual resolve\_status **get\_resolve\_status** () const
- std::shared\_ptr< const [config\\_mergeable](#) > [with\\_fallback](#) (std::shared\_ptr< const [config\\_mergeable](#) > other) const override  
*Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*
- virtual bool **operator==** ([config\\_value](#) const &other) const =0
- virtual std::string **transform\_to\_string** () const

## Static Public Member Functions

- static char const \* [type\\_name](#) ([type](#) t)

## Protected Member Functions

- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &[at\\_key](#), [config\\_render\\_options](#) options) const
- virtual void **render** (std::string &result, int indent, bool at\_root, [config\\_render\\_options](#) options) const
- shared\_config **at\_key** (shared\_origin [origin](#), std::string const &key) const
- shared\_config **at\_path** (shared\_origin [origin](#), [path](#) raw\_path) const
- virtual shared\_value **new\_copy** (shared\_origin [origin](#)) const =0
- virtual [resolve\\_result](#)< shared\_value > **resolve\_substitutions** ([resolve\\_context](#) const &context, [resolve\\_source](#) const &source) const
- void **require\_not\_ignoring\_fallbacks** () const

- virtual bool **ignores\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const [unmergeable](#) > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const [unmergeable](#) > fallback) const
- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- virtual shared\_value **construct\_delayed\_merge** (shared\_origin [origin](#), std::vector< shared\_value > stack) const
- shared\_value **to\_fallback\_value** () const override

*Converts a config to its root object and a [config\\_value](#) to itself.*

## Static Protected Member Functions

- static void **indent** (std::string &result, int indent, [config\\_render\\_options](#) const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** ([config\\_value](#) const &other, std::function< bool(T const &)> checker)

### 7.19.1 Detailed Description

Subtype of ConfigValue representing a list value, as in JSON's.

[1, 2, 3]

syntax.

`ConfigList`

implements

`java.util.List<ConfigValue>`

so you can use it like a regular Java list. Or call `unwrapped()` to unwrap the list elements into plain Java values.

Like all ConfigValue subtypes,

`ConfigList`

is immutable. This makes it threadsafe and you never have to create "defensive copies." The mutator methods from `java.util.List` all throw `java.lang.UnsupportedOperationException`.

The `ConfigValue#valueType` method on a list returns `ConfigValueType#LIST`.

*Do not implement*

`ConfigList`

; it should only be implemented by the config library. Arbitrary implementations will not work because the library internals assume a specific concrete implementation. Also, this interface is likely to grow new methods over time, so third-party implementations will break.

Definition at line 37 of file `config_list.hpp`.

## 7.19.2 Member Enumeration Documentation

### 7.19.2.1 type

```
enum hocon::config_value::type [strong], [inherited]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file `config_value.hpp`.

## 7.19.3 Member Function Documentation

### 7.19.3.1 at\_key()

```
shared_config hocon::config_value::at_key (  
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also `config_value#at_path(string)`.

#### Parameters

<i>key</i>	key to store this value at.
------------	-----------------------------

#### Returns

**a**  
`config`

instance containing this value at the given key.

### 7.19.3.2 at\_path()

```
shared_config hocon::config_value::at_path (  
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.



#### Parameters

<i>path</i>	path to store this value at.
-------------	------------------------------

#### Returns

**a**  
config  
instance containing this value at the given path.

#### 7.19.3.3 origin()

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

#### Returns

where the value came from

#### 7.19.3.4 relativized()

```
virtual shared_value hocon::config_value::relativized (
    std::string prefix ) const [inline], [virtual], [inherited]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at foo.bar in the parent, and the included file as a substitution \${a.b.c}, the included substitution now needs to be \${foo.bar.a.b.c} because we resolve substitutions globally only after parsing everything.

#### Parameters

<i>prefix</i>	
---------------	--

#### Returns

value relativized to the given path or the same value if nothing to do

Reimplemented in [hocon::config\\_concatenation](#), and [hocon::simple\\_config\\_list](#).

Definition at line 181 of file config\_value.hpp.

**7.19.3.5 render()** [1/2]

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to  
[render](#)([config\\_render\\_options](#)() )

.

**Returns**

the rendered value

**7.19.3.6 render()** [2/2]

```
virtual std::string hocon::config_value::render (
    config\_render\_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

**Parameters**

<i>options</i>	the rendering options
----------------	-----------------------

**Returns**

the rendered value

**7.19.3.7 to\_fallback\_value()**

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

#### 7.19.3.8 value\_type()

```
virtual type hocon::config_value::value_type ( ) const [pure virtual], [inherited]
```

The type of the value; matches the JSON type schema.

##### Returns

value's type

Implemented in [hocon::simple\\_config\\_list](#), [hocon::config\\_string](#), [hocon::config\\_reference](#), [hocon::config\\_number](#), [hocon::config\\_null](#), [hocon::config\\_delayed\\_merge](#), [hocon::config\\_concatenation](#), [hocon::config\\_boolean](#), and [hocon::config\\_object](#).

#### 7.19.3.9 value\_type\_name()

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

##### Returns

value's type's name

Definition at line 92 of file [config\\_value.hpp](#).

### 7.19.3.10 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

#### Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

#### Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

### 7.19.3.11 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```

Returns a.  
config\_value

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single config\_value.

#### Parameters

<i>origin</i>	the origin set on the returned value
---------------	--------------------------------------

#### Returns

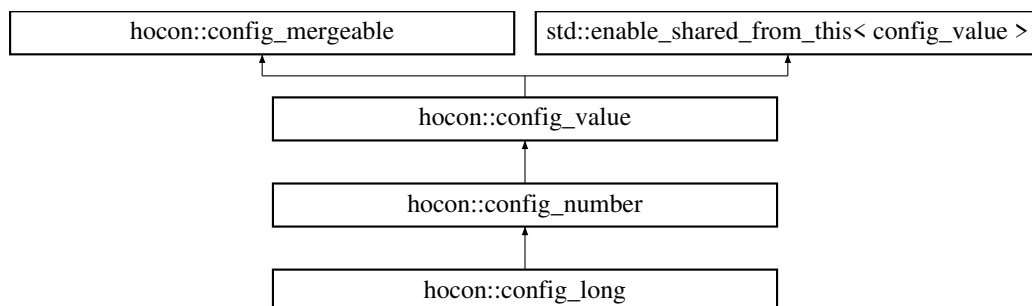
the new config\_value with the given origin

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

- hocon/config\_list.hpp

## 7.20 hocon::config\_long Class Reference

Inheritance diagram for hocon::config\_long:



### Public Types

- enum class type {  
  **OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
  **CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

*The type of a configuration value (following the JSON type schema).*

### Public Member Functions

- **config\_long** (shared\_origin origin, int64\_t value, std::string original\_text)
- std::string **transform\_to\_string** () const override
- unwrapped\_value **unwrapped** () const override
- int64\_t **long\_value** () const override
- double **double\_value** () const override
- config\_value::type value\_type () const override

*The type of the value; matches the JSON type schema.*

- bool **is\_whole** () const
- bool **operator==** (const [config\\_number](#) &other) const
- bool **operator==** ([config\\_value](#) const &other) const override
- bool **operator!=** (const [config\\_number](#) &other) const
- int **int\_value\_range\_checked** (std::string const &path) const
- virtual shared\_origin const & **origin** () const  
*The origin of the value (file, line number, etc.), for debugging and error messages.*
- char const \* **value\_type\_name** () const  
*The printable name of the value type.*
- virtual std::string **render** () const  
*Renders the config value as a HOCON string.*
- virtual std::string **render** ([config\\_render\\_options](#) options) const  
*Renders the config value to a string, using the provided options.*
- shared\_config **at\_key** (std::string const &key) const  
*Places the value inside a [config](#) at the given key.*
- shared\_config **at\_path** (std::string const &path\_expression) const  
*Places the value inside a [config](#) at the given path.*
- virtual shared\_value **with\_origin** (shared\_origin origin) const  
*Returns a.*
- virtual shared\_value **relativized** (std::string prefix) const  
*This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.*
- virtual resolve\_status **get\_resolve\_status** () const
- std::shared\_ptr< const [config\\_mergeable](#) > **with\_fallback** (std::shared\_ptr< const [config\\_mergeable](#) > other) const override  
*Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*

## Static Public Member Functions

- static std::shared\_ptr< [config\\_number](#) > **new\_number** (shared\_origin origin, int64\_t value, std::string original\_text)
- static std::shared\_ptr< [config\\_number](#) > **new\_number** (shared\_origin origin, double value, std::string original\_text)
- static char const \* **type\_name** (type t)

## Protected Member Functions

- shared\_value **new\_copy** (shared\_origin) const override
- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &at\_key, [config\\_render\\_options](#) options) const
- virtual void **render** (std::string &result, int indent, bool at\_root, [config\\_render\\_options](#) options) const
- shared\_config **at\_key** (shared\_origin origin, std::string const &key) const
- shared\_config **at\_path** (shared\_origin origin, [path](#) raw\_path) const
- virtual [resolve\\_result](#)< shared\_value > **resolve\_substitutions** ([resolve\\_context](#) const &context, [resolve\\_source](#) const &source) const
- void **require\_not\_ignoring\_fallbacks** () const
- virtual bool **ignores\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const [unmergeable](#) > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const [unmergeable](#) > fallback) const

- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- virtual shared\_value **construct\_delayed\_merge** (shared\_origin [origin](#), std::vector< shared\_value > stack) const
- shared\_value [to\\_fallback\\_value](#) () const override

*Converts a config to its root object and a [config\\_value](#) to itself.*

## Static Protected Member Functions

- static void **indent** (std::string &result, int indent, [config\\_render\\_options](#) const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** ([config\\_value](#) const &other, std::function< bool(T const &)> checker)

## Protected Attributes

- std::string **\_original\_text**

### 7.20.1 Detailed Description

Definition at line 10 of file config\_long.hpp.

### 7.20.2 Member Enumeration Documentation

#### 7.20.2.1 type

```
enum hocon::config\_value::type [strong], [inherited]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file config\_value.hpp.

### 7.20.3 Member Function Documentation

#### 7.20.3.1 at\_key()

```
shared_config hocon::config_value::at_key (
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also [config\\_value#at\\_path\(string\)](#).

**Parameters**

<i>key</i>	key to store this value at.
------------	-----------------------------

**Returns**

**a**  
config

instance containing this value at the given key.

**7.20.3.2 at\_path()**

```
shared_config hocon::config_value::at_path (  
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

**Parameters**

<i>path</i>	path to store this value at.
-------------	------------------------------

**Returns**

**a**  
config

instance containing this value at the given path.

**7.20.3.3 origin()**

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

**Returns**

where the value came from



#### 7.20.3.4 relativized()

```
virtual shared_value hocon::config_value::relativized (
    std::string prefix ) const    [inline], [virtual], [inherited]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at `foo.bar` in the parent, and the included file as a substitution `$(a.b.c)`, the included substitution now needs to be `$(foo.bar.a.b.c)` because we resolve substitutions globally only after parsing everything.

## Parameters

<i>prefix</i>	
---------------	--

## Returns

value relativized to the given path or the same value if nothing to do

Reimplemented in [hocon::config\\_concatenation](#), and [hocon::simple\\_config\\_list](#).

Definition at line 181 of file `config_value.hpp`.

**7.20.3.5 render() [1/2]**

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to

```
render(config_render_options())
```

.

## Returns

the rendered value

**7.20.3.6 render() [2/2]**

```
virtual std::string hocon::config_value::render (
    config\_render\_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

**Parameters**

<i>options</i>	the rendering options
----------------	-----------------------

**Returns**

the rendered value

**7.20.3.7 to\_fallback\_value()**

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],  
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

**7.20.3.8 value\_type()**

```
config_value::type hocon::config_number::value_type ( ) const [override], [virtual], [inherited]
```

The type of the value; matches the JSON type schema.

**Returns**

value's type

Implements [hocon::config\\_value](#).

**7.20.3.9 value\_type\_name()**

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

**Returns**

value's type's name

Definition at line 92 of file config\_value.hpp.

### 7.20.3.10 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

#### Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

#### Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

### 7.20.3.11 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```

Returns a.  
config\_value

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single config\_value.

#### Parameters

<i>origin</i>	the origin set on the returned value
---------------	--------------------------------------

#### Returns

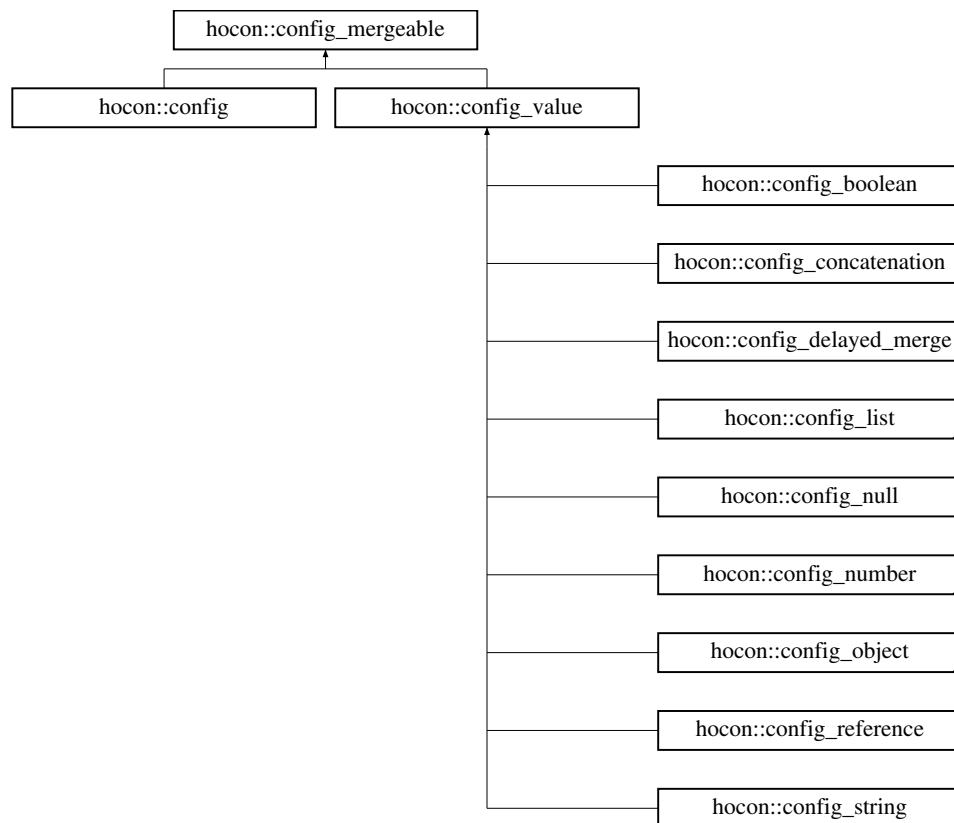
the new config\_value with the given origin

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

- internal/values/config\_long.hpp

## 7.21 hocon::config\_mergeable Class Reference

Inheritance diagram for hocon::config\_mergeable:



### Public Member Functions

- virtual std::shared\_ptr< const config\_mergeable > with\_fallback (std::shared\_ptr< const config\_mergeable > other) const =0

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

## Protected Member Functions

- virtual `shared_value to_fallback_value () const =0`  
*Converts a config to its root object and a `config_value` to itself.*

## Friends

- class `config_value`

### 7.21.1 Detailed Description

Definition at line 8 of file `config_mergeable.hpp`.

### 7.21.2 Member Function Documentation

#### 7.21.2.1 to\_fallback\_value()

```
virtual shared_value hocon::config_mergeable::to_fallback_value ( ) const [protected], [pure virtual]
```

Converts a config to its root object and a `config_value` to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implemented in `hocon::config_value`, and `hocon::config`.

#### 7.21.2.2 with\_fallback()

```
virtual std::shared_ptr<const config_mergeable> hocon::config_mergeable::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [pure virtual]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

**Parameters**

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

**Returns**

a new object (or the original one, if the fallback doesn't get used)

Implemented in [hocon::config\\_value](#), and [hocon::config](#).

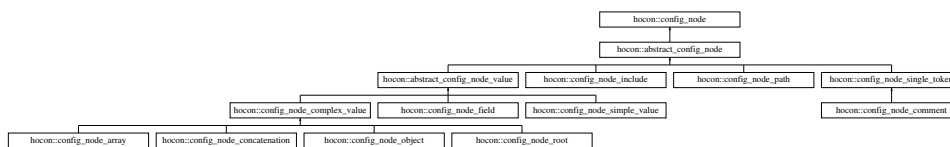
The documentation for this class was generated from the following file:

- [hocon/config\\_mergeable.hpp](#)

## 7.22 hocon::config\_node Class Reference

A node in the syntax tree for a HOCON or JSON document.

Inheritance diagram for `hocon::config_node`:

**Public Member Functions**

- virtual `std::string render () const =0`

*The original text of the input which was used to form this particular node.*

### 7.22.1 Detailed Description

A node in the syntax tree for a HOCON or JSON document.

Because this object is immutable, it is safe to use from multiple threads and there's no need for "defensive copies."

*Do not implement interface*

`ConfigNode`

; it should only be implemented by the config library. Arbitrary implementations will not work because the library internals assume a specific concrete implementation. Also, this interface is likely to grow new methods over time, so third-party implementations will break.

Definition at line 21 of file `config_node.hpp`.

### 7.22.2 Member Function Documentation

### 7.22.2.1 render()

```
virtual std::string hocon::config_node::render ( ) const [pure virtual]
```

The original text of the input which was used to form this particular node.

#### Returns

the original text used to form this node as a String

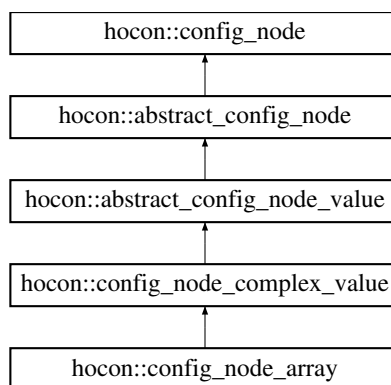
Implemented in [hocon::abstract\\_config\\_node](#).

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

- [hocon/parser/config\\_node.hpp](#)

## 7.23 hocon::config\_node\_array Class Reference

Inheritance diagram for hocon::config\_node\_array:



### Public Member Functions

- **config\_node\_array** (shared\_node\_list children)
- std::shared\_ptr< const [config\\_node\\_complex\\_value](#) > **new\_node** (shared\_node\_list nodes) const override
- token\_list **get\_tokens** () const override
- shared\_node\_list const & **children** () const
- std::shared\_ptr< const [config\\_node\\_complex\\_value](#) > **indent\_text** (shared\_node indentation) const
- std::string **render** () const
- *The original text of the input which was used to form this particular node.*
- bool **operator==** (const [abstract\\_config\\_node](#) &other) const

### 7.23.1 Detailed Description

Definition at line 10 of file config\_node\_array.hpp.



## 7.23.2 Member Function Documentation

### 7.23.2.1 render()

```
std::string hocon::abstract_config_node::render ( ) const [virtual], [inherited]
```

The original text of the input which was used to form this particular node.

#### Returns

the original text used to form this node as a String

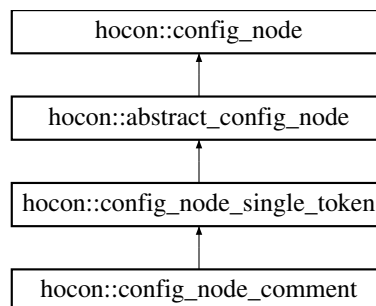
Implements [hocon::config\\_node](#).

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

- internal/nodes/config\_node\_array.hpp

## 7.24 hocon::config\_node\_comment Class Reference

Inheritance diagram for hocon::config\_node\_comment:



### Public Member Functions

- **config\_node\_comment** (shared\_token [comment](#))
- std::string **comment\_text** ( ) const
- token\_list **get\_tokens** ( ) const override
- shared\_token **get\_token** ( ) const
- std::string **render** ( ) const  
*The original text of the input which was used to form this particular node.*
- bool **operator==** (const [abstract\\_config\\_node](#) &other) const

### 7.24.1 Detailed Description

Definition at line 7 of file config\_node\_comment.hpp.

## 7.24.2 Member Function Documentation

### 7.24.2.1 render()

```
std::string hocon::abstract_config_node::render ( ) const [virtual], [inherited]
```

The original text of the input which was used to form this particular node.

#### Returns

the original text used to form this node as a String

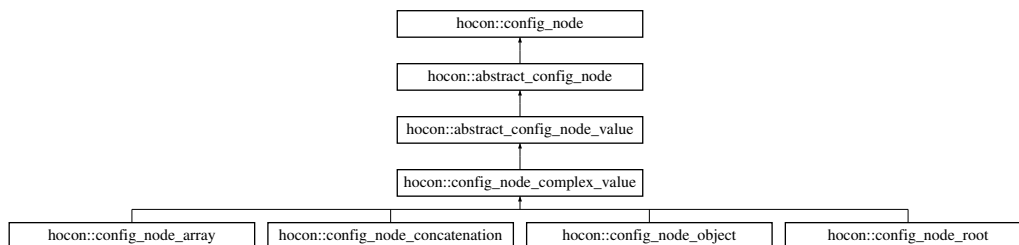
Implements [hocon::config\\_node](#).

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

- internal/nodes/config\_node\_comment.hpp

## 7.25 hocon::config\_node\_complex\_value Class Reference

Inheritance diagram for hocon::config\_node\_complex\_value:



### Public Member Functions

- **config\_node\_complex\_value** (shared\_node\_list children)
- token\_list **get\_tokens** ( ) const override
- shared\_node\_list const & **children** ( ) const
- std::shared\_ptr< const [config\\_node\\_complex\\_value](#) > **indent\_text** (shared\_node indentation) const
- virtual std::shared\_ptr< const [config\\_node\\_complex\\_value](#) > **new\_node** (shared\_node\_list nodes) const =0
- std::string **render** ( ) const
- *The original text of the input which was used to form this particular node.*
- bool **operator==** (const [abstract\\_config\\_node](#) &other) const

### 7.25.1 Detailed Description

Definition at line 7 of file config\_node\_complex\_value.hpp.

## 7.25.2 Member Function Documentation

### 7.25.2.1 render()

```
std::string hocon::abstract_config_node::render ( ) const [virtual], [inherited]
```

The original text of the input which was used to form this particular node.

#### Returns

the original text used to form this node as a String

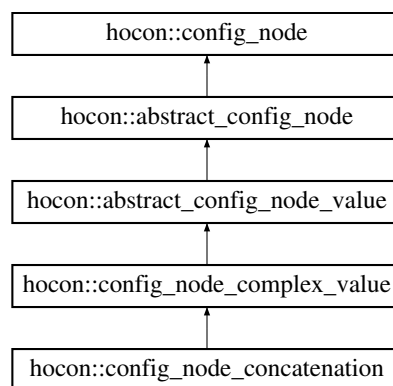
Implements [hocon::config\\_node](#).

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

- internal/nodes/config\_node\_complex\_value.hpp

## 7.26 hocon::config\_node\_concatenation Class Reference

Inheritance diagram for hocon::config\_node\_concatenation:



### Public Member Functions

- **config\_node\_concatenation** (shared\_node\_list children)
- std::shared\_ptr< const [config\\_node\\_complex\\_value](#) > **new\_node** (shared\_node\_list nodes) const override
- token\_list **get\_tokens** () const override
- shared\_node\_list const & **children** () const
- std::shared\_ptr< const [config\\_node\\_complex\\_value](#) > **indent\_text** (shared\_node indentation) const
- std::string **render** () const
 

*The original text of the input which was used to form this particular node.*
- bool **operator==** (const [abstract\\_config\\_node](#) &other) const

### 7.26.1 Detailed Description

Definition at line 7 of file `config_node_concatenation.hpp`.

### 7.26.2 Member Function Documentation

#### 7.26.2.1 `render()`

```
std::string hocon::abstract_config_node::render ( ) const [virtual], [inherited]
```

The original text of the input which was used to form this particular node.

#### Returns

the original text used to form this node as a String

Implements [hocon::config\\_node](#).

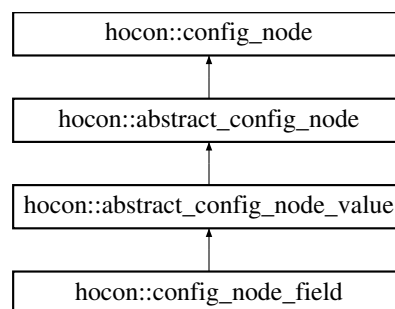
The documentation for this class was generated from the following file:

- `internal/nodes/config_node_concatenation.hpp`

## 7.27 `hocon::config_node_field` Class Reference

A field represents a key-value pair of the format "key : value", where key is a quoted or unquoted string, and value can be any node type.

Inheritance diagram for `hocon::config_node_field`:



### Public Member Functions

- **`config_node_field`** (`shared_node_list` children)
- `token_list` **`get_tokens`** ( ) const override
- `std::shared_ptr< const config\_node\_field >` **`replace_value`** (`shared_node_value` new\_value) const
- `shared_node_value` **`get_value`** ( ) const
- `shared_token` **`separator`** ( ) const
- `std::vector< std::string >` **`comments`** ( ) const
- `std::shared_ptr< const config\_node\_path >` **`path`** ( ) const
- `std::string` **`render`** ( ) const

*The original text of the input which was used to form this particular node.*

- `bool` **`operator==`** (const [abstract\\_config\\_node](#) &other) const

### 7.27.1 Detailed Description

A field represents a key-value pair of the format "key : value", where key is a quoted or unquoted string, and value can be any node type.

Definition at line 13 of file config\_node\_field.hpp.

### 7.27.2 Member Function Documentation

#### 7.27.2.1 render()

```
std::string hocon::abstract_config_node::render ( ) const [virtual], [inherited]
```

The original text of the input which was used to form this particular node.

#### Returns

the original text used to form this node as a String

Implements [hocon::config\\_node](#).

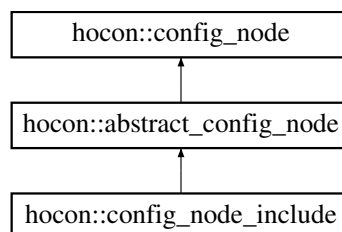
The documentation for this class was generated from the following file:

- internal/nodes/config\_node\_field.hpp

## 7.28 hocon::config\_node\_include Class Reference

Represents an include statement of the form "include include\_kind(include\_path)".

Inheritance diagram for hocon::config\_node\_include:



### Public Member Functions

- **config\_node\_include** (shared\_node\_list children, config\_include\_kind kind)
- token\_list **get\_tokens** ( ) const override
- shared\_node\_list const & **children** ( ) const
- config\_include\_kind **kind** ( ) const
- std::string **name** ( ) const
- std::string **render** ( ) const

*The original text of the input which was used to form this particular node.*

- bool **operator==** (const [abstract\\_config\\_node](#) &other) const

### 7.28.1 Detailed Description

Represents an include statement of the form "include include\_kind(include\_path)".

Definition at line 10 of file config\_node\_include.hpp.

### 7.28.2 Member Function Documentation

#### 7.28.2.1 render()

```
std::string hocon::abstract_config_node::render ( ) const [virtual], [inherited]
```

The original text of the input which was used to form this particular node.

#### Returns

the original text used to form this node as a String

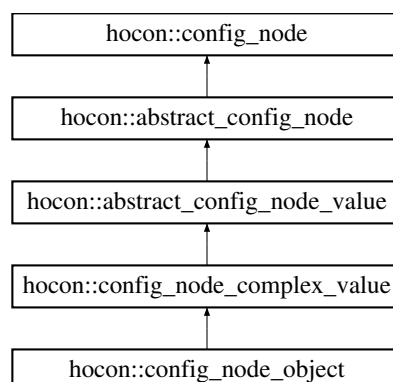
Implements [hocon::config\\_node](#).

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

- internal/nodes/config\_node\_include.hpp

## 7.29 hocon::config\_node\_object Class Reference

Inheritance diagram for hocon::config\_node\_object:



## Public Member Functions

- **config\_node\_object** (shared\_node\_list children)
- std::shared\_ptr< const [config\\_node\\_complex\\_value](#) > **new\_node** (shared\_node\_list nodes) const override
- bool **has\_value** ([path](#) desired\_path) const
- shared\_node\_object **change\_value\_on\_path** ([path](#) desired\_path, shared\_node\_value [value](#), config\_syntax flavor) const
- shared\_node\_object **set\_value\_on\_path** (std::string desired\_path, shared\_node\_value [value](#), config\_syntax flavor=config\_syntax::CONF) const
- shared\_node\_object **set\_value\_on\_path** ([config\\_node\\_path](#) desired\_path, shared\_node\_value [value](#), config\_syntax flavor=config\_syntax::CONF) const
- shared\_node\_list **indentation** () const
- shared\_node\_object **add\_value\_on\_path** ([config\\_node\\_path](#) desired\_path, shared\_node\_value [value](#), config\_syntax flavor) const
- shared\_node\_object **remove\_value\_on\_path** (std::string desired\_path, config\_syntax flavor) const
- token\_list **get\_tokens** () const override
- shared\_node\_list const & **children** () const
- std::shared\_ptr< const [config\\_node\\_complex\\_value](#) > **indent\_text** (shared\_node indentation) const
- std::string **render** () const

*The original text of the input which was used to form this particular node.*

- bool **operator==** (const [abstract\\_config\\_node](#) &other) const

## Static Public Member Functions

- static bool **contains\_token** (shared\_node node, token\_type [token](#))

### 7.29.1 Detailed Description

Definition at line 13 of file config\_node\_object.hpp.

### 7.29.2 Member Function Documentation

#### 7.29.2.1 render()

```
std::string hocon::abstract_config_node::render ( ) const [virtual], [inherited]
```

The original text of the input which was used to form this particular node.

#### Returns

the original text used to form this node as a String

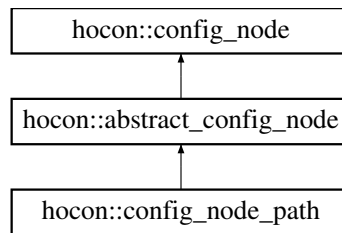
Implements [hocon::config\\_node](#).

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

- internal/nodes/config\_node\_object.hpp

## 7.30 hocon::config\_node\_path Class Reference

Inheritance diagram for hocon::config\_node\_path:



### Public Member Functions

- **config\_node\_path** ([path](#) node\_path, token\_list [tokens](#))
- token\_list **get\_tokens** () const override
- [path](#) **get\_path** () const
- [config\\_node\\_path](#) **sub\_path** (int to\_remove)
- [config\\_node\\_path](#) **first** ()
- std::string **render** () const
  - The original text of the input which was used to form this particular node.*
- bool **operator==** (const [abstract\\_config\\_node](#) &other) const

### 7.30.1 Detailed Description

Definition at line 9 of file config\_node\_path.hpp.

### 7.30.2 Member Function Documentation

#### 7.30.2.1 render()

```
std::string hocon::abstract_config_node::render ( ) const [virtual], [inherited]
```

The original text of the input which was used to form this particular node.

#### Returns

the original text used to form this node as a String

Implements [hocon::config\\_node](#).

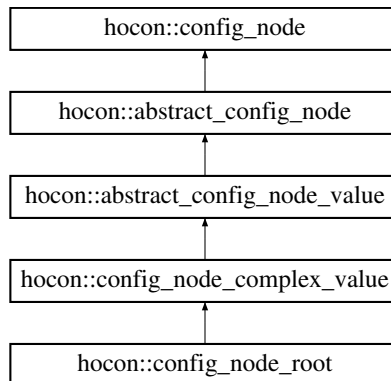
The documentation for this class was generated from the following file:

- internal/nodes/config\_node\_path.hpp



## 7.31 hocon::config\_node\_root Class Reference

Inheritance diagram for hocon::config\_node\_root:



### Public Member Functions

- **config\_node\_root** (shared\_node\_list children, shared\_origin origin)
- std::shared\_ptr< const [config\\_node\\_complex\\_value](#) > **new\_node** (shared\_node\_list nodes) const override
- std::shared\_ptr< const [config\\_node\\_complex\\_value](#) > **value** () const
- std::shared\_ptr< const [config\\_node\\_root](#) > **set\_value** (std::string desired\_path, shared\_node\_value, config\_syntax flavor) const
- bool **has\_value** (std::string desired\_path) const
- token\_list **get\_tokens** () const override
- shared\_node\_list const & **children** () const
- std::shared\_ptr< const [config\\_node\\_complex\\_value](#) > **indent\_text** (shared\_node indentation) const
- std::string **render** () const
  - The original text of the input which was used to form this particular node.*
- bool **operator==** (const [abstract\\_config\\_node](#) &other) const

### 7.31.1 Detailed Description

Definition at line 8 of file config\_node\_root.hpp.

### 7.31.2 Member Function Documentation

#### 7.31.2.1 render()

```
std::string hocon::abstract_config_node::render ( ) const [virtual], [inherited]
```

The original text of the input which was used to form this particular node.

#### Returns

the original text used to form this node as a String

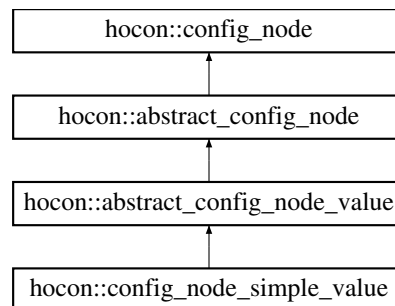
Implements [hocon::config\\_node](#).

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

- internal/nodes/config\_node\_root.hpp

## 7.32 hocon::config\_node\_simple\_value Class Reference

Inheritance diagram for hocon::config\_node\_simple\_value:



### Public Member Functions

- **config\_node\_simple\_value** (shared\_token [value](#))
  - shared\_token **get\_token** () const
  - shared\_value **get\_value** () const
  - token\_list **get\_tokens** () const override
  - std::string **render** () const
- The original text of the input which was used to form this particular node.*
- bool **operator==** (const [abstract\\_config\\_node](#) &other) const

### 7.32.1 Detailed Description

Definition at line 8 of file config\_node\_simple\_value.hpp.

### 7.32.2 Member Function Documentation

#### 7.32.2.1 render()

```
std::string hocon::abstract_config_node::render ( ) const [virtual], [inherited]
```

The original text of the input which was used to form this particular node.

#### Returns

the original text used to form this node as a String

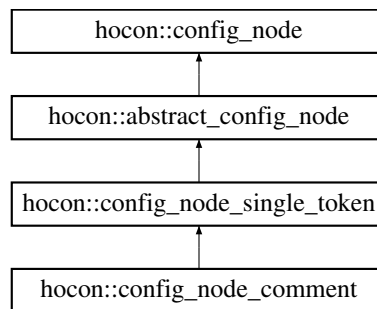
Implements [hocon::config\\_node](#).

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

- internal/nodes/config\_node\_simple\_value.hpp

## 7.33 hocon::config\_node\_single\_token Class Reference

Inheritance diagram for hocon::config\_node\_single\_token:



### Public Member Functions

- **config\_node\_single\_token** (shared\_token t)
  - token\_list **get\_tokens** () const override
  - shared\_token **get\_token** () const
  - std::string **render** () const
- The original text of the input which was used to form this particular node.*
- bool **operator==** (const [abstract\\_config\\_node](#) &other) const

#### 7.33.1 Detailed Description

Definition at line 8 of file config\_node\_single\_token.hpp.

#### 7.33.2 Member Function Documentation

##### 7.33.2.1 render()

```
std::string hocon::abstract_config_node::render ( ) const [virtual], [inherited]
```

The original text of the input which was used to form this particular node.

##### Returns

the original text used to form this node as a String

Implements [hocon::config\\_node](#).

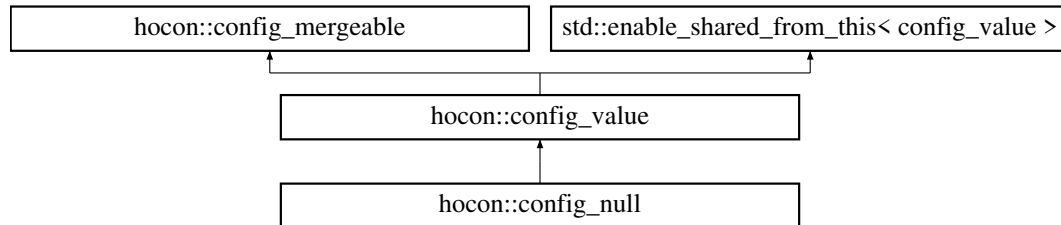
The documentation for this class was generated from the following file:

- internal/nodes/config\_node\_single\_token.hpp

## 7.34 hocon::config\_null Class Reference

This exists because sometimes null is not the same as missing.

Inheritance diagram for hocon::config\_null:



### Public Types

- enum class [type](#) {  
**OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
**CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

*The type of a configuration value (following the [JSON](#) type schema).*

### Public Member Functions

- config\_null** (shared\_origin [origin](#))
- config\_value::type value\_type** () const override  
*The type of the value; matches the JSON type schema.*
- std::string **transform\_to\_string** () const override
- unwrapped\_value **unwrapped** () const override
- bool **operator==** ([config\\_value](#) const &other) const override
- virtual shared\_origin const & **origin** () const  
*The origin of the value (file, line number, etc.), for debugging and error messages.*
- char const \* **value\_type\_name** () const  
*The printable name of the value type.*
- virtual std::string **render** () const  
*Renders the config value as a HOCON string.*
- virtual std::string **render** ([config\\_render\\_options](#) options) const  
*Renders the config value to a string, using the provided options.*
- shared\_config **at\_key** (std::string const &key) const  
*Places the value inside a [config](#) at the given key.*
- shared\_config **at\_path** (std::string const &path\_expression) const  
*Places the value inside a [config](#) at the given path.*
- virtual shared\_value **with\_origin** (shared\_origin [origin](#)) const  
*Returns a.*
- virtual shared\_value **relativized** (std::string prefix) const  
*This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.*
- virtual resolve\_status **get\_resolve\_status** () const
- std::shared\_ptr< const [config\\_mergeable](#) > **with\_fallback** (std::shared\_ptr< const [config\\_mergeable](#) > other) const override  
*Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*

## Static Public Member Functions

- static char const \* **type\_name** (type t)

## Protected Member Functions

- shared\_value **new\_copy** (shared\_origin) const override
- void **render** (std::string &result, int indent, bool at\_root, [config\\_render\\_options](#) options) const override
- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &at\_key, [config\\_render\\_options](#) options) const
- shared\_config **at\_key** (shared\_origin [origin](#), std::string const &key) const
- shared\_config **at\_path** (shared\_origin [origin](#), [path](#) raw\_path) const
- virtual [resolve\\_result](#)< shared\_value > **resolve\_substitutions** ([resolve\\_context](#) const &context, [resolve\\_source](#) const &source) const
- void **require\_not\_ignoring\_fallbacks** () const
- virtual bool **ignores\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const [unmergeable](#) > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const [unmergeable](#) > fallback) const
- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- virtual shared\_value **construct\_delayed\_merge** (shared\_origin [origin](#), std::vector< shared\_value > stack) const
- shared\_value **to\_fallback\_value** () const override

*Converts a config to its root object and a [config\\_value](#) to itself.*

## Static Protected Member Functions

- static void **indent** (std::string &result, int indent, [config\\_render\\_options](#) const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** ([config\\_value](#) const &other, std::function< bool(T const &)> checker)

### 7.34.1 Detailed Description

This exists because sometimes null is not the same as missing.

Specifically, if a value is set to null we can give a better error message (indicating where it was set to null) in case someone asks for the value. Also, null overrides values set "earlier" in the search path, while missing values do not.

Definition at line 17 of file config\_null.hpp.

### 7.34.2 Member Enumeration Documentation

### 7.34.2.1 type

```
enum hocon::config_value::type [strong], [inherited]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file config\_value.hpp.

## 7.34.3 Member Function Documentation

### 7.34.3.1 at\_key()

```
shared_config hocon::config_value::at_key (  
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also `config_value#at_path(string)`.

#### Parameters

<i>key</i>	key to store this value at.
------------	-----------------------------

#### Returns

**a**  
`config`  
instance containing this value at the given key.

### 7.34.3.2 at\_path()

```
shared_config hocon::config_value::at_path (  
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

#### Parameters

<i>path</i>	path to store this value at.
-------------	------------------------------

**Returns**

**a**  
config

instance containing this value at the given path.

**7.34.3.3 origin()**

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

**Returns**

where the value came from

**7.34.3.4 relativized()**

```
virtual shared_value hocon::config_value::relativized (
    std::string prefix ) const [inline], [virtual], [inherited]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at foo.bar in the parent, and the included file as a substitution \${a.b.c}, the included substitution now needs to be \${foo.bar.a.b.c} because we resolve substitutions globally only after parsing everything.

**Parameters**

<i>prefix</i>	
---------------	--

**Returns**

value relativized to the given path or the same value if nothing to do

Reimplemented in [hocon::config\\_concatenation](#), and [hocon::simple\\_config\\_list](#).

Definition at line 181 of file config\_value.hpp.

**7.34.3.5 render() [1/2]**

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to  
[render](#)([config\\_render\\_options](#)())

.

#### Returns

the rendered value

#### 7.34.3.6 [render\(\)](#) [2/2]

```
virtual std::string hocon::config_value::render (
    config\_render\_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

#### Parameters

<i>options</i>	the rendering options
----------------	-----------------------

#### Returns

the rendered value

#### 7.34.3.7 [to\\_fallback\\_value\(\)](#)

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).



### 7.34.3.8 value\_type()

```
config_value::type hocon::config_null::value_type ( ) const [override], [virtual]
```

The type of the value; matches the JSON type schema.

#### Returns

value's type

Implements [hocon::config\\_value](#).

### 7.34.3.9 value\_type\_name()

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

#### Returns

value's type's name

Definition at line 92 of file config\_value.hpp.

### 7.34.3.10 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

## Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

## Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

## 7.34.3.11 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```

Returns a.  
`config_value`

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single [config\\_value](#).

## Parameters

<i>origin</i>	the origin set on the returned value
---------------	--------------------------------------

## Returns

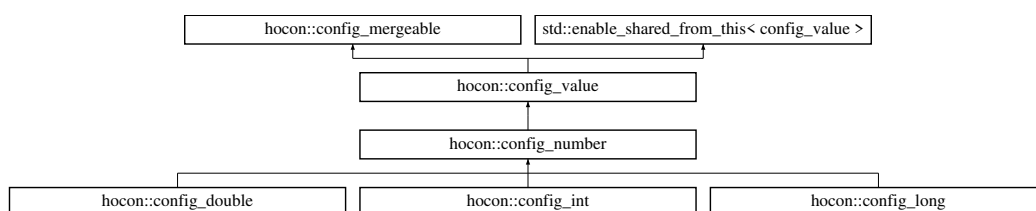
the new [config\\_value](#) with the given origin

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

- `internal/values/config_null.hpp`

## 7.35 hocon::config\_number Class Reference

Inheritance diagram for `hocon::config_number`:



## Public Types

- enum class [type](#) {  
**OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
**CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

The type of a configuration value (following the [JSON](#) type schema).

## Public Member Functions

- config\_number** (shared\_origin [origin](#), std::string original\_text)
- std::string **transform\_to\_string** () const override
- [config\\_value::type](#) **value\_type** () const override

The type of the value; matches the JSON type schema.

- virtual int64\_t **long\_value** () const =0
- virtual double **double\_value** () const =0
- bool **is\_whole** () const
- bool **operator==** (const [config\\_number](#) &other) const
- bool **operator!=** (const [config\\_number](#) &other) const
- bool **operator==** ([config\\_value](#) const &other) const override
- int **int\_value\_range\_checked** (std::string const &path) const
- virtual shared\_origin const & [origin](#) () const

The origin of the value (file, line number, etc.), for debugging and error messages.

- char const \* [value\\_type\\_name](#) () const

The printable name of the value type.

- virtual unwrapped\_value **unwrapped** () const =0
- virtual std::string [render](#) () const
- virtual std::string [render](#) ([config\\_render\\_options](#) options) const

Renders the config value as a HOCON string.

Renders the config value to a string, using the provided options.

- shared\_config [at\\_key](#) (std::string const &key) const
- shared\_config [at\\_path](#) (std::string const &path\_expression) const
- virtual shared\_value [with\\_origin](#) (shared\_origin [origin](#)) const

Places the value inside a [config](#) at the given key.

Places the value inside a [config](#) at the given path.

Returns a.

- virtual shared\_value [relativized](#) (std::string prefix) const

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

- virtual resolve\_status **get\_resolve\_status** () const
- std::shared\_ptr< const [config\\_mergeable](#) > [with\\_fallback](#) (std::shared\_ptr< const [config\\_mergeable](#) > other) const override

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

## Static Public Member Functions

- static std::shared\_ptr< [config\\_number](#) > **new\_number** (shared\_origin [origin](#), int64\_t [value](#), std::string original\_text)
- static std::shared\_ptr< [config\\_number](#) > **new\_number** (shared\_origin [origin](#), double [value](#), std::string original\_text)
- static char const \* **type\_name** ([type](#) t)

## Protected Member Functions

- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &at\_key, [config\\_render\\_options](#) options) const
- virtual void **render** (std::string &result, int indent, bool at\_root, [config\\_render\\_options](#) options) const
- shared\_config **at\_key** (shared\_origin [origin](#), std::string const &key) const
- shared\_config **at\_path** (shared\_origin [origin](#), [path](#) raw\_path) const
- virtual shared\_value **new\_copy** (shared\_origin [origin](#)) const =0
- virtual [resolve\\_result](#)< shared\_value > **resolve\_substitutions** ([resolve\\_context](#) const &context, [resolve\\_source](#) const &source) const
- void **require\_not\_ignoring\_fallbacks** () const
- virtual bool **ignores\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const [unmergeable](#) > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const [unmergeable](#) > fallback) const
- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- virtual shared\_value **construct\_delayed\_merge** (shared\_origin [origin](#), std::vector< shared\_value > stack) const
- shared\_value **to\_fallback\_value** () const override

*Converts a config to its root object and a [config\\_value](#) to itself.*

## Static Protected Member Functions

- static void **indent** (std::string &result, int indent, [config\\_render\\_options](#) const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** ([config\\_value](#) const &other, std::function< bool(T const &)> checker)

## Protected Attributes

- std::string **\_original\_text**

### 7.35.1 Detailed Description

Definition at line 10 of file [config\\_number.hpp](#).

### 7.35.2 Member Enumeration Documentation

### 7.35.2.1 type

```
enum hocon::config_value::type [strong], [inherited]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file config\_value.hpp.

## 7.35.3 Member Function Documentation

### 7.35.3.1 at\_key()

```
shared_config hocon::config_value::at_key (  
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also `config_value#at_path(string)`.

#### Parameters

<i>key</i>	key to store this value at.
------------	-----------------------------

#### Returns

a  
`config`

instance containing this value at the given key.

### 7.35.3.2 at\_path()

```
shared_config hocon::config_value::at_path (  
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

#### Parameters

<i>path</i>	path to store this value at.
-------------	------------------------------

**Returns**

**a**  
config

instance containing this value at the given path.

**7.35.3.3 origin()**

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

**Returns**

where the value came from

**7.35.3.4 relativized()**

```
virtual shared_value hocon::config_value::relativized (
    std::string prefix ) const [inline], [virtual], [inherited]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at foo.bar in the parent, and the included file as a substitution \${a.b.c}, the included substitution now needs to be \${foo.bar.a.b.c} because we resolve substitutions globally only after parsing everything.

**Parameters**

<i>prefix</i>	
---------------	--

**Returns**

value relativized to the given path or the same value if nothing to do

Reimplemented in [hocon::config\\_concatenation](#), and [hocon::simple\\_config\\_list](#).

Definition at line 181 of file config\_value.hpp.

**7.35.3.5 render() [1/2]**

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to  
[render](#)([config\\_render\\_options](#)())

.

#### Returns

the rendered value

#### 7.35.3.6 render() [2/2]

```
virtual std::string hocon::config_value::render (
    config\_render\_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

#### Parameters

<i>options</i>	the rendering options
----------------	-----------------------

#### Returns

the rendered value

#### 7.35.3.7 to\_fallback\_value()

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

### 7.35.3.8 value\_type()

```
config_value::type hocon::config_number::value_type ( ) const [override], [virtual]
```

The type of the value; matches the JSON type schema.

#### Returns

value's type

Implements [hocon::config\\_value](#).

### 7.35.3.9 value\_type\_name()

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

#### Returns

value's type's name

Definition at line 92 of file `config_value.hpp`.

### 7.35.3.10 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only `ConfigObject` and `Config` instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.



## Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

## Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

## 7.35.3.11 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```

Returns a.  
`config_value`

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single [config\\_value](#).

## Parameters

<i>origin</i>	the origin set on the returned value
---------------	--------------------------------------

## Returns

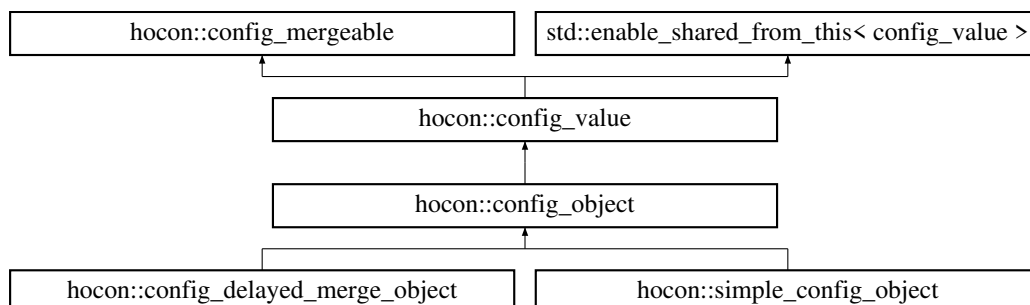
the new [config\\_value](#) with the given origin

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

- `internal/values/config_number.hpp`

## 7.36 hocon::config\_object Class Reference

Inheritance diagram for `hocon::config_object`:



## Public Types

- using **iterator** = std::unordered\_map< std::string, shared\_value >::const\_iterator
- enum class **type** {  
**OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
**CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

*The type of a configuration value (following the **JSON** type schema).*

## Public Member Functions

- virtual std::shared\_ptr< const **config** > **to\_config** () const  
*Converts this object to a Config instance, enabling you to use path expressions to find values in the object.*
- **config\_object** (shared\_origin **origin**)
- **config\_value::type** **value\_type** () const override  
*The type of the value; matches the JSON type schema.*
- virtual shared\_object **with\_value** (**path** raw\_path, shared\_value **value**) const =0
- virtual shared\_object **with\_value** (std::string key, shared\_value **value**) const =0
- virtual shared\_value **attempt\_peek\_with\_partial\_resolve** (std::string const &key) const =0  
*Look up the key on an only-partially-resolved object, with no transformation or type conversion of any kind; if 'this' is not resolved then try to look up the key anyway if possible.*
- virtual std::vector< std::string > **key\_set** () const =0  
*Construct a list of keys in the \_value map.*
- virtual bool **is\_empty** () const =0
- virtual size\_t **size** () const =0
- virtual shared\_value **operator[]** (std::string const &key) const =0
- virtual shared\_value **get** (std::string const &key) const =0
- virtual iterator **begin** () const =0
- virtual iterator **end** () const =0
- virtual shared\_origin const & **origin** () const  
*The origin of the value (file, line number, etc.), for debugging and error messages.*
- char const \* **value\_type\_name** () const  
*The printable name of the value type.*
- virtual unwrapped\_value **unwrapped** () const =0
- virtual std::string **render** () const  
*Renders the config value as a HOCON string.*
- virtual std::string **render** (**config\_render\_options** options) const  
*Renders the config value to a string, using the provided options.*
- shared\_config **at\_key** (std::string const &key) const  
*Places the value inside a **config** at the given key.*
- shared\_config **at\_path** (std::string const &path\_expression) const  
*Places the value inside a **config** at the given path.*
- virtual shared\_value **with\_origin** (shared\_origin **origin**) const  
*Returns a.*
- virtual shared\_value **relativized** (std::string prefix) const  
*This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.*
- virtual resolve\_status **get\_resolve\_status** () const
- std::shared\_ptr< const **config\_mergeable** > **with\_fallback** (std::shared\_ptr< const **config\_mergeable** > other) const override  
*Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*
- virtual bool **operator==** (**config\_value** const &other) const =0
- virtual std::string **transform\_to\_string** () const

## Static Public Member Functions

- static char const \* **type\_name** (type t)

## Protected Member Functions

- shared\_value **peek\_path** (path desired\_path) const
- shared\_value **peek\_assuming\_resolved** (std::string const &key, path original\_path) const
- virtual shared\_object **new\_copy** (resolve\_status const &status, shared\_origin origin) const =0
- shared\_value **new\_copy** (shared\_origin origin) const override
- shared\_value **construct\_delayed\_merge** (shared\_origin origin, std::vector< shared\_value > stack) const override
- virtual std::unordered\_map< std::string, shared\_value > const & **entry\_set** () const =0
- virtual shared\_object **without\_path** (path raw\_path) const =0
- virtual shared\_object **with\_only\_path** (path raw\_path) const =0
- virtual shared\_object **with\_only\_path\_or\_null** (path raw\_path) const =0
- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &at\_key, config\_render\_options options) const
- virtual void **render** (std::string &result, int indent, bool at\_root, config\_render\_options options) const
- shared\_config **at\_key** (shared\_origin origin, std::string const &key) const
- shared\_config **at\_path** (shared\_origin origin, path raw\_path) const
- virtual resolve\_result< shared\_value > **resolve\_substitutions** (resolve\_context const &context, resolve\_source const &source) const
- void **require\_not\_ignoring\_fallbacks** () const
- virtual bool **ignores\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const unmergeable > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const unmergeable > fallback) const
- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- shared\_value **to\_fallback\_value** () const override

*Converts a config to its root object and a config\_value to itself.*

## Static Protected Member Functions

- static shared\_value **peek\_path** (const config\_object \*self, path desired\_path)
- static shared\_origin **merge\_origins** (std::vector< shared\_value > const &stack)
- static void **indent** (std::string &result, int indent, config\_render\_options const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** (config\_value const &other, std::function< bool(T const &)> checker)

## Friends

- class **config**
- class **config\_value**
- class **simple\_config\_object**
- class **resolve\_source**
- class **config\_delayed\_merge\_object**

### 7.36.1 Detailed Description

Definition at line 11 of file config\_object.hpp.

### 7.36.2 Member Enumeration Documentation

#### 7.36.2.1 type

```
enum hocon::config_value::type [strong], [inherited]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file config\_value.hpp.

### 7.36.3 Member Function Documentation

#### 7.36.3.1 at\_key()

```
shared_config hocon::config_value::at_key (  
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also `config_value#at_path(string)`.

##### Parameters

<i>key</i>	key to store this value at.
------------	-----------------------------

##### Returns

**a**  
`config`

instance containing this value at the given key.

#### 7.36.3.2 at\_path()

```
shared_config hocon::config_value::at_path (  
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

## Parameters

<i>path</i>	path to store this value at.
-------------	------------------------------

## Returns

a  
config  
instance containing this value at the given path.

## 7.36.3.3 attempt\_peek\_with\_partial\_resolve()

```
virtual shared_value hocon::config_object::attempt_peek_with_partial_resolve (
    std::string const & key ) const [pure virtual]
```

Look up the key on an only-partially-resolved object, with no transformation or type conversion of any kind; if 'this' is not resolved then try to look up the key anyway if possible.

## Parameters

<i>key</i>	key to look up
------------	----------------

## Returns

the value of the key, or null if known not to exist

## Exceptions

<a href="#"><i>config_exception</i></a>	if can't figure out key's value (or existence) without more resolving
---	---

Implemented in [hocon::simple\\_config\\_object](#), and [hocon::config\\_delayed\\_merge\\_object](#).

## 7.36.3.4 key\_set()

```
virtual std::vector<std::string> hocon::config_object::key_set ( ) const [pure virtual]
```

Construct a list of keys in the \_value map.

Use a vector rather than set, because most of the time we just want to iterate over them.

Implemented in [hocon::simple\\_config\\_object](#), and [hocon::config\\_delayed\\_merge\\_object](#).

### 7.36.3.5 origin()

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

#### Returns

where the value came from

### 7.36.3.6 relativized()

```
virtual shared_value hocon::config_value::relativized (
    std::string prefix ) const [inline], [virtual], [inherited]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at `foo.bar` in the parent, and the included file as a substitution `$(a.b.c)`, the included substitution now needs to be `$(foo.bar.a.b.c)` because we resolve substitutions globally only after parsing everything.

#### Parameters

<i>prefix</i>	
---------------	--

#### Returns

value relativized to the given path or the same value if nothing to do

Reimplemented in [hocon::config\\_concatenation](#), and [hocon::simple\\_config\\_list](#).

Definition at line 181 of file `config_value.hpp`.

### 7.36.3.7 render() [1/2]

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to  
[render](#)(`config_render_options()`)

.

#### Returns

the rendered value

### 7.36.3.8 render() [2/2]

```
virtual std::string hocon::config_value::render (
    config_render_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

#### Parameters

<i>options</i>	the rendering options
----------------	-----------------------

#### Returns

the rendered value

### 7.36.3.9 to\_config()

```
virtual std::shared_ptr<const config> hocon::config_object::to_config ( ) const [virtual]
```

Converts this object to a Config instance, enabling you to use path expressions to find values in the object.

This is a constant-time operation (it is not proportional to the size of the object).

#### Returns

a Config with this object as its root

### 7.36.3.10 to\_fallback\_value()

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

### 7.36.3.11 value\_type()

```
config_value::type hocon::config_object::value_type ( ) const [override], [virtual]
```

The type of the value; matches the JSON type schema.

#### Returns

value's type

Implements [hocon::config\\_value](#).

### 7.36.3.12 value\_type\_name()

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

#### Returns

value's type's name

Definition at line 92 of file `config_value.hpp`.

### 7.36.3.13 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only `ConfigObject` and `Config` instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.



## Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

## Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

## 7.36.3.14 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```

Returns a.  
`config_value`

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single [config\\_value](#).

## Parameters

<i>origin</i>	the origin set on the returned value
---------------	--------------------------------------

## Returns

the new [config\\_value](#) with the given origin

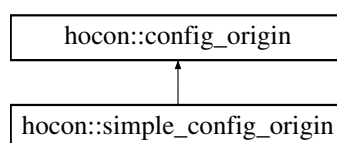
The documentation for this class was generated from the following file:

- `hocon/config_object.hpp`

## 7.37 hocon::config\_origin Class Reference

Represents the origin (such as filename and line number) of a [config\\_value](#) for use in error messages.

Inheritance diagram for `hocon::config_origin`:



## Public Member Functions

- virtual LIBCPP\_HOCON\_EXPORT std::string const & [description](#) () const =0  
*Returns a string describing the origin of a value or exception.*
- virtual LIBCPP\_HOCON\_EXPORT shared\_origin [with\\_line\\_number](#) (int [line\\_number](#)) const =0  
*Returns a.*
- virtual LIBCPP\_HOCON\_EXPORT int [line\\_number](#) () const =0  
*Returns a line number where the value or exception originated.*
- virtual LIBCPP\_HOCON\_EXPORT std::vector< std::string > const & [comments](#) () const =0  
*Returns any comments that appeared to "go with" this place in the file.*
- virtual LIBCPP\_HOCON\_EXPORT shared\_origin [with\\_comments](#) (std::vector< std::string > [comments](#)) const =0  
*Returns a.*

### 7.37.1 Detailed Description

Represents the origin (such as filename and line number) of a [config\\_value](#) for use in error messages.

Obtain the origin of a value with [config\\_value#origin](#). Exceptions may have an origin, see [config\\_exception#origin](#), but be careful because [config\\_exception.origin\(\)](#) may return null.

It's best to use this interface only for debugging; its accuracy is "best effort" rather than guaranteed, and a potentially-noticeable amount of memory could probably be saved if origins were not kept around, so in the future there might be some option to discard origins.

*Do not implement this interface*; it should only be implemented by the config library. Arbitrary implementations will not work because the library internals assume a specific concrete implementation. Also, this interface is likely to grow new methods over time, so third-party implementations will break.

Definition at line 31 of file [config\\_origin.hpp](#).

### 7.37.2 Member Function Documentation

#### 7.37.2.1 [comments\(\)](#)

```
virtual LIBCPP_HOCON_EXPORT std::vector<std::string> const& hocon::config_origin::comments ( )
const [pure virtual]
```

Returns any comments that appeared to "go with" this place in the file.

Often an empty list, but never null. The details of this are subject to change, but at the moment comments that are immediately before an array element or object field, with no blank line after the comment, "go with" that element or field.

#### Returns

any comments that seemed to "go with" this origin, empty list if none

Implemented in [hocon::simple\\_config\\_origin](#).

### 7.37.2.2 description()

```
virtual LIBCPP_HOCON_EXPORT std::string const& hocon::config_origin::description ( ) const
[pure virtual]
```

Returns a string describing the origin of a value or exception.

This will never return null.

#### Returns

string describing the origin

Implemented in [hocon::simple\\_config\\_origin](#).

### 7.37.2.3 line\_number()

```
virtual LIBCPP_HOCON_EXPORT int hocon::config_origin::line_number ( ) const [pure virtual]
```

Returns a line number where the value or exception originated.

This will return -1 if there's no meaningful line number.

#### Returns

line number or -1 if none is available

Implemented in [hocon::simple\\_config\\_origin](#).

### 7.37.2.4 with\_comments()

```
virtual LIBCPP_HOCON_EXPORT shared_origin hocon::config_origin::with_comments (
    std::vector< std::string > comments ) const [pure virtual]
```

Returns a  
config\_origin

based on this one, but with the given comments. Does not modify this instance or any  
config\_value

s with this origin (since they are immutable). To set the returned origin to a  
config\_value

, use [config\\_value#with\\_origin](#).

Note that when the given comments are equal to the comments on this object, a new instance may not be created  
and  
`this`

is returned directly.

#### Since

1.3.0

**Parameters**

<i>comments</i>	the comments used on the returned origin
-----------------	--

**Returns**

the [config\\_origin](#) with the given comments

Implemented in [hocon::simple\\_config\\_origin](#).

**7.37.2.5 with\_line\_number()**

```
virtual LIBCPP_HOCON_EXPORT shared_origin hocon::config_origin::with_line_number (
    int line_number ) const [pure virtual]
```

**Returns a.**

`ConfigOrigin`

based on this one, but with the given line number. This origin must be a FILE, URL or RESOURCE. Does not modify this instance or any

`ConfigValue`

s with this origin (since they are immutable). To set the returned origin to a

`ConfigValue`

, use `ConfigValue#withOrigin`.

Note that when the given lineNumber are equal to the lineNumber on this object, a new instance may not be created

and

`this`

is returned directly.

**Since**

1.3.0

**Parameters**

<i>lineNumber</i>	the new line number
-------------------	---------------------

**Returns**

the created `ConfigOrigin`

Implemented in [hocon::simple\\_config\\_origin](#).

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

- `hocon/config_origin.hpp`

## 7.38 hocon::config\_parse\_options Class Reference

A set of options related to parsing.

### Public Member Functions

- [config\\_parse\\_options](#) ()  
*Gets an instance of [config\\_parse\\_options](#) with all fields set to the default values.*
- [config\\_parse\\_options set\\_syntax](#) (config\_syntax syntax) const  
*Set the file format.*
- config\_syntax const & [get\\_syntax](#) () const  
*Gets the current syntax option.*
- [config\\_parse\\_options set\\_origin\\_description](#) (shared\_string origin\_description) const  
*Set a description for the thing being parsed.*
- shared\_string const & [get\\_origin\\_description](#) () const  
*Gets the current origin description, which may be null for "automatic".*
- [config\\_parse\\_options set\\_allow\\_missing](#) (bool allow\_missing) const  
*Set to false to throw an exception if the item being parsed (for example a file) is missing.*
- bool [get\\_allow\\_missing](#) () const  
*Gets the current "allow missing" flag.*
- [config\\_parse\\_options set\\_includer](#) (shared\_includer includer) const  
*Set a [config\\_includer](#) which customizes how includes are handled.*
- [config\\_parse\\_options prepend\\_includer](#) (shared\_includer includer) const  
*Prepends a [config\\_includer](#) which customizes how includes are handled.*
- [config\\_parse\\_options append\\_includer](#) (shared\_includer includer) const  
*Appends a [config\\_includer](#) which customizes how includes are handled.*
- shared\_includer const & [get\\_includer](#) () const  
*Gets the current includer (will be null for the default includer).*

### Static Public Member Functions

- static [config\\_parse\\_options defaults](#) ()  
*Gets an instance of [ConfigParseOptions](#) with all fields set to the default values.*

#### 7.38.1 Detailed Description

A set of options related to parsing.

This object is immutable, so the "setters" return a new object.

Here is an example of creating a custom [config\\_parse\\_options](#)

:

```
config_parse_options options = config_parse_options()
    .set_syntax(config_syntax.JSON)
    .set_allow_missing(false)
```

ClassLoader is Java-specific, so it was not ported to C++.

Definition at line 25 of file [config\\_parse\\_options.hpp](#).

## 7.38.2 Constructor & Destructor Documentation

### 7.38.2.1 `config_parse_options()`

```
hocon::config_parse_options::config_parse_options ( )
```

Gets an instance of `config_parse_options` with all fields set to the default values.

Start with this instance and make any changes you need.

#### Returns

the default parse options

## 7.38.3 Member Function Documentation

### 7.38.3.1 `append_includer()`

```
config_parse_options hocon::config_parse_options::append_includer (
    shared_includer includer ) const
```

Appends a `config_includer` which customizes how includes are handled.

To append, the library calls {} on the existing includer. includer the includer to append (may not be null) new version of the parse option

### 7.38.3.2 `defaults()`

```
static config_parse_options hocon::config_parse_options::defaults ( ) [static]
```

Gets an instance of `ConfigParseOptions` with all fields set to the default values.

Start with this instance and make any changes you need.

#### Returns

the default parse options

### 7.38.3.3 get\_allow\_missing()

```
bool hocon::config_parse_options::get_allow_missing ( ) const
```

Gets the current "allow missing" flag.

#### Returns

whether we allow missing files

### 7.38.3.4 get\_includer()

```
shared_includer const& hocon::config_parse_options::get_includer ( ) const
```

Gets the current includer (will be null for the default includer).

#### Returns

current includer or null

### 7.38.3.5 get\_origin\_description()

```
shared_string const& hocon::config_parse_options::get_origin_description ( ) const
```

Gets the current origin description, which may be null for "automatic".

#### Returns

the current origin description or null

### 7.38.3.6 get\_syntax()

```
config_syntax const& hocon::config_parse_options::get_syntax ( ) const
```

Gets the current syntax option.

### 7.38.3.7 prepend\_includer()

```
config_parse_options hocon::config_parse_options::prepend_includer (
    shared_includer includer ) const
```

Prepends a [config\\_includer](#) which customizes how includes are handled.

To prepend your includer, the library calls [config\\_includer#with\\_fallback](#) on your includer to append the existing includer to it.

**Parameters**

<i>includer</i>	the includer to prepend (may not be null)
-----------------	---

**Returns**

new version of the parse options with different includer

**7.38.3.8 set\_allow\_missing()**

```
config_parse_options hocon::config_parse_options::set_allow_missing (
    bool allow_missing ) const
```

Set to false to throw an exception if the item being parsed (for example a file) is missing.

Set to true to just return an empty document in that case.

**Parameters**

<i>allow_missing</i>	true to silently ignore missing item
----------------------	--------------------------------------

**Returns**

options with the "allow missing" flag set

**7.38.3.9 set\_includer()**

```
config_parse_options hocon::config_parse_options::set_includer (
    shared_includer includer ) const
```

Set a [config\\_includer](#) which customizes how includes are handled.

null means to use the default includer.

**Parameters**

<i>includer</i>	the includer to use or null for default
-----------------	---

**Returns**

new version of the parse options with different includer



### 7.38.3.10 set\_origin\_description()

```
config_parse_options hocon::config_parse_options::set_origin_description (
    shared_string origin_description ) const
```

Set a description for the thing being parsed.

In most cases this will be set up for you to something like the filename, but if you provide just an input stream you might want to improve on it. Set to null to allow the library to come up with something automatically. This description is the basis for the [config\\_origin](#) of the parsed values.

#### Parameters

<i>origin_description</i>	description to put in the <a href="#">config_origin</a>
---------------------------	---

#### Returns

options with the origin description set

### 7.38.3.11 set\_syntax()

```
config_parse_options hocon::config_parse_options::set_syntax (
    config_syntax syntax ) const
```

Set the file format.

If set to null, try to guess from any available filename extension; if guessing fails, assume [config\\_syntax#CONF](#).

#### Parameters

<i>syntax</i>	a syntax or <i>nullptr</i> for best guess
---------------	---

#### Returns

options with the syntax set

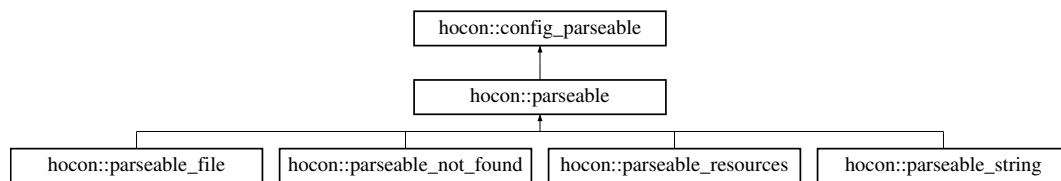
The documentation for this class was generated from the following file:

- hocon/config\_parse\_options.hpp

## 7.39 hocon::config\_parseable Class Reference

An opaque handle to something that can be parsed, obtained from [config\\_include\\_context](#).

Inheritance diagram for `hocon::config_parseable`:



## Public Member Functions

- virtual shared\_object `parse` (`config_parse_options` const &`options`) const =0  
*Parse whatever it is.*
- virtual shared\_origin `origin` () const =0  
*Returns a `config_origin` describing the origin of the parseable item.*
- virtual `config_parse_options` const & `options` () const =0  
*Get the initial options, which can be modified then passed to `parse()`.*

### 7.39.1 Detailed Description

An opaque handle to something that can be parsed, obtained from `config_include_context`.

*Do not implement this interface;* it should only be implemented by the config library. Arbitrary implementations will not work because the library internals assume a specific concrete implementation. Also, this interface is likely to grow new methods over time, so third-party implementations will break.

Definition at line 19 of file `config_parseable.hpp`.

### 7.39.2 Member Function Documentation

#### 7.39.2.1 `options()`

```
virtual config_parse_options const& hocon::config_parseable::options ( ) const [pure virtual]
```

Get the initial options, which can be modified then passed to `parse()`.

These options will have the right description, includer, and other parameters already set up.

#### Returns

the initial options

Implemented in `hocon::parseable`.

### 7.39.2.2 origin()

```
virtual shared_origin hocon::config_parseable::origin ( ) const [pure virtual]
```

Returns a [config\\_origin](#) describing the origin of the parseable item.

Implemented in [hocon::parseable](#).

### 7.39.2.3 parse()

```
virtual shared_object hocon::config_parseable::parse (
    config_parse_options const & options ) const [pure virtual]
```

Parse whatever it is.

The options should come from [config\\_parseable#options\(\)](#) but you could tweak them if you like.

#### Parameters

<i>options</i>	parse options, should be based on the ones from <a href="#">config_parseable#options()</a>
----------------	--

#### Returns

the parsed object

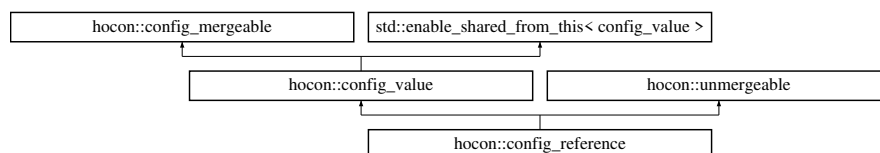
Implemented in [hocon::parseable](#).

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

- [hocon/config\\_parseable.hpp](#)

## 7.40 hocon::config\_reference Class Reference

Inheritance diagram for hocon::config\_reference:



### Public Types

- enum class [type](#) {  
**OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
**CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

*The type of a configuration value (following the [JSON](#) type schema).*

## Public Member Functions

- **config\_reference** (shared\_origin [origin](#), std::shared\_ptr< [substitution\\_expression](#) > expr, int prefix\_↔length=0)
- **type\_value\_type** () const override  
*The type of the value; matches the JSON type schema.*
- std::vector< shared\_value > **unmerged\_values** () const override
- resolve\_status **get\_resolve\_status** () const override
- unwrapped\_value **unwrapped** () const override
- std::shared\_ptr< [substitution\\_expression](#) > **expression** () const
- bool **operator==** ([config\\_value](#) const &other) const override
- virtual shared\_origin const & [origin](#) () const  
*The origin of the value (file, line number, etc.), for debugging and error messages.*
- char const \* [value\\_type\\_name](#) () const  
*The printable name of the value type.*
- virtual std::string [render](#) () const  
*Renders the config value as a HOCON string.*
- virtual std::string [render](#) ([config\\_render\\_options](#) options) const  
*Renders the config value to a string, using the provided options.*
- shared\_config [at\\_key](#) (std::string const &key) const  
*Places the value inside a [config](#) at the given key.*
- shared\_config [at\\_path](#) (std::string const &path\_expression) const  
*Places the value inside a [config](#) at the given path.*
- virtual shared\_value [with\\_origin](#) (shared\_origin [origin](#)) const  
*Returns a.*
- virtual shared\_value [relativized](#) (std::string prefix) const  
*This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.*
- std::shared\_ptr< const [config\\_mergeable](#) > [with\\_fallback](#) (std::shared\_ptr< const [config\\_mergeable](#) > other) const override  
*Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*
- virtual std::string **transform\_to\_string** () const

## Static Public Member Functions

- static char const \* **type\_name** ([type](#) t)

## Protected Member Functions

- shared\_value **new\_copy** (shared\_origin [origin](#)) const override
- [resolve\\_result](#)< shared\_value > **resolve\_substitutions** ([resolve\\_context](#) const &context, [resolve\\_source](#) const &source) const override
- bool **ignores\_fallbacks** () const override
- void **render** (std::string &s, int indent, bool at\_root, [config\\_render\\_options](#) options) const override
- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &[at\\_key](#), [config\\_render\\_options](#) options) const
- shared\_config **at\_key** (shared\_origin [origin](#), std::string const &key) const
- shared\_config **at\_path** (shared\_origin [origin](#), [path](#) raw\_path) const
- void **require\_not\_ignoring\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const

- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const [unmergeable](#) > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const [unmergeable](#) > fallback) const
- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- virtual shared\_value **construct\_delayed\_merge** (shared\_origin [origin](#), std::vector< shared\_value > stack) const
- shared\_value **to\_fallback\_value** () const override

*Converts a config to its root object and a [config\\_value](#) to itself.*

## Static Protected Member Functions

- static void **indent** (std::string &result, int indent, [config\\_render\\_options](#) const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** ([config\\_value](#) const &other, std::function< bool(T const &)> checker)

### 7.40.1 Detailed Description

Definition at line 10 of file config\_reference.hpp.

### 7.40.2 Member Enumeration Documentation

#### 7.40.2.1 type

```
enum hocon::config\_value::type [strong], [inherited]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file config\_value.hpp.

### 7.40.3 Member Function Documentation

#### 7.40.3.1 at\_key()

```
shared_config hocon::config_value::at_key (
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also config\_value#at\_path(string).

**Parameters**

<i>key</i>	key to store this value at.
------------	-----------------------------

**Returns**

**a**  
`config`

instance containing this value at the given key.

**7.40.3.2 at\_path()**

```
shared_config hocon::config_value::at_path (  
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

**Parameters**

<i>path</i>	path to store this value at.
-------------	------------------------------

**Returns**

**a**  
`config`

instance containing this value at the given path.

**7.40.3.3 origin()**

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

**Returns**

where the value came from

#### 7.40.3.4 relativized()

```
virtual shared_value hocon::config_value::relativized (
    std::string prefix ) const [inline], [virtual], [inherited]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at `foo.bar` in the parent, and the included file as a substitution `$(a.b.c)`, the included substitution now needs to be `$(foo.bar.a.b.c)` because we resolve substitutions globally only after parsing everything.

## Parameters

<i>prefix</i>	
---------------	--

## Returns

value relativized to the given path or the same value if nothing to do

Reimplemented in [hocon::config\\_concatenation](#), and [hocon::simple\\_config\\_list](#).

Definition at line 181 of file `config_value.hpp`.

**7.40.3.5 render() [1/2]**

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to

```
render(config_render_options())
```

.

## Returns

the rendered value

**7.40.3.6 render() [2/2]**

```
virtual std::string hocon::config_value::render (
    config\_render\_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.



**Parameters**

<i>options</i>	the rendering options
----------------	-----------------------

**Returns**

the rendered value

**7.40.3.7 to\_fallback\_value()**

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],  
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

**7.40.3.8 value\_type()**

```
type hocon::config_reference::value_type ( ) const [override], [virtual]
```

The type of the value; matches the JSON type schema.

**Returns**

value's type

Implements [hocon::config\\_value](#).

**7.40.3.9 value\_type\_name()**

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

**Returns**

value's type's name

Definition at line 92 of file config\_value.hpp.

### 7.40.3.10 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

#### Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

#### Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

### 7.40.3.11 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```

Returns a.  
config\_value

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single [config\\_value](#).

#### Parameters

<i>origin</i>	the origin set on the returned value
---------------	--------------------------------------

#### Returns

the new [config\\_value](#) with the given origin

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

- internal/values/config\_reference.hpp

## 7.41 hocon::config\_render\_options Class Reference

### Public Member Functions

- [config\\_render\\_options](#) (bool origin\_comments=true, bool comments=true, bool formatted=true, bool json=true)  
*Leaving the default arguments will result in a verbose rendering, which contains comments and therefore is not valid JSON.*
- [config\\_render\\_options set\\_comments](#) (bool value)  
*Returns options with comments toggled.*
- bool [get\\_comments](#) () const  
*Returns whether the options enable comments.*
- [config\\_render\\_options set\\_origin\\_comments](#) (bool value)  
*Returns options with origin comments toggled.*
- bool [get\\_origin\\_comments](#) () const  
*Returns whether the options enable automated origin comments.*
- [config\\_render\\_options set\\_formatted](#) (bool value)  
*Returns options with formatting toggled.*
- bool [get\\_formatted](#) () const  
*Returns whether the options enable formatting.*
- [config\\_render\\_options set\\_json](#) (bool value)  
*Returns options with JSON toggled.*
- bool [get\\_json](#) () const  
*Returns whether the options enable JSON.*

### Static Public Member Functions

- static [config\\_render\\_options concise](#) ()  
*Returns concise render options (no whitespace or comments).*

### 7.41.1 Detailed Description

A set of options related to rendering a [config\\_value](#). Passed to `config_value#render(config_render_options)`.

Here is an example of creating a [config\\_render\\_options](#)

:

```
config_render_options options =  
    config_render_options().set_comments(false)
```

Definition at line 20 of file `config_render_options.hpp`.

### 7.41.2 Constructor & Destructor Documentation

#### 7.41.2.1 config\_render\_options()

```
hocon::config_render_options::config_render_options (  
    bool origin_comments = true,  
    bool comments = true,  
    bool formatted = true,  
    bool json = true )
```

Leaving the default arguments will result in a verbose rendering, which contains comments and therefore is not valid JSON.

See [config\\_render\\_options#concise](#) for stripped-down options.

### 7.41.3 Member Function Documentation

#### 7.41.3.1 concise()

```
static config_render_options hocon::config_render_options::concise ( ) [static]
```

Returns concise render options (no whitespace or comments).

For a resolved [config](#), the concise rendering will be valid JSON.

#### Returns

the concise render options

### 7.41.3.2 get\_comments()

```
bool hocon::config_render_options::get_comments ( ) const
```

Returns whether the options enable comments.

This method is mostly used by the config lib internally, not by applications.

#### Returns

true if comments should be rendered

### 7.41.3.3 get\_formatted()

```
bool hocon::config_render_options::get_formatted ( ) const
```

Returns whether the options enable formatting.

This method is mostly used by the config lib internally, not by applications.

#### Returns

true if the options enable formatting

### 7.41.3.4 get\_json()

```
bool hocon::config_render_options::get_json ( ) const
```

Returns whether the options enable JSON.

This method is mostly used by the config lib internally, not by applications.

#### Returns

true if only JSON should be rendered

### 7.41.3.5 get\_origin\_comments()

```
bool hocon::config_render_options::get_origin_comments ( ) const
```

Returns whether the options enable automated origin comments.

This method is mostly used by the config lib internally, not by applications.

#### Returns

true if origin comments should be rendered

### 7.41.3.6 set\_comments()

```
config_render_options hocon::config_render_options::set_comments (
    bool value )
```

Returns options with comments toggled.

This controls human-written comments but not the autogenerated "origin of this setting" comments, which are controlled by [config\\_render\\_options#set\\_origin\\_comments](#).

**Parameters**

<i>value</i>	true to include comments in the render
--------------	--

**Returns**

options with requested setting for comments

**7.41.3.7 set\_formatted()**

```
config_render_options hocon::config_render_options::set_formatted (
    bool value )
```

Returns options with formatting toggled.

Formatting means indentation and whitespace, enabling formatting makes things prettier but larger.

**Parameters**

<i>value</i>	true to enable formatting
--------------	---------------------------

**Returns**

options with requested setting for formatting

**7.41.3.8 set\_json()**

```
config_render_options hocon::config_render_options::set_json (
    bool value )
```

Returns options with JSON toggled.

JSON means that HOCON extensions (omitting commas, quotes for example) won't be used. However, whether to use comments is controlled by the separate `set_comments(boolean)` and `set_origin_comments(boolean)` options. So if you enable comments you will get invalid JSON despite setting this to true.

**Parameters**

<i>value</i>	true to include non-JSON extensions in the render
--------------	---

**Returns**

options with requested setting for JSON

### 7.41.3.9 set\_origin\_comments()

```
config_render_options hocon::config_render_options::set_origin_comments (
    bool value )
```

Returns options with origin comments toggled.

If this is enabled, the library generates comments for each setting based on the [config\\_value#origin](#) of that setting's value. For example these comments might tell you which file a setting comes from.

```
set_origin_comments ()
```

controls only these autogenerated "origin of this setting" comments, to toggle regular comments use [config\\_render\\_options#set\\_comments](#).

#### Parameters

<i>value</i>	true to include autogenerated setting-origin comments in the render
--------------	---

#### Returns

options with origin comments toggled

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

- [hocon/config\\_render\\_options.hpp](#)

## 7.42 hocon::config\_resolve\_options Class Reference

A set of options related to resolving substitutions.

### Public Member Functions

- [config\\_resolve\\_options](#) (bool use\_system\_environment=true, bool allow\_unresolved=false)  
*Returns the default resolve options.*
- [config\\_resolve\\_options set\\_use\\_system\\_environment](#) (bool *value*) const  
*Returns resolve options that disable any reference to "system" data (currently, this means environment variables).*
- bool [get\\_use\\_system\\_environment](#) () const  
*Returns whether the options enable use of system environment variables.*
- [config\\_resolve\\_options set\\_allow\\_unresolved](#) (bool *value*) const  
*Returns options with "allow unresolved" set to the given value.*
- bool [get\\_allow\\_unresolved](#) () const  
*Returns whether the options allow unresolved substitutions.*

### 7.42.1 Detailed Description

A set of options related to resolving substitutions.

Substitutions use the `${foo.bar}` syntax and are documented in the [HOCON](#) spec.

Typically this class would be used with the method `config#resolve(config_resolve_options)`.

This object is immutable, so the "setters" return a new object.

Here is an example of creating a custom `config_resolve_options`

:

```
config_resolve_options options = config_resolve_options()
    .set_use_system_environment(false)
```

<p><blockquote>

In addition to `config_resolve_options`, there's a prebuilt `config_resolve_options#no_system` which avoids looking at any system environment variables or other external system information. (Right now, environment variables are the only example.)

Definition at line 30 of file `config_resolve_options.hpp`.

### 7.42.2 Constructor & Destructor Documentation

#### 7.42.2.1 config\_resolve\_options()

```
hocon::config_resolve_options::config_resolve_options (
    bool use_system_environment = true,
    bool allow_unresolved = false )
```

Returns the default resolve options.

By default the system environment will be used and unresolved substitutions are not allowed.

Returns

the default resolve options

### 7.42.3 Member Function Documentation



### 7.42.3.1 get\_allow\_unresolved()

```
bool hocon::config_resolve_options::get_allow_unresolved ( ) const
```

Returns whether the options allow unresolved substitutions.

This method is mostly used by the config lib internally, not by applications.

#### Returns

true if unresolved substitutions are allowed

### 7.42.3.2 get\_use\_system\_environment()

```
bool hocon::config_resolve_options::get_use_system_environment ( ) const
```

Returns whether the options enable use of system environment variables.

This method is mostly used by the config lib internally, not by applications.

#### Returns

true if environment variables should be used

### 7.42.3.3 set\_allow\_unresolved()

```
config_resolve_options hocon::config_resolve_options::set_allow_unresolved (
    bool value ) const
```

Returns options with "allow unresolved" set to the given value.

By default, unresolved substitutions are an error. If unresolved substitutions are allowed, then a future attempt to use the unresolved value may fail, but config#resolve(config\_resolve\_options) itself will not throw.

#### Parameters

<i>value</i>	true to silently ignore unresolved substitutions.
--------------	---

#### Returns

options with requested setting for whether to allow substitutions

### 7.42.3.4 set\_use\_system\_environment()

```
config_resolve_options hocon::config_resolve_options::set_use_system_environment (
    bool value ) const
```

Returns resolve options that disable any reference to "system" data (currently, this means environment variables).

#### Returns

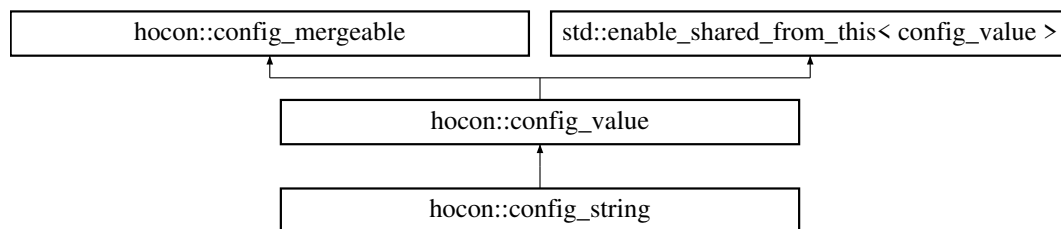
the resolve options with env variables disabled

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

- hocon/config\_resolve\_options.hpp

## 7.43 hocon::config\_string Class Reference

Inheritance diagram for hocon::config\_string:



### Public Types

- enum class `type` {  
**OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
**CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }  
*The type of a configuration value (following the [JSON](#) type schema).*

### Public Member Functions

- **config\_string** (shared\_origin `origin`, std::string text, config\_string\_type quoted)
- `config_value::type value_type` () const override  
*The type of the value; matches the JSON type schema.*
- std::string **transform\_to\_string** () const override
- unwrapped\_value **unwrapped** () const override
- bool **was\_quoted** () const
- bool **operator==** (config\_value const &other) const override
- virtual shared\_origin const & **origin** () const  
*The origin of the value (file, line number, etc.), for debugging and error messages.*
- char const \* **value\_type\_name** () const  
*The printable name of the value type.*
- virtual std::string **render** () const

- Renders the config value as a HOCON string.*
- virtual std::string **render** (config\_render\_options options) const  
*Renders the config value to a string, using the provided options.*
- shared\_config **at\_key** (std::string const &key) const  
*Places the value inside a config at the given key.*
- shared\_config **at\_path** (std::string const &path\_expression) const  
*Places the value inside a config at the given path.*
- virtual shared\_value **with\_origin** (shared\_origin origin) const  
*Returns a.*
- virtual shared\_value **relativized** (std::string prefix) const  
*This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.*
- virtual resolve\_status **get\_resolve\_status** () const
- std::shared\_ptr< const config\_mergeable > **with\_fallback** (std::shared\_ptr< const config\_mergeable > other) const override  
*Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*

## Static Public Member Functions

- static char const \* **type\_name** (type t)

## Protected Member Functions

- shared\_value **new\_copy** (shared\_origin) const override
- void **render** (std::string &s, int indent, bool at\_root, config\_render\_options options) const override
- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &at\_key, config\_render\_options options) const
- shared\_config **at\_key** (shared\_origin origin, std::string const &key) const
- shared\_config **at\_path** (shared\_origin origin, path raw\_path) const
- virtual resolve\_result< shared\_value > **resolve\_substitutions** (resolve\_context const &context, resolve\_source const &source) const
- void **require\_not\_ignoring\_fallbacks** () const
- virtual bool **ignores\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const unmergeable > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const unmergeable > fallback) const
- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- virtual shared\_value **construct\_delayed\_merge** (shared\_origin origin, std::vector< shared\_value > stack) const
- shared\_value **to\_fallback\_value** () const override  
*Converts a config to its root object and a config\_value to itself.*

## Static Protected Member Functions

- static void **indent** (std::string &result, int indent, [config\\_render\\_options](#) const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** ([config\\_value](#) const &other, std::function< bool(T const &)> checker)

### 7.43.1 Detailed Description

Definition at line 10 of file config\_string.hpp.

### 7.43.2 Member Enumeration Documentation

#### 7.43.2.1 type

```
enum hocon::config\_value::type [strong], [inherited]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file config\_value.hpp.

### 7.43.3 Member Function Documentation

#### 7.43.3.1 at\_key()

```
shared_config hocon::config_value::at_key (
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also `config_value#at_path(string)`.

#### Parameters

<i>key</i>	key to store this value at.
------------	-----------------------------

**Returns**

a  
config

instance containing this value at the given key.

**7.43.3.2 at\_path()**

```
shared_config hocon::config_value::at_path (
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

**Parameters**

<i>path</i>	path to store this value at.
-------------	------------------------------

**Returns**

a  
config

instance containing this value at the given path.

**7.43.3.3 origin()**

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

**Returns**

where the value came from

**7.43.3.4 relativized()**

```
virtual shared_value hocon::config_value::relativized (
    std::string prefix ) const [inline], [virtual], [inherited]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at `foo.bar` in the parent, and the included file as a substitution `$(a.b.c)`, the included substitution now needs to be `$(foo.bar.a.b.c)` because we resolve substitutions globally only after parsing everything.

## Parameters

<i>prefix</i>	
---------------	--

## Returns

value relativized to the given path or the same value if nothing to do

Reimplemented in [hocon::config\\_concatenation](#), and [hocon::simple\\_config\\_list](#).

Definition at line 181 of file `config_value.hpp`.

**7.43.3.5 render() [1/2]**

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to

```
render(config_render_options())
```

.

## Returns

the rendered value

**7.43.3.6 render() [2/2]**

```
virtual std::string hocon::config_value::render (
    config\_render\_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

**Parameters**

<i>options</i>	the rendering options
----------------	-----------------------

**Returns**

the rendered value

**7.43.3.7 to\_fallback\_value()**

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],  
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

**7.43.3.8 value\_type()**

```
config_value::type hocon::config_string::value_type ( ) const [override], [virtual]
```

The type of the value; matches the JSON type schema.

**Returns**

value's type

Implements [hocon::config\\_value](#).

**7.43.3.9 value\_type\_name()**

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

**Returns**

value's type's name

Definition at line 92 of file config\_value.hpp.

### 7.43.3.10 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

#### Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

#### Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

### 7.43.3.11 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```



Returns a.  
config\_value

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single config\_value.

#### Parameters

<i>origin</i>	the origin set on the returned value
---------------	--------------------------------------

#### Returns

the new config\_value with the given origin

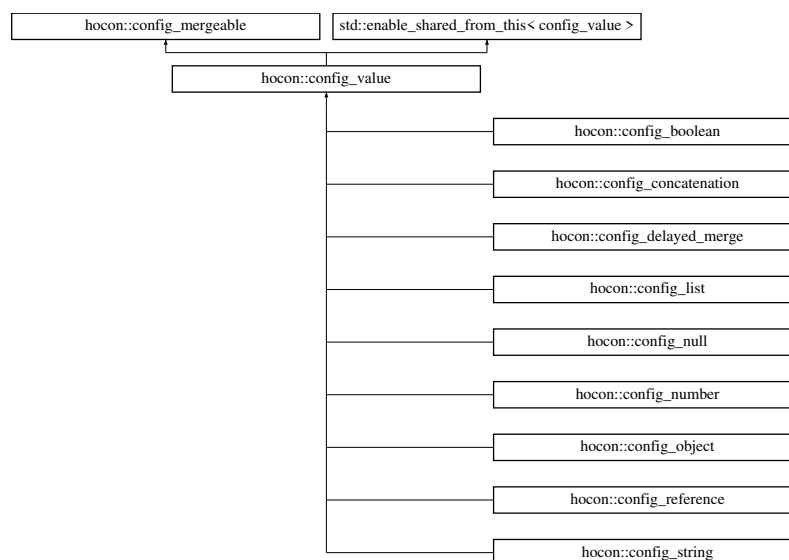
The documentation for this class was generated from the following file:

- internal/values/config\_string.hpp

## 7.44 hocon::config\_value Class Reference

An immutable value, following the JSON type schema.

Inheritance diagram for hocon::config\_value:



### Classes

- class [modifier](#)
- class [no\\_exceptions\\_modifier](#)

## Public Types

- enum class `type` {  
**OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
**CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

The type of a configuration value (following the `JSON` type schema).

## Public Member Functions

- virtual `shared_origin` const & `origin` () const  
The origin of the value (file, line number, etc.), for debugging and error messages.
- virtual `type` `value_type` () const =0  
The type of the value; matches the JSON type schema.
- char const \* `value_type_name` () const  
The printable name of the value type.
- virtual `unwrapped_value` `unwrapped` () const =0
- virtual `std::string` `render` () const  
Renders the config value as a HOCON string.
- virtual `std::string` `render` (`config_render_options` options) const  
Renders the config value to a string, using the provided options.
- `shared_config` `at_key` (`std::string` const &key) const  
Places the value inside a `config` at the given key.
- `shared_config` `at_path` (`std::string` const &path\_expression) const  
Places the value inside a `config` at the given path.
- virtual `shared_value` `with_origin` (`shared_origin` `origin`) const  
Returns a.
- virtual `shared_value` `relativized` (`std::string` prefix) const  
This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- virtual `resolve_status` `get_resolve_status` () const
- `std::shared_ptr`< const `config_mergeable` > `with_fallback` (`std::shared_ptr`< const `config_mergeable` > other) const override  
Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.
- virtual `bool` `operator==` (`config_value` const &other) const =0
- virtual `std::string` `transform_to_string` () const

## Static Public Member Functions

- static char const \* `type_name` (`type` t)

## Protected Member Functions

- `config_value` (`shared_origin` `origin`)
- virtual `void` `render` (`std::string` &result, int indent, bool at\_root, `std::string` const &at\_key, `config_render_options` options) const
- virtual `void` `render` (`std::string` &result, int indent, bool at\_root, `config_render_options` options) const
- `shared_config` `at_key` (`shared_origin` `origin`, `std::string` const &key) const
- `shared_config` `at_path` (`shared_origin` `origin`, `path` raw\_path) const
- virtual `shared_value` `new_copy` (`shared_origin` `origin`) const =0

- virtual [resolve\\_result](#)< shared\_value > **resolve\_substitutions** ([resolve\\_context](#) const &context, [resolve\\_source](#) const &source) const
- void **require\_not\_ignoring\_fallbacks** () const
- virtual bool **ignores\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const [unmergeable](#) > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const [unmergeable](#) > fallback) const
- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- virtual shared\_value **construct\_delayed\_merge** (shared\_origin [origin](#), std::vector< shared\_value > stack) const
- shared\_value [to\\_fallback\\_value](#) () const override

*Converts a config to its root object and a [config\\_value](#) to itself.*

## Static Protected Member Functions

- static void **indent** (std::string &result, int indent, [config\\_render\\_options](#) const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** ([config\\_value](#) const &other, std::function< bool(T const &)> checker)

## Friends

- class **token**
- class **value**
- class **default\_transformer**
- class **config**
- class **config\_object**
- class **simple\_config\_object**
- class **simple\_config\_list**
- class **config\_concatenation**
- class **resolve\_context**
- class **config\_delayed\_merge**
- class **config\_delayed\_merge\_object**
- resolve\_status **resolve\_status\_from\_values** (std::vector< shared\_value > const &v)

### 7.44.1 Detailed Description

An immutable value, following the [JSON](#) type schema.

Because this object is immutable, it is safe to use from multiple threads and there's no need for "defensive copies."

*Do not implement interface*

`ConfigValue`

; it should only be implemented by the config library. Arbitrary implementations will not work because the library internals assume a specific concrete implementation. Also, this interface is likely to grow new methods over time, so third-party implementations will break.

Definition at line 39 of file `config_value.hpp`.

## 7.44.2 Member Enumeration Documentation

### 7.44.2.1 type

```
enum hocon::config_value::type [strong]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file config\_value.hpp.

## 7.44.3 Member Function Documentation

### 7.44.3.1 at\_key()

```
shared_config hocon::config_value::at_key (  
    std::string const & key ) const
```

Places the value inside a [config](#) at the given key.

See also `config_value#at_path(string)`.

#### Parameters

<i>key</i>	key to store this value at.
------------	-----------------------------

#### Returns

**a**  
`config`

instance containing this value at the given key.

### 7.44.3.2 at\_path()

```
shared_config hocon::config_value::at_path (  
    std::string const & path_expression ) const
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

#### Parameters

<i>path</i>	path to store this value at.
-------------	------------------------------

#### Returns

a  
config  
instance containing this value at the given path.

#### 7.44.3.3 origin()

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

#### Returns

where the value came from

#### 7.44.3.4 relativized()

```
virtual shared_value hocon::config_value::relativized (
    std::string prefix ) const [inline], [virtual]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at foo.bar in the parent, and the included file as a substitution \${a.b.c}, the included substitution now needs to be \${foo.bar.a.b.c} because we resolve substitutions globally only after parsing everything.

#### Parameters

<i>prefix</i>	
---------------	--

#### Returns

value relativized to the given path or the same value if nothing to do

Reimplemented in [hocon::config\\_concatenation](#), and [hocon::simple\\_config\\_list](#).

Definition at line 181 of file config\_value.hpp.

**7.44.3.5 render() [1/2]**

```
virtual std::string hocon::config_value::render ( ) const [virtual]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to  
[render](#)([config\\_render\\_options](#)() )

.

**Returns**

the rendered value

**7.44.3.6 render() [2/2]**

```
virtual std::string hocon::config_value::render (
    config_render_options options ) const [virtual]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

**Parameters**

<i>options</i>	the rendering options
----------------	-----------------------

**Returns**

the rendered value

**7.44.3.7 to\_fallback\_value()**

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

#### 7.44.3.8 value\_type()

```
virtual type hocon::config_value::value_type ( ) const [pure virtual]
```

The type of the value; matches the JSON type schema.

##### Returns

value's type

Implemented in [hocon::simple\\_config\\_list](#), [hocon::config\\_string](#), [hocon::config\\_reference](#), [hocon::config\\_number](#), [hocon::config\\_null](#), [hocon::config\\_delayed\\_merge](#), [hocon::config\\_concatenation](#), [hocon::config\\_boolean](#), and [hocon::config\\_object](#).

#### 7.44.3.9 value\_type\_name()

```
char const* hocon::config_value::value_type_name ( ) const [inline]
```

The printable name of the value type.

##### Returns

value's type's name

Definition at line 92 of file [config\\_value.hpp](#).

#### 7.44.3.10 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

##### Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

##### Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

#### 7.44.3.11 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual]
```



Returns a.  
config\_value

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single [config\\_value](#).

#### Parameters

<i>origin</i>	the origin set on the returned value
---------------	--------------------------------------

#### Returns

the new [config\\_value](#) with the given origin

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

- hocon/config\_value.hpp

## 7.45 hocon::config\_value\_factory Class Reference

### Static Public Member Functions

- static shared\_value [from\\_any\\_ref](#) (unwrapped\_value [value](#), std::string origin\_description="")  
*Creates a ConfigValue from a plain value, which may be a bool, long, string, unordered\_map, vector or nullptr.*

### 7.45.1 Detailed Description

Definition at line 8 of file config\_value\_factory.hpp.

### 7.45.2 Member Function Documentation

#### 7.45.2.1 from\_any\_ref()

```
static shared_value hocon::config_value_factory::from_any_ref (
    unwrapped_value value,
    std::string origin_description = "" ) [static]
```

Creates a ConfigValue from a plain value, which may be a bool, long, string, unordered\_map, vector or nullptr.

An unordered\_map must be a unordered\_map from string to more values that can be supplied to [from\\_any\\_ref\(\)](#). An unordered\_map will become a ConfigObject and a vector will become a ConfigList.

In a unordered\_map passed to [from\\_any\\_ref\(\)](#), the map's keys are plain keys, not path expressions. So if your unordered\_map has a key "foo.bar" then you will get one object with a key called "foo.bar", rather than an object with a key "foo" containing another object with a key "bar".

The origin\_description will be used to set the origin() field on the ConfigValue. It should normally be the name of the file the values came from, or something short describing the value such as "default settings". The origin\_description is prefixed to error messages so users can tell where problematic values are coming from.

Supplying the result of ConfigValue.unwrapped() to this function is guaranteed to work and should give you back a ConfigValue that matches the one you unwrapped. The re-wrapped ConfigValue will lose some information that was present in the original such as its origin, but it will have matching values.

## Parameters

<i>unwrapped_value</i>	object object to convert to ConfigValue
<i>string</i>	origin_description name of origin file or brief description of what the value is

## Returns

shared\_value a new value

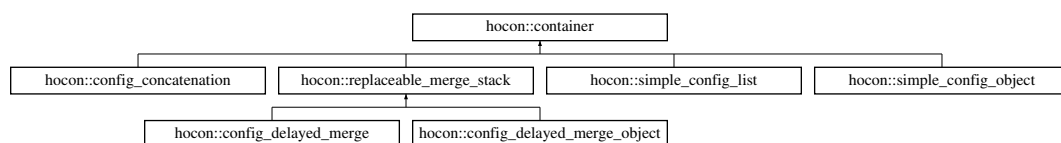
The documentation for this class was generated from the following file:

- hocon/config\_value\_factory.hpp

## 7.46 hocon::container Class Reference

An AbstractConfigValue which contains other values.

Inheritance diagram for hocon::container:



### Public Member Functions

- virtual shared\_value [replace\\_child](#) (shared\_value const &child, shared\_value replacement) const =0  
*Replace a child of this value.*
- virtual bool [has\\_descendant](#) (shared\_value const &descendant) const =0  
*Super-expensive full traversal to see if descendant is anywhere underneath this container.*

#### 7.46.1 Detailed Description

An AbstractConfigValue which contains other values.

Java has no way to express "this has to be an AbstractConfigValue also" other than making AbstractConfigValue an interface which would be aggravating. But we can say we are a ConfigValue.

Definition at line 12 of file container.hpp.

#### 7.46.2 Member Function Documentation

### 7.46.2.1 has\_descendant()

```
virtual bool hocon::container::has_descendant (
    shared_value const & descendant ) const [pure virtual]
```

Super-expensive full traversal to see if descendant is anywhere underneath this container.

Implemented in [hocon::simple\\_config\\_object](#), [hocon::simple\\_config\\_list](#), [hocon::config\\_delayed\\_merge\\_object](#), [hocon::config\\_delayed\\_merge](#), and [hocon::config\\_concatenation](#).

### 7.46.2.2 replace\_child()

```
virtual shared_value hocon::container::replace_child (
    shared_value const & child,
    shared_value replacement ) const [pure virtual]
```

Replace a child of this value.

CAUTION if replacement is null, delete the child, which may also delete the parent, or make the parent into a non-container.

Implemented in [hocon::simple\\_config\\_object](#), [hocon::simple\\_config\\_list](#), [hocon::config\\_delayed\\_merge\\_object](#), [hocon::config\\_delayed\\_merge](#), and [hocon::config\\_concatenation](#).

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

- internal/container.hpp

## 7.47 hocon::default\_transformer Class Reference

### Static Public Member Functions

- static shared\_value **transform** (shared\_value [value](#), [config\\_value::type](#) requested)

### 7.47.1 Detailed Description

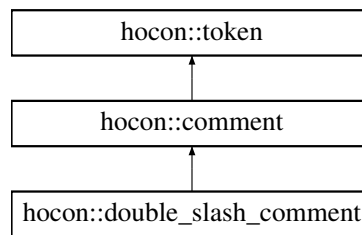
Definition at line 7 of file default\_transformer.hpp.

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

- internal/default\_transformer.hpp

## 7.48 hocon::double\_slash\_comment Class Reference

Inheritance diagram for hocon::double\_slash\_comment:



### Public Member Functions

- **double\_slash\_comment** (shared\_origin origin, std::string text)
- std::string **token\_text** () const override
- std::string **text** () const
- std::string **to\_string** () const override
- bool **operator==** (const [token](#) &other) const override
- virtual token\_type **get\_token\_type** () const
- virtual shared\_origin const & **origin** () const
- int **line\_number** () const

### 7.48.1 Detailed Description

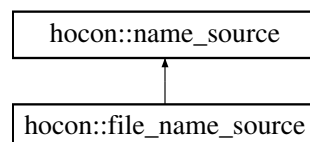
Definition at line 82 of file tokens.hpp.

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

- internal/tokens.hpp

## 7.49 hocon::file\_name\_source Class Reference

Inheritance diagram for hocon::file\_name\_source:



### Public Member Functions

- **file\_name\_source** (shared\_include\_context context)
- shared\_parseable **name\_to\_parseable** (std::string name, [config\\_parse\\_options](#) parse\_options) const override
- void **set\_context** (shared\_include\_context context)
- shared\_include\_context **get\_context** () const
- bool **context\_initialized** () const

### 7.49.1 Detailed Description

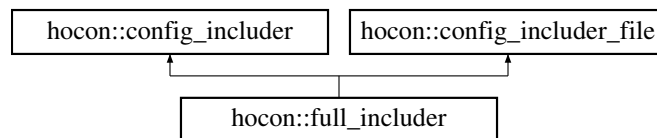
Definition at line 98 of file simple\_includer.hpp.

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

- internal/simple\_includer.hpp

## 7.50 hocon::full\_includer Class Reference

Inheritance diagram for hocon::full\_includer:



### Public Member Functions

- virtual shared\_includer [with\\_fallback](#) (shared\_includer fallback) const =0  
*Returns a new includer that falls back to the given includer.*
- virtual shared\_object [include](#) (shared\_include\_context context, std::string what) const =0  
*Parses another item to be included.*
- virtual shared\_object [include\\_file](#) (shared\_include\_context context, std::string what) const =0  
*Parses another item to be included.*

### 7.50.1 Detailed Description

Definition at line 9 of file full\_includer.hpp.

### 7.50.2 Member Function Documentation

#### 7.50.2.1 include()

```
virtual shared_object hocon::config_includer::include (
    shared_include_context context,
    std::string what ) const [pure virtual], [inherited]
```

Parses another item to be included.

The returned object typically would not have substitutions resolved. You can throw a [config\\_exception](#) here to abort parsing, or return an empty object, but may not return null.

This method is used for a "heuristic" include statement that does not specify file, or URL resource. If the include statement does specify, then the same class implementing [config\\_includer](#) must also implement [config\\_includer\\_file](#) or [config\\_includer\\_URL](#) as needed, or a default includer will be used.

## Parameters

<i>context</i>	some info about the include context
<i>what</i>	the include statement's argument

## Returns

a non-null [config\\_object](#)

Implemented in [hocon::simple\\_includer](#).

### 7.50.2.2 include\_file()

```
virtual shared_object hocon::config_includer_file::include_file (
    shared_include_context context,
    std::string what ) const [pure virtual], [inherited]
```

Parses another item to be included.

The returned object typically would not have substitutions resolved. You can throw a [config\\_exception](#) here to abort parsing, or return an empty object, but may not return null.

## Parameters

<i>context</i>	some info about the include context
<i>what</i>	the include statement's argument (a file path)

## Returns

a non-null [config\\_object](#)

Implemented in [hocon::simple\\_includer](#).

### 7.50.2.3 with\_fallback()

```
virtual shared_includer hocon::config_includer::with_fallback (
    shared_includer fallback ) const [pure virtual], [inherited]
```

Returns a new includer that falls back to the given includer.

This is how you can obtain the default includer; it will be provided as a fallback. It's up to your includer to chain to it if you want to. You might want to merge any files found by the fallback includer with any objects you load yourself.

It's important to handle the case where you already have the fallback with a "return this", i.e. this method should not create a new object if the fallback is the same one you already have. The same fallback may be added repeatedly.

## Parameters

<i>fallback</i>	the previous includer for chaining
-----------------	------------------------------------

## Returns

a new includer

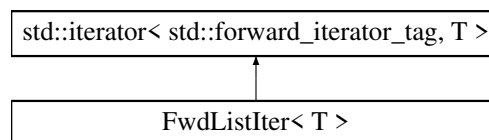
Implemented in [hocon::simple\\_includer](#).

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

- [internal/full\\_includer.hpp](#)

## 7.51 FwdListIter< T > Class Template Reference

Inheritance diagram for FwdListIter< T >:



### Public Member Functions

- **FwdListIter** ([List](#)< T > const &lst)
- T **operator\*** () const
- [FwdListIter](#) & **operator++** ()
- bool **operator==** ([FwdListIter](#)< T > const &other)
- bool **operator!=** ([FwdListIter](#)< T > const &other)

### 7.51.1 Detailed Description

```
template<class T>
class FwdListIter< T >
```

Definition at line 117 of file [functional\\_list.hpp](#).

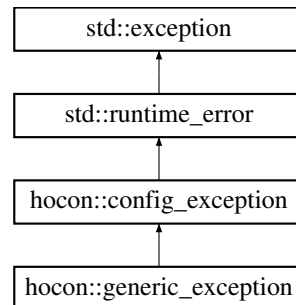
The documentation for this class was generated from the following file:

- [hocon/functional\\_list.hpp](#)

## 7.52 hocon::generic\_exception Struct Reference

Exception that doesn't fall into any other category.

Inheritance diagram for hocon::generic\_exception:



### Public Member Functions

- **generic\_exception** (std::string const &message)

#### 7.52.1 Detailed Description

Exception that doesn't fall into any other category.

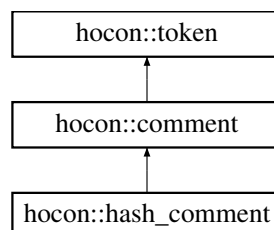
Definition at line 181 of file config\_exception.hpp.

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

- hocon/config\_exception.hpp

## 7.53 hocon::hash\_comment Class Reference

Inheritance diagram for hocon::hash\_comment:



### Public Member Functions

- **hash\_comment** (shared\_origin origin, std::string text)
- std::string **token\_text** () const override
- std::string **text** () const
- std::string **to\_string** () const override
- bool **operator==** (const token &other) const override
- virtual token\_type **get\_token\_type** () const
- virtual shared\_origin const & **origin** () const
- int **line\_number** () const



### 7.53.1 Detailed Description

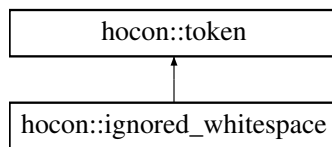
Definition at line 89 of file tokens.hpp.

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

- internal/tokens.hpp

## 7.54 hocon::ignored\_whitespace Class Reference

Inheritance diagram for hocon::ignored\_whitespace:



### Public Member Functions

- **ignored\_whitespace** (shared\_origin origin, std::string whitespace)
- std::string **to\_string** () const override
- bool **operator==** (const [token](#) &other) const override
- virtual token\_type **get\_token\_type** () const
- virtual std::string **token\_text** () const
- virtual shared\_origin const & **origin** () const
- int **line\_number** () const

### 7.54.1 Detailed Description

Definition at line 42 of file tokens.hpp.

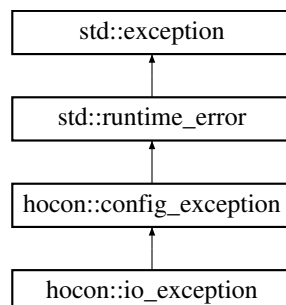
The documentation for this class was generated from the following file:

- internal/tokens.hpp

## 7.55 hocon::io\_exception Struct Reference

Exception indicating that there was an IO error.

Inheritance diagram for hocon::io\_exception:



## Public Member Functions

- **io\_exception** ([config\\_origin](#) const &origin, std::string const &message)

### 7.55.1 Detailed Description

Exception indicating that there was an IO error.

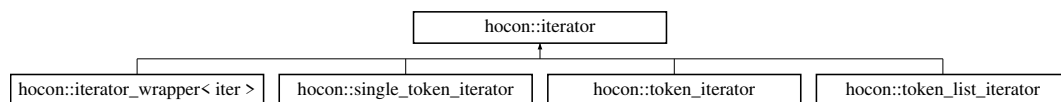
Definition at line 93 of file `config_exception.hpp`.

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

- `hocon/config_exception.hpp`

## 7.56 hocon::iterator Class Reference

Inheritance diagram for `hocon::iterator`:



## Public Member Functions

- virtual bool **has\_next** ()=0
- virtual shared\_token **next** ()=0

### 7.56.1 Detailed Description

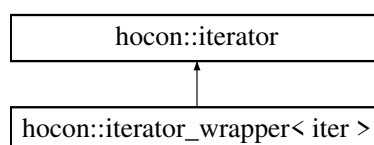
Definition at line 24 of file `tokenizer.hpp`.

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

- `internal/tokenizer.hpp`

## 7.57 hocon::iterator\_wrapper< iter > Class Template Reference

Inheritance diagram for `hocon::iterator_wrapper< iter >`:



## Public Member Functions

- **iterator\_wrapper** (iter begin, iter end)
- bool **has\_next** () override
- shared\_token **next** () override

### 7.57.1 Detailed Description

```
template<typename iter>
class hocon::iterator_wrapper< iter >
```

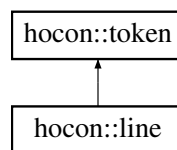
Definition at line 31 of file tokenizer.hpp.

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

- internal/tokenizer.hpp

## 7.58 hocon::line Class Reference

Inheritance diagram for hocon::line:



## Public Member Functions

- **line** (shared\_origin origin)
- std::string **to\_string** () const override
- bool **operator==** (**token** const &other) const override
- virtual token\_type **get\_token\_type** () const
- virtual std::string **token\_text** () const
- virtual shared\_origin const & **origin** () const
- int **line\_number** () const

### 7.58.1 Detailed Description

Definition at line 24 of file tokens.hpp.

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

- internal/tokens.hpp

## 7.59 List< T > Class Template Reference

### Public Member Functions

- **List** (T v, [List](#) const &tail)
- **List** (T v)
- bool **isEmpty** () const
- T **front** () const
- [List](#) **popped\_front** () const
- [List](#) **pushed\_front** (T v) const
- [List](#) **take** (int n)
- [List](#) **insertedAt** (int i, T v) const
- [List](#) **removed** (T v) const
- [List](#) **removed1** (T v) const
- bool **member** (T v) const
- template<class F >  
void **forEach** (F f) const
- int **headCount** () const

### Friends

- class **FwdListIter**< T >

### 7.59.1 Detailed Description

```
template<class T>
class List< T >
```

Definition at line 13 of file functional\_list.hpp.

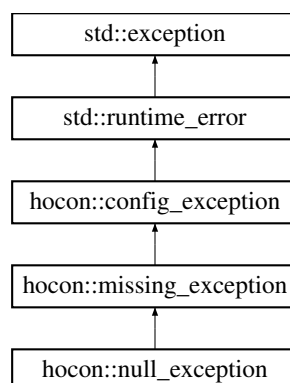
The documentation for this class was generated from the following file:

- hocon/functional\_list.hpp

## 7.60 hocon::missing\_exception Struct Reference

Exception indicates that the setting was never set to anything, not even null.

Inheritance diagram for hocon::missing\_exception:



## Public Member Functions

- **missing\_exception** (std::string const &path)
- **config\_exception** (config\_origin const &origin, std::string const &message)
- **config\_exception** (std::string const &message)
- **config\_exception** (std::string const &message, std::exception const &e)

### 7.60.1 Detailed Description

Exception indicates that the setting was never set to anything, not even null.

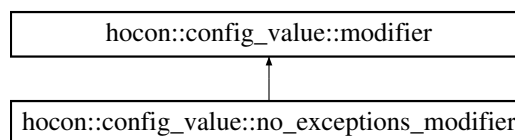
Definition at line 38 of file config\_exception.hpp.

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

- hocon/config\_exception.hpp

## 7.61 hocon::config\_value::modifier Class Reference

Inheritance diagram for hocon::config\_value::modifier:



## Public Member Functions

- virtual shared\_value **modify\_child\_may\_throw** (std::string const &key\_or\_null, shared\_value v)=0

### 7.61.1 Detailed Description

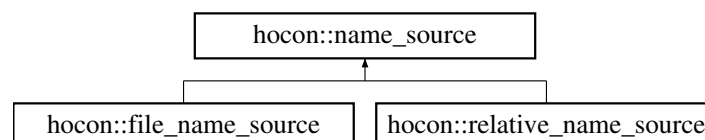
Definition at line 212 of file config\_value.hpp.

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

- hocon/config\_value.hpp

## 7.62 hocon::name\_source Class Reference

Inheritance diagram for hocon::name\_source:



## Public Member Functions

- **name\_source** (shared\_include\_context context)
- virtual shared\_parseable **name\_to\_parseable** (std::string name, [config\\_parse\\_options](#) parse\_options) const =0
- void **set\_context** (shared\_include\_context context)
- shared\_include\_context **get\_context** () const
- bool **context\_initialized** () const

### 7.62.1 Detailed Description

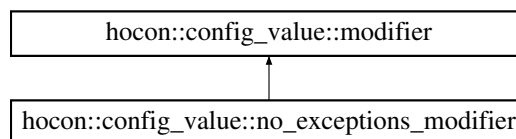
Definition at line 57 of file simple\_includer.hpp.

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

- internal/simple\_includer.hpp

## 7.63 hocon::config\_value::no\_exceptions\_modifier Class Reference

Inheritance diagram for hocon::config\_value::no\_exceptions\_modifier:



## Public Member Functions

- **no\_exceptions\_modifier** (std::string prefix)
- shared\_value **modify\_child\_may\_throw** (std::string const &key\_or\_null, shared\_value v) override
- shared\_value **modify\_child** (std::string const &key, shared\_value v) const

### 7.63.1 Detailed Description

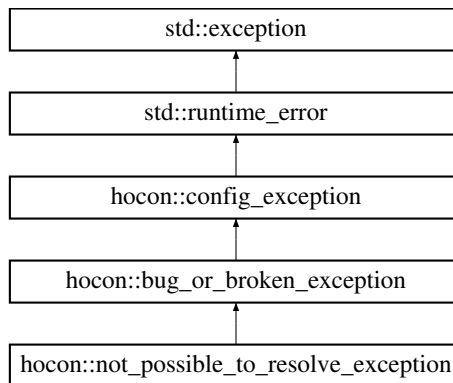
Definition at line 217 of file config\_value.hpp.

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

- hocon/config\_value.hpp

## 7.64 hocon::not\_possible\_to\_resolve\_exception Struct Reference

Inheritance diagram for hocon::not\_possible\_to\_resolve\_exception:



### Public Member Functions

- **not\_possible\_to\_resolve\_exception** (std::string const &message)

#### 7.64.1 Detailed Description

Definition at line 127 of file `config_exception.hpp`.

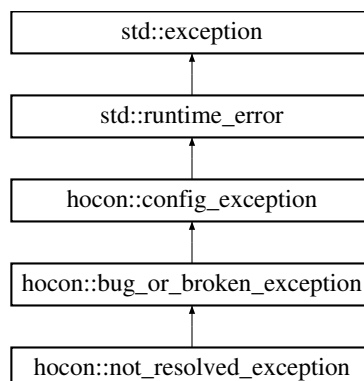
The documentation for this struct was generated from the following file:

- `hocon/config_exception.hpp`

## 7.65 hocon::not\_resolved\_exception Struct Reference

Exception indicating that you tried to use a function that requires substitutions to be resolved, but substitutions have not been resolved (that is, `config#resolve` was not called).

Inheritance diagram for hocon::not\_resolved\_exception:



## Public Member Functions

- **not\_resolved\_exception** (std::string const &message)

### 7.65.1 Detailed Description

Exception indicating that you tried to use a function that requires substitutions to be resolved, but substitutions have not been resolved (that is, [config#resolve](#) was not called).

This is always a bug in either application code or the library; it's wrong to write a handler for this exception because you should be able to fix the code to avoid it by adding calls to [config#resolve](#).

Definition at line 123 of file `config_exception.hpp`.

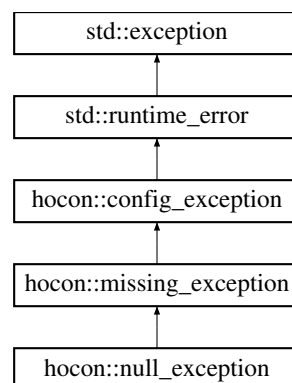
The documentation for this struct was generated from the following file:

- `hocon/config_exception.hpp`

## 7.66 hocon::null\_exception Struct Reference

Exception indicates that the setting was treated as missing because it was set to null.

Inheritance diagram for `hocon::null_exception`:



## Public Member Functions

- **null\_exception** ([config\\_origin](#) const &origin, std::string const &[path](#), std::string const &expected="")
- **config\_exception** ([config\\_origin](#) const &origin, std::string const &message)
- **config\_exception** (std::string const &message)
- **config\_exception** (std::string const &message, std::exception const &e)

### 7.66.1 Detailed Description

Exception indicates that the setting was treated as missing because it was set to null.

Definition at line 48 of file `config_exception.hpp`.

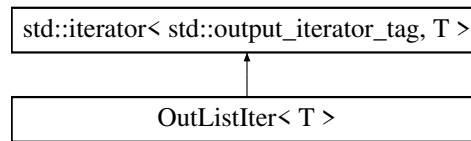
The documentation for this struct was generated from the following file:

- `hocon/config_exception.hpp`



## 7.67 OutListIter< T > Class Template Reference

Inheritance diagram for OutListIter< T >:



### Public Member Functions

- T & **operator\*** ()
- [OutListIter](#) & **operator++** ()
- [List](#)< T > **getList** () const

#### 7.67.1 Detailed Description

```
template<class T>
class OutListIter< T >
```

Definition at line 142 of file functional\_list.hpp.

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

- hocon/functional\_list.hpp

## 7.68 hocon::config\_document\_parser::parse\_context Class Reference

### Public Member Functions

- **parse\_context** (config\_syntax flavor, shared\_origin origin, [token\\_iterator](#) tokens)
- std::shared\_ptr< [config\\_node\\_root](#) > **parse** ()
- shared\_node\_value [parse\\_single\\_value](#) ()

*Parse a given input stream into a single value node.*

#### 7.68.1 Detailed Description

Definition at line 22 of file config\_document\_parser.hpp.

#### 7.68.2 Member Function Documentation

### 7.68.2.1 `parse_single_value()`

```
shared_node_value hocon::config_document_parser::parse_context::parse_single_value ( )
```

Parse a given input stream into a single value node.

Used when doing a replace inside a ConfigDocument.

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

- `internal/config_document_parser.hpp`

## 7.69 `hocon::config_parser::parse_context` Class Reference

### Public Member Functions

- **`parse_context`** (`config_syntax` flavor, `shared_origin` origin, `std::shared_ptr< const config_node_root >` document, `std::shared_ptr< const full_includer >` includer, `shared_include_context` include\_context)
- `shared_value` **`parse`** ()

### Public Attributes

- `int` **`array_count`**

### 7.69.1 Detailed Description

Definition at line 27 of file `config_parser.hpp`.

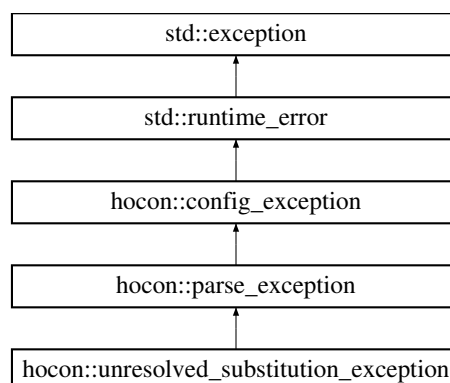
The documentation for this class was generated from the following file:

- `internal/config_parser.hpp`

## 7.70 `hocon::parse_exception` Struct Reference

Exception indicating that there was a parse error.

Inheritance diagram for `hocon::parse_exception`:



## Public Member Functions

- **parse\_exception** ([config\\_origin](#) const &origin, std::string const &message)

### 7.70.1 Detailed Description

Exception indicating that there was a parse error.

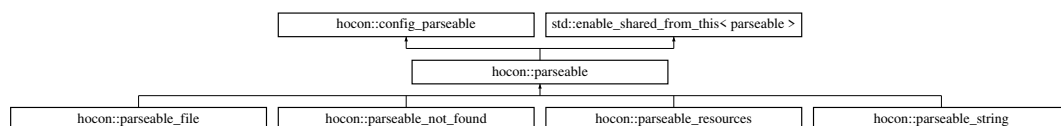
Definition at line 101 of file `config_exception.hpp`.

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

- `hocon/config_exception.hpp`

## 7.71 hocon::parseable Class Reference

Inheritance diagram for `hocon::parseable`:



## Public Member Functions

- void **post\_construct** ([config\\_parse\\_options](#) const &base\_options)
- std::shared\_ptr< [config\\_document](#) > **parse\_config\_document** ()
- shared\_object **parse** ([config\\_parse\\_options](#) const &[options](#)) const override  
*Parse whatever it is.*
- shared\_object **parse** () const
- shared\_value **parse\_value** () const
- [config\\_parse\\_options](#) const & [options](#) () const override  
*Get the initial options, which can be modified then passed to [parse\(\)](#).*
- std::shared\_ptr< const [config\\_origin](#) > **origin** () const override  
*Returns a [config\\_origin](#) describing the origin of the parseable item.*
- virtual std::unique\_ptr< std::istream > **reader** ([config\\_parse\\_options](#) const &[options](#)) const
- virtual std::unique\_ptr< std::istream > **reader** () const =0
- virtual shared\_origin **create\_origin** () const =0
- virtual config\_syntax **guess\_syntax** () const
- virtual config\_syntax **content\_type** () const
- virtual std::shared\_ptr< [config\\_parseable](#) > **relative\_to** (std::string file\_name) const
- std::string **to\_string** () const
- std::string **get\_cur\_dir** () const
- void **set\_cur\_dir** (std::string dir) const
- void **separate\_filepath** (const std::string &[path](#), std::string \*file\_dir, std::string \*file\_name) const
- **parseable** ([parseable](#) const &)=delete
- [parseable](#) & **operator=** ([parseable](#) const &)=delete

## Static Public Member Functions

- static std::shared\_ptr< [parseable](#) > **new\_file** (std::string input\_file\_path, [config\\_parse\\_options options](#))
- static std::shared\_ptr< [parseable](#) > **new\_string** (std::string s, [config\\_parse\\_options options](#))
- static std::shared\_ptr< [parseable](#) > **new\_not\_found** (std::string what\_not\_found, std::string message, [config\\_parse\\_options options](#))
- static config\_syntax **syntax\_from\_extension** (std::string name)

### 7.71.1 Detailed Description

Definition at line 13 of file `parseable.hpp`.

### 7.71.2 Member Function Documentation

#### 7.71.2.1 options()

```
config\_parse\_options const& hocon::parseable::options ( ) const [override], [virtual]
```

Get the initial options, which can be modified then passed to [parse\(\)](#).

These options will have the right description, includer, and other parameters already set up.

#### Returns

the initial options

Implements [hocon::config\\_parseable](#).

#### 7.71.2.2 origin()

```
std::shared_ptr<const config\_origin> hocon::parseable::origin ( ) const [override], [virtual]
```

Returns a [config\\_origin](#) describing the origin of the parseable item.

Implements [hocon::config\\_parseable](#).

#### 7.71.2.3 parse()

```
shared_object hocon::parseable::parse (
    config\_parse\_options const & options ) const [override], [virtual]
```

Parse whatever it is.

The options should come from [config\\_parseable#options\(\)](#) but you could tweak them if you like.

## Parameters

<i>options</i>	parse options, should be based on the ones from <a href="#">config_parseable#options()</a>
----------------	--

## Returns

the parsed object

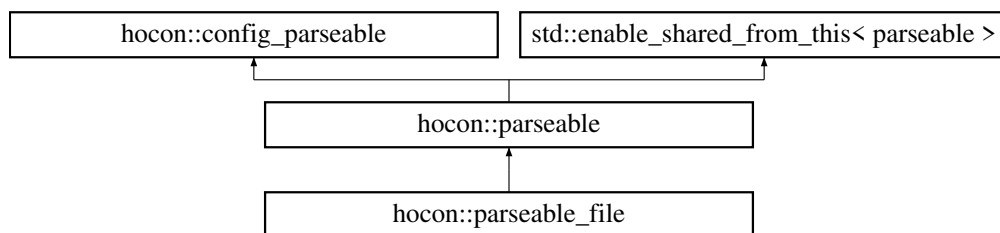
Implements [hocon::config\\_parseable](#).

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

- internal/parseable.hpp

## 7.72 hocon::parseable\_file Class Reference

Inheritance diagram for hocon::parseable\_file:



### Public Member Functions

- **parseable\_file** (std::string input\_file\_path, [config\\_parse\\_options options](#))
- std::unique\_ptr< std::istream > **reader** () const override
- shared\_origin **create\_origin** () const override
- config\_syntax **guess\_syntax** () const override
- void **post\_construct** ([config\\_parse\\_options](#) const &base\_options)
- std::shared\_ptr< [config\\_document](#) > **parse\_config\_document** ()
- shared\_object **parse** ([config\\_parse\\_options](#) const &[options](#)) const override
- *Parse whatever it is.*
- shared\_object **parse** () const
- shared\_value **parse\_value** () const
- [config\\_parse\\_options](#) const & **options** () const override
- *Get the initial options, which can be modified then passed to [parse\(\)](#).*
- std::shared\_ptr< const [config\\_origin](#) > **origin** () const override
- *Returns a [config\\_origin](#) describing the origin of the parseable item.*
- virtual std::unique\_ptr< std::istream > **reader** ([config\\_parse\\_options](#) const &[options](#)) const
- virtual config\_syntax **content\_type** () const
- virtual std::shared\_ptr< [config\\_parseable](#) > **relative\_to** (std::string file\_name) const
- std::string **to\_string** () const
- std::string **get\_cur\_dir** () const
- void **set\_cur\_dir** (std::string dir) const
- void **separate\_filepath** (const std::string &[path](#), std::string \*file\_dir, std::string \*file\_name) const

## Static Public Member Functions

- static std::shared\_ptr< [parseable](#) > **new\_file** (std::string input\_file\_path, [config\\_parse\\_options](#) options)
- static std::shared\_ptr< [parseable](#) > **new\_string** (std::string s, [config\\_parse\\_options](#) options)
- static std::shared\_ptr< [parseable](#) > **new\_not\_found** (std::string what\_not\_found, std::string message, [config\\_parse\\_options](#) options)
- static config\_syntax **syntax\_from\_extension** (std::string name)

### 7.72.1 Detailed Description

Definition at line 79 of file [parseable.hpp](#).

### 7.72.2 Member Function Documentation

#### 7.72.2.1 options()

```
config\_parse\_options const& hocon::parseable::options ( ) const [override], [virtual], [inherited]
```

Get the initial options, which can be modified then passed to [parse\(\)](#).

These options will have the right description, includer, and other parameters already set up.

#### Returns

the initial options

Implements [hocon::config\\_parseable](#).

#### 7.72.2.2 origin()

```
std::shared_ptr<const config\_origin> hocon::parseable::origin ( ) const [override], [virtual], [inherited]
```

Returns a [config\\_origin](#) describing the origin of the parseable item.

Implements [hocon::config\\_parseable](#).

#### 7.72.2.3 parse()

```
shared_object hocon::parseable::parse (
    config\_parse\_options const & options ) const [override], [virtual], [inherited]
```

Parse whatever it is.

The options should come from [config\\_parseable#options\(\)](#) but you could tweak them if you like.

## Parameters

<i>options</i>	parse options, should be based on the ones from <a href="#">config_parseable#options()</a>
----------------	--

## Returns

the parsed object

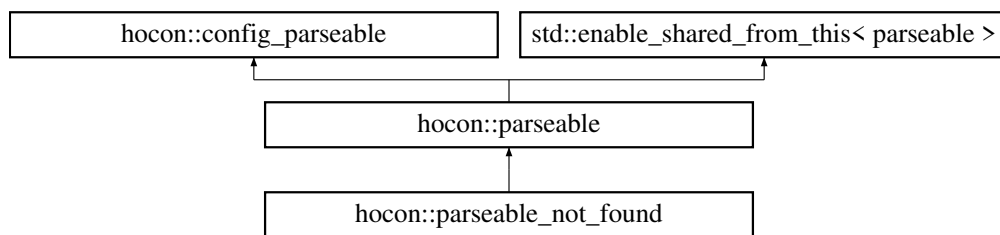
Implements [hocon::config\\_parseable](#).

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

- internal/parseable.hpp

## 7.73 hocon::parseable\_not\_found Class Reference

Inheritance diagram for hocon::parseable\_not\_found:



### Public Member Functions

- **parseable\_not\_found** (std::string what, std::string message, [config\\_parse\\_options options](#))
- std::unique\_ptr< std::istream > **reader** () const override
- shared\_origin **create\_origin** () const override
- void **post\_construct** ([config\\_parse\\_options](#) const &base\_options)
- std::shared\_ptr< [config\\_document](#) > **parse\_config\_document** ()
- shared\_object **parse** ([config\\_parse\\_options](#) const &[options](#)) const override
- *Parse whatever it is.*
- shared\_object **parse** () const
- shared\_value **parse\_value** () const
- [config\\_parse\\_options](#) const & **options** () const override
- *Get the initial options, which can be modified then passed to [parse\(\)](#).*
- std::shared\_ptr< const [config\\_origin](#) > **origin** () const override
- *Returns a [config\\_origin](#) describing the origin of the parseable item.*
- virtual std::unique\_ptr< std::istream > **reader** ([config\\_parse\\_options](#) const &[options](#)) const
- virtual config\_syntax **guess\_syntax** () const
- virtual config\_syntax **content\_type** () const
- virtual std::shared\_ptr< [config\\_parseable](#) > **relative\_to** (std::string file\_name) const
- std::string **to\_string** () const
- std::string **get\_cur\_dir** () const
- void **set\_cur\_dir** (std::string dir) const
- void **separate\_filepath** (const std::string &[path](#), std::string \*file\_dir, std::string \*file\_name) const

## Static Public Member Functions

- static std::shared\_ptr< [parseable](#) > **new\_file** (std::string input\_file\_path, [config\\_parse\\_options](#) options)
- static std::shared\_ptr< [parseable](#) > **new\_string** (std::string s, [config\\_parse\\_options](#) options)
- static std::shared\_ptr< [parseable](#) > **new\_not\_found** (std::string what\_not\_found, std::string message, [config\\_parse\\_options](#) options)
- static config\_syntax **syntax\_from\_extension** (std::string name)

### 7.73.1 Detailed Description

Definition at line 120 of file [parseable.hpp](#).

### 7.73.2 Member Function Documentation

#### 7.73.2.1 options()

```
config\_parse\_options const& hocon::parseable::options ( ) const [override], [virtual], [inherited]
```

Get the initial options, which can be modified then passed to [parse\(\)](#).

These options will have the right description, includer, and other parameters already set up.

Returns

the initial options

Implements [hocon::config\\_parseable](#).

#### 7.73.2.2 origin()

```
std::shared_ptr<const config\_origin> hocon::parseable::origin ( ) const [override], [virtual], [inherited]
```

Returns a [config\\_origin](#) describing the origin of the parseable item.

Implements [hocon::config\\_parseable](#).

#### 7.73.2.3 parse()

```
shared_object hocon::parseable::parse (
    config\_parse\_options const & options ) const [override], [virtual], [inherited]
```

Parse whatever it is.

The options should come from [config\\_parseable#options\(\)](#) but you could tweak them if you like.



## Parameters

<i>options</i>	parse options, should be based on the ones from <a href="#">config_parseable#options()</a>
----------------	--

## Returns

the parsed object

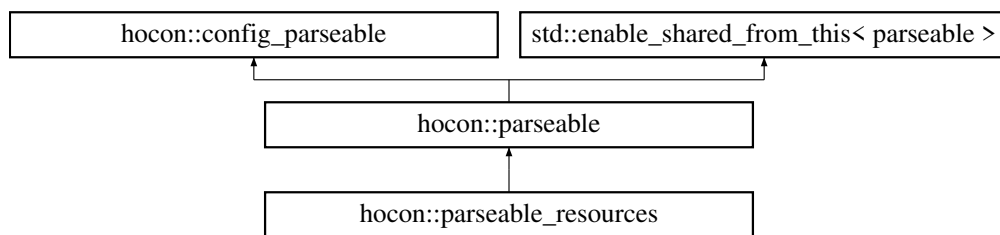
Implements [hocon::config\\_parseable](#).

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

- internal/parseable.hpp

## 7.74 hocon::parseable\_resources Class Reference

Inheritance diagram for hocon::parseable\_resources:



### Public Member Functions

- **parseable\_resources** (std::string resource, [config\\_parse\\_options options](#))
- std::unique\_ptr< std::istream > **reader** () const override
- shared\_origin **create\_origin** () const override
- void **post\_construct** ([config\\_parse\\_options](#) const &base\_options)
- std::shared\_ptr< [config\\_document](#) > **parse\_config\_document** ()
- shared\_object **parse** ([config\\_parse\\_options](#) const &[options](#)) const override
- *Parse whatever it is.*
- shared\_object **parse** () const
- shared\_value **parse\_value** () const
- [config\\_parse\\_options](#) const & **options** () const override
- *Get the initial options, which can be modified then passed to [parse\(\)](#).*
- std::shared\_ptr< const [config\\_origin](#) > **origin** () const override
- *Returns a [config\\_origin](#) describing the origin of the parseable item.*
- virtual std::unique\_ptr< std::istream > **reader** ([config\\_parse\\_options](#) const &[options](#)) const
- virtual config\_syntax **guess\_syntax** () const
- virtual config\_syntax **content\_type** () const
- virtual std::shared\_ptr< [config\\_parseable](#) > **relative\_to** (std::string file\_name) const
- std::string **to\_string** () const
- std::string **get\_cur\_dir** () const
- void **set\_cur\_dir** (std::string dir) const
- void **separate\_filepath** (const std::string &[path](#), std::string \*file\_dir, std::string \*file\_name) const

## Static Public Member Functions

- static std::shared\_ptr< [parseable](#) > **new\_file** (std::string input\_file\_path, [config\\_parse\\_options](#) options)
- static std::shared\_ptr< [parseable](#) > **new\_string** (std::string s, [config\\_parse\\_options](#) options)
- static std::shared\_ptr< [parseable](#) > **new\_not\_found** (std::string what\_not\_found, std::string message, [config\\_parse\\_options](#) options)
- static [config\\_syntax](#) **syntax\_from\_extension** (std::string name)

### 7.74.1 Detailed Description

Definition at line 107 of file [parseable.hpp](#).

### 7.74.2 Member Function Documentation

#### 7.74.2.1 options()

```
config\_parse\_options const& hocon::parseable::options ( ) const [override], [virtual], [inherited]
```

Get the initial options, which can be modified then passed to [parse\(\)](#).

These options will have the right description, includer, and other parameters already set up.

#### Returns

the initial options

Implements [hocon::config\\_parseable](#).

#### 7.74.2.2 origin()

```
std::shared_ptr<const config\_origin> hocon::parseable::origin ( ) const [override], [virtual], [inherited]
```

Returns a [config\\_origin](#) describing the origin of the parseable item.

Implements [hocon::config\\_parseable](#).

#### 7.74.2.3 parse()

```
shared_object hocon::parseable::parse (
    config\_parse\_options const & options ) const [override], [virtual], [inherited]
```

Parse whatever it is.

The options should come from [config\\_parseable#options\(\)](#) but you could tweak them if you like.

## Parameters

<i>options</i>	parse options, should be based on the ones from <a href="#">config_parseable#options()</a>
----------------	--

## Returns

the parsed object

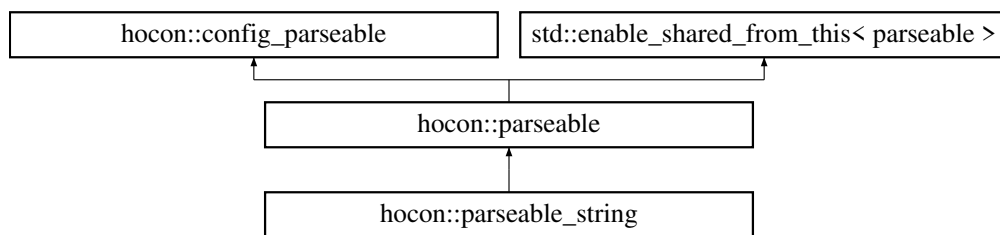
Implements [hocon::config\\_parseable](#).

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

- internal/parseable.hpp

## 7.75 hocon::parseable\_string Class Reference

Inheritance diagram for hocon::parseable\_string:



### Public Member Functions

- **parseable\_string** (std::string s, [config\\_parse\\_options](#) options)
- std::unique\_ptr< std::istream > **reader** () const override
- shared\_origin **create\_origin** () const override
- void **post\_construct** ([config\\_parse\\_options](#) const &base\_options)
- std::shared\_ptr< [config\\_document](#) > **parse\_config\_document** ()
- shared\_object **parse** ([config\\_parse\\_options](#) const &options) const override
 

*Parse whatever it is.*
- shared\_object **parse** () const
- shared\_value **parse\_value** () const
- [config\\_parse\\_options](#) const & **options** () const override
 

*Get the initial options, which can be modified then passed to [parse\(\)](#).*
- std::shared\_ptr< const [config\\_origin](#) > **origin** () const override
 

*Returns a [config\\_origin](#) describing the origin of the parseable item.*
- virtual std::unique\_ptr< std::istream > **reader** ([config\\_parse\\_options](#) const &options) const
- virtual config\_syntax **guess\_syntax** () const
- virtual config\_syntax **content\_type** () const
- virtual std::shared\_ptr< [config\\_parseable](#) > **relative\_to** (std::string file\_name) const
- std::string **to\_string** () const
- std::string **get\_cur\_dir** () const
- void **set\_cur\_dir** (std::string dir) const
- void **separate\_filepath** (const std::string &path, std::string \*file\_dir, std::string \*file\_name) const

## Static Public Member Functions

- static std::shared\_ptr< [parseable](#) > **new\_file** (std::string input\_file\_path, [config\\_parse\\_options](#) options)
- static std::shared\_ptr< [parseable](#) > **new\_string** (std::string s, [config\\_parse\\_options](#) options)
- static std::shared\_ptr< [parseable](#) > **new\_not\_found** (std::string what\_not\_found, std::string message, [config\\_parse\\_options](#) options)
- static config\_syntax **syntax\_from\_extension** (std::string name)

### 7.75.1 Detailed Description

Definition at line 90 of file [parseable.hpp](#).

### 7.75.2 Member Function Documentation

#### 7.75.2.1 options()

```
config\_parse\_options const& hocon::parseable::options ( ) const [override], [virtual], [inherited]
```

Get the initial options, which can be modified then passed to [parse\(\)](#).

These options will have the right description, includer, and other parameters already set up.

#### Returns

the initial options

Implements [hocon::config\\_parseable](#).

#### 7.75.2.2 origin()

```
std::shared_ptr<const config\_origin> hocon::parseable::origin ( ) const [override], [virtual], [inherited]
```

Returns a [config\\_origin](#) describing the origin of the parseable item.

Implements [hocon::config\\_parseable](#).

#### 7.75.2.3 parse()

```
shared_object hocon::parseable::parse (
    config\_parse\_options const & options ) const [override], [virtual], [inherited]
```

Parse whatever it is.

The options should come from [config\\_parseable#options\(\)](#) but you could tweak them if you like.

## Parameters

<i>options</i>	parse options, should be based on the ones from <a href="#">config_parseable#options()</a>
----------------	--

## Returns

the parsed object

Implements [hocon::config\\_parseable](#).

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

- internal/parseable.hpp

## 7.76 hocon::path Class Reference

### Public Member Functions

- **path** (std::string first, [path](#) const &remainder)
- **path** (std::vector< std::string > elements)
- **path** (std::vector< [path](#) > paths\_to\_concat)
- shared\_string **first** () const
- [path](#) **remainder** () const  
*Returns the path minus the first element, or null if no more elements.*
- [path](#) **parent** () const  
*Returns the path minus the last element or null if we have just one element.*
- bool **has\_remainder** () const
- bool **empty** () const
- shared\_string **last** () const
- [path](#) **prepend** ([path](#) prefix)
- int **length** () const
- [path](#) **sub\_path** (int remove\_from\_front)
- [path](#) **sub\_path** (int start\_index, int end\_index)
- bool **starts\_with** ([path](#) other) const
- bool **operator==** ([path](#) const &other) const
- bool **operator!=** ([path](#) const &other) const
- void **append\_to\_string** (std::string &base) const
- std::string **to\_string** () const  
*For debugging.*
- std::string **render** () const  
*Human-readable error-message-oriented string representation of path.*

### Static Public Member Functions

- static bool [has\\_funky\\_chars](#) (std::string const &s)  
*Signals whether quotes and other noise need to be removed/ignored.*
- static [path](#) **new\_key** (std::string key)
- static [path](#) **new\_path** (std::string [path](#))

### 7.76.1 Detailed Description

Definition at line 13 of file path.hpp.

### 7.76.2 Member Function Documentation

#### 7.76.2.1 has\_funky\_chars()

```
static bool hocon::path::has_funky_chars (
    std::string const & s ) [static]
```

Signals whether quotes and other noise need to be removed/ignored.

#### 7.76.2.2 parent()

```
path hocon::path::parent ( ) const
```

Returns the path minus the last element or null if we have just one element.

#### 7.76.2.3 remainder()

```
path hocon::path::remainder ( ) const
```

Returns the path minus the first element, or null if no more elements.

#### 7.76.2.4 render()

```
std::string hocon::path::render ( ) const
```

Human-readable error-message-oriented string representation of path.

#### 7.76.2.5 to\_string()

```
std::string hocon::path::to_string ( ) const
```

For debugging.

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

- hocon/path.hpp

## 7.77 hocon::path\_builder Class Reference

### Public Member Functions

- void **append\_key** (std::string key)
- void **append\_path** (path path\_to\_append)
- path **result** ()

*Returns null if keys is empty.*

#### 7.77.1 Detailed Description

Definition at line 8 of file path\_builder.hpp.

#### 7.77.2 Member Function Documentation

##### 7.77.2.1 result()

```
path hocon::path_builder::result ( )
```

Returns null if keys is empty.

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

- internal/path\_builder.hpp

## 7.78 hocon::path\_parser Class Reference

### Static Public Member Functions

- static config\_node\_path **parse\_path\_node** (std::string const &path\_string, config\_syntax flavor=config\_syntax::CONF)
- static path **parse\_path** (std::string const &path\_string)
- static path **parse\_path\_expression** (iterator &expression, shared\_origin origin, std::string const &original\_text="", token\_list \*path\_tokens=nullptr, config\_syntax flavor=config\_syntax::CONF)
- static config\_node\_path **parse\_path\_node\_expression** (iterator &expression, shared\_origin origin, std::string const &original\_text="", config\_syntax flavor=config\_syntax::CONF)

#### 7.78.1 Detailed Description

Definition at line 14 of file path\_parser.hpp.

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

- internal/path\_parser.hpp

## 7.79 hocon::problem Class Reference

Inheritance diagram for hocon::problem:



### Public Member Functions

- **problem** (shared\_origin origin, std::string what, std::string message, bool suggest\_quotes)
- std::string **what** () const
- std::string **message** () const
- bool **suggest\_quotes** () const
- std::string **to\_string** () const override
- bool **operator==** (const [token](#) &other) const override
- virtual token\_type **get\_token\_type** () const
- virtual std::string **token\_text** () const
- virtual shared\_origin const & **origin** () const
- int **line\_number** () const

### 7.79.1 Detailed Description

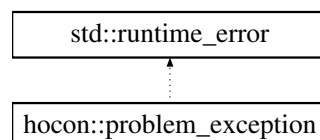
Definition at line 51 of file tokens.hpp.

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

- internal/tokens.hpp

## 7.80 hocon::problem\_exception Class Reference

Inheritance diagram for hocon::problem\_exception:



### Public Member Functions

- **problem\_exception** ([problem](#) prob)
- [problem](#) const & **get\_problem** () const



### 7.80.1 Detailed Description

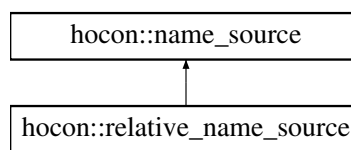
Definition at line 15 of file tokenizer.hpp.

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

- internal/tokenizer.hpp

## 7.81 hocon::relative\_name\_source Class Reference

Inheritance diagram for hocon::relative\_name\_source:



### Public Member Functions

- **relative\_name\_source** (shared\_include\_context context)
- shared\_parseable **name\_to\_parseable** (std::string name, [config\\_parse\\_options](#) parse\_options) const override
- void **set\_context** (shared\_include\_context context)
- shared\_include\_context **get\_context** () const
- bool **context\_initialized** () const

### 7.81.1 Detailed Description

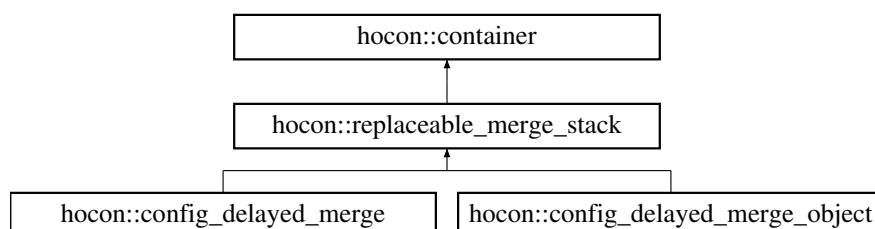
Definition at line 90 of file simple\_includer.hpp.

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

- internal/simple\_includer.hpp

## 7.82 hocon::replaceable\_merge\_stack Class Reference

Inheritance diagram for hocon::replaceable\_merge\_stack:



## Public Member Functions

- virtual shared\_value **make\_replacement** ([resolve\\_context](#) const &context, int skipping) const =0
- virtual shared\_value **replace\_child** (shared\_value const &child, shared\_value replacement) const =0  
*Replace a child of this value.*
- virtual bool **has\_descendant** (shared\_value const &descendant) const =0  
*Super-expensive full traversal to see if descendant is anywhere underneath this container.*

### 7.82.1 Detailed Description

Definition at line 9 of file replaceable\_merge\_stack.hpp.

### 7.82.2 Member Function Documentation

#### 7.82.2.1 has\_descendant()

```
virtual bool hocon::container::has_descendant (
    shared_value const & descendant ) const [pure virtual], [inherited]
```

Super-expensive full traversal to see if descendant is anywhere underneath this container.

Implemented in [hocon::simple\\_config\\_object](#), [hocon::simple\\_config\\_list](#), [hocon::config\\_delayed\\_merge\\_object](#), [hocon::config\\_delayed\\_merge](#), and [hocon::config\\_concatenation](#).

#### 7.82.2.2 replace\_child()

```
virtual shared_value hocon::container::replace_child (
    shared_value const & child,
    shared_value replacement ) const [pure virtual], [inherited]
```

Replace a child of this value.

CAUTION if replacement is null, delete the child, which may also delete the parent, or make the parent into a non-container.

Implemented in [hocon::simple\\_config\\_object](#), [hocon::simple\\_config\\_list](#), [hocon::config\\_delayed\\_merge\\_object](#), [hocon::config\\_delayed\\_merge](#), and [hocon::config\\_concatenation](#).

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

- internal/replaceable\_merge\_stack.hpp

## 7.83 hocon::resolve\_context Class Reference

### Public Member Functions

- **resolve\_context** ([config\\_resolve\\_options](#) options, [path](#) restrict\_to\_child, std::vector< shared\_value > cycle\_markers)
- **resolve\_context** ([config\\_resolve\\_options](#) options, [path](#) restrict\_to\_child)
- bool **is\_restricted\_to\_child** () const
- [config\\_resolve\\_options](#) **options** () const
- [resolve\\_result](#)< shared\_value > **resolve** (shared\_value original, [resolve\\_source](#) const &source) const
- [path](#) **restrict\_to\_child** () const
- [resolve\\_context](#) **add\_cycle\_marker** (shared\_value [value](#)) const
- [resolve\\_context](#) **remove\_cycle\_marker** (shared\_value [value](#))
- [resolve\\_context](#) **restrict** ([path](#) restrict\_to) const
- [resolve\\_context](#) **unrestricted** () const

### Static Public Member Functions

- static shared\_value **resolve** (shared\_value [value](#), shared\_object root, [config\\_resolve\\_options](#) options)

#### 7.83.1 Detailed Description

Definition at line 16 of file `resolve_context.hpp`.

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

- `internal/resolve_context.hpp`

## 7.84 hocon::resolve\_result< V > Struct Template Reference

### Public Member Functions

- **resolve\_result** ([resolve\\_context](#) c, V v)

### Public Attributes

- [resolve\\_context](#) **context**
- V **value**

#### 7.84.1 Detailed Description

```
template<typename V>
struct hocon::resolve_result< V >
```

Definition at line 8 of file `resolve_result.hpp`.

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

- `hocon/config_value.hpp`
- `internal/resolve_result.hpp`

## 7.85 hocon::resolve\_source Class Reference

### Classes

- struct [result\\_with\\_path](#)

### Public Types

- typedef std::list< std::shared\_ptr< const [container](#) > > **node**

### Public Member Functions

- **resolve\_source** (shared\_object root)
- **resolve\_source** (shared\_object root, node path\_from\_root)
- **resolve\_source push\_parent** (std::shared\_ptr< const [container](#) > parent) const
- **result\_with\_path lookup\_subst** ([resolve\\_context](#) context, std::shared\_ptr< [substitution\\_expression](#) > subst, int prefix\_length) const
- **resolve\_source replace\_current\_parent** (std::shared\_ptr< const [container](#) > old, std::shared\_ptr< const [container](#) > replacement) const
- **resolve\_source replace\_within\_current\_parent** (shared\_value old, shared\_value replacement) const
- **resolve\_source reset\_parents** () const

#### 7.85.1 Detailed Description

Definition at line 15 of file resolve\_source.hpp.

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

- internal/resolve\_source.hpp

## 7.86 hocon::resolve\_source::result\_with\_path Struct Reference

### Public Member Functions

- **result\_with\_path** ([resolve\\_result](#)< shared\_value > result\_value, node path\_from\_root\_value)

### Public Attributes

- [resolve\\_result](#)< shared\_value > **result**
- node **path\_from\_root**

#### 7.86.1 Detailed Description

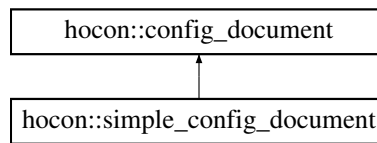
Definition at line 19 of file resolve\_source.hpp.

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

- internal/resolve\_source.hpp

## 7.87 hocon::simple\_config\_document Class Reference

Inheritance diagram for hocon::simple\_config\_document:



### Public Member Functions

- **simple\_config\_document** (std::shared\_ptr< const [config\\_node\\_root](#) > root, [config\\_parse\\_options](#) opts)
- std::unique\_ptr< [config\\_document](#) > [with\\_value\\_text](#) (std::string path, std::string new\_value) const override  
*Returns a new [config\\_document](#) that is a copy of the current [config\\_document](#), but with the desired value set at the desired path.*
- std::unique\_ptr< [config\\_document](#) > [with\\_value](#) (std::string path, std::shared\_ptr< [config\\_value](#) > new\_value) const override  
*Returns a new [config\\_document](#) that is a copy of the current [config\\_document](#), but with the desired value set at the desired path.*
- std::unique\_ptr< [config\\_document](#) > [without\\_path](#) (std::string path) const override  
*Returns a new [config\\_document](#) that is a copy of the current [config\\_document](#), but with all values at the desired path removed.*
- bool [has\\_path](#) (std::string const &path) const override  
*Returns a boolean indicating whether or not a [config\\_document](#) has a value at the desired path.*
- std::string [render](#) () const override  
*The original text of the input, modified if necessary with any replaced or added values.*

### 7.87.1 Detailed Description

Definition at line 10 of file simple\_config\_document.hpp.

### 7.87.2 Member Function Documentation

#### 7.87.2.1 has\_path()

```
bool hocon::simple_config_document::has_path (
    std::string const & path ) const [override], [virtual]
```

Returns a boolean indicating whether or not a [config\\_document](#) has a value at the desired path.

null counts as a value for purposes of this check.

#### Parameters

<i>path</i>	the path to check
-------------	-------------------

**Returns**

true if the path exists in the document, otherwise false

Implements [hocon::config\\_document](#).

**7.87.2.2 render()**

```
std::string hocon::simple_config_document::render ( ) const [override], [virtual]
```

The original text of the input, modified if necessary with any replaced or added values.

**Returns**

the modified original text

Implements [hocon::config\\_document](#).

**7.87.2.3 with\_value()**

```
std::unique_ptr<config_document> hocon::simple_config_document::with_value (
    std::string path,
    std::shared_ptr< config_value > new_value ) const [override], [virtual]
```

Returns a new [config\\_document](#) that is a copy of the current [config\\_document](#), but with the desired value set at the desired path.

Works like `with_value_text(string, string)`, but takes a [config\\_value](#) instead of a string.

**Parameters**

<i>path</i>	the path at which to set the desired value
<i>new_value</i>	the value to set at the desired path, represented as a ConfigValue. The rendered text of the ConfigValue will be inserted into the <a href="#">config_document</a> .

**Returns**

a copy of the [config\\_document](#) with the desired value at the desired path

Implements [hocon::config\\_document](#).

**7.87.2.4 with\_value\_text()**

```
std::unique_ptr<config_document> hocon::simple_config_document::with_value_text (
    std::string path,
    std::string newValue ) const [override], [virtual]
```

Returns a new [config\\_document](#) that is a copy of the current [config\\_document](#), but with the desired value set at the desired path.

If the path exists, it will remove all duplicates before the final occurrence of the path, and replace the value at the final occurrence of the path. If the path does not exist, it will be added. If the document has an array as the root value, an exception will be thrown.

#### Parameters

<i>path</i>	the path at which to set the desired value
<i>new_value</i>	the value to set at the desired path, represented as a string. This string will be parsed into a <a href="#">config_node</a> using the same options used to parse the entire document, and the text will be inserted as-is into the document. Leading and trailing comments, whitespace, or newlines are not allowed, and if present an exception will be thrown. If a concatenation is passed in for new_value but the document was parsed with JSON, the first value in the concatenation will be parsed and inserted into the <a href="#">config_document</a> .

#### Returns

a copy of the [config\\_document](#) with the desired value at the desired path

Implements [hocon::config\\_document](#).

#### 7.87.2.5 without\_path()

```
std::unique_ptr<config_document> hocon::simple_config_document::without_path (
    std::string path ) const [override], [virtual]
```

Returns a new [config\\_document](#) that is a copy of the current [config\\_document](#), but with all values at the desired path removed.

If the path does not exist in the document, a copy of the current document will be returned. If there is an array at the root, an exception will be thrown.

#### Parameters

<i>path</i>	the path to remove from the document
-------------	--------------------------------------

#### Returns

a copy of the [config\\_document](#) with the desired value removed from the document.

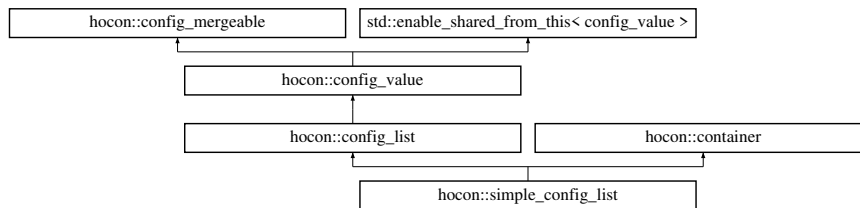
Implements [hocon::config\\_document](#).

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

- internal/simple\_config\_document.hpp

## 7.88 hocon::simple\_config\_list Class Reference

Inheritance diagram for hocon::simple\_config\_list:



### Public Types

- using **iterator** = std::vector< shared\_value >::const\_iterator
- enum class **type** {  
**OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
**CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

*The type of a configuration value (following the [JSON](#) type schema).*

### Public Member Functions

- **simple\_config\_list** (shared\_origin [origin](#), std::vector< shared\_value > [value](#))
- **simple\_config\_list** (shared\_origin [origin](#), std::vector< shared\_value > [value](#), resolve\_status status)
- **config\_value::type** [value\\_type](#) () const override  
*The type of the value; matches the JSON type schema.*
- resolve\_status **get\_resolve\_status** () const override
- shared\_value **replace\_child** (shared\_value const &child, shared\_value replacement) const override  
*Replace a child of this value.*
- bool **has\_descendant** (shared\_value const &descendant) const override  
*Super-expensive full traversal to see if descendant is anywhere underneath this container.*
- shared\_value **relativized** (const std::string prefix) const override  
*This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.*
- bool **contains** (shared\_value v) const
- bool **contains\_all** (std::vector< shared\_value >) const
- int **index\_of** (shared\_value v)
- bool **is\_empty** () const override
- size\_t **size** () const override
- shared\_value **operator[]** (size\_t index) const override
- shared\_value **get** (size\_t index) const override
- iterator **begin** () const override
- iterator **end** () const override
- std::shared\_ptr< const [simple\\_config\\_list](#) > **concatenate** (std::shared\_ptr< const [simple\\_config\\_list](#) > other) const
- unwrapped\_value **unwrapped** () const override
- bool **operator==** ([config\\_value](#) const &other) const override
- virtual shared\_origin const & **origin** () const  
*The origin of the value (file, line number, etc.), for debugging and error messages.*
- char const \* **value\_type\_name** () const  
*The printable name of the value type.*



- virtual std::string **render** () const  
*Renders the config value as a HOCON string.*
- virtual std::string **render** (config\_render\_options options) const  
*Renders the config value to a string, using the provided options.*
- shared\_config **at\_key** (std::string const &key) const  
*Places the value inside a config at the given key.*
- shared\_config **at\_path** (std::string const &path\_expression) const  
*Places the value inside a config at the given path.*
- virtual shared\_value **with\_origin** (shared\_origin origin) const  
*Returns a.*
- std::shared\_ptr< const config\_mergeable > **with\_fallback** (std::shared\_ptr< const config\_mergeable > other) const override  
*Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*
- virtual std::string **transform\_to\_string** () const

## Static Public Member Functions

- static char const \* **type\_name** (type t)

## Protected Member Functions

- resolve\_result< shared\_value > **resolve\_substitutions** (resolve\_context const &context, resolve\_source const &source) const override
- shared\_value **new\_copy** (shared\_origin origin) const override
- void **render** (std::string &result, int indent, bool at\_root, config\_render\_options options) const override
- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &at\_key, config\_render\_options options) const
- shared\_config **at\_key** (shared\_origin origin, std::string const &key) const
- shared\_config **at\_path** (shared\_origin origin, path raw\_path) const
- void **require\_not\_ignoring\_fallbacks** () const
- virtual bool **ignores\_fallbacks** () const
- virtual shared\_value **with\_fallbacks\_ignored** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const unmergeable > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const unmergeable > fallback) const
- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- virtual shared\_value **merged\_with\_object** (shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- virtual shared\_value **construct\_delayed\_merge** (shared\_origin origin, std::vector< shared\_value > stack) const
- shared\_value **to\_fallback\_value** () const override  
*Converts a config to its root object and a config\_value to itself.*

## Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config\_render\_options const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** (config\_value const &other, std::function< bool(T const &)> checker)

### 7.88.1 Detailed Description

Definition at line 15 of file `simple_config_list.hpp`.

### 7.88.2 Member Enumeration Documentation

#### 7.88.2.1 type

```
enum hocon::config_value::type [strong], [inherited]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file `config_value.hpp`.

### 7.88.3 Member Function Documentation

#### 7.88.3.1 at\_key()

```
shared_config hocon::config_value::at_key (  
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also `config_value#at_path(string)`.

##### Parameters

<i>key</i>	key to store this value at.
------------	-----------------------------

##### Returns

**a**  
`config`

instance containing this value at the given key.

#### 7.88.3.2 at\_path()

```
shared_config hocon::config_value::at_path (  
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

## Parameters

<i>path</i>	path to store this value at.
-------------	------------------------------

## Returns

*a*  
*config*  
 instance containing this value at the given path.

**7.88.3.3 has\_descendant()**

```
bool hocon::simple_config_list::has_descendant (
    shared_value const & descendant ) const [override], [virtual]
```

Super-expensive full traversal to see if descendant is anywhere underneath this container.

Implements [hocon::container](#).

**7.88.3.4 origin()**

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

## Returns

where the value came from

**7.88.3.5 relativized()**

```
shared_value hocon::simple_config_list::relativized (
    const std::string prefix ) const [override], [virtual]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at foo.bar in the parent, and the included file as a substitution \${a.b.c}, the included substitution now needs to be \${foo.bar.a.b.c} because we resolve substitutions globally only after parsing everything.

## Parameters

<i>prefix</i>	
---------------	--

## Returns

value relativized to the given path or the same value if nothing to do

Reimplemented from [hocon::config\\_value](#).

**7.88.3.6** `render()` [1/2]

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to  
`render(config_render_options())`

.

## Returns

the rendered value

**7.88.3.7** `render()` [2/2]

```
virtual std::string hocon::config_value::render (
    config_render_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

## Parameters

<i>options</i>	the rendering options
----------------	-----------------------

## Returns

the rendered value

### 7.88.3.8 replace\_child()

```
shared_value hocon::simple_config_list::replace_child (
    shared_value const & child,
    shared_value replacement ) const [override], [virtual]
```

Replace a child of this value.

CAUTION if replacement is null, delete the child, which may also delete the parent, or make the parent into a non-container.

Implements [hocon::container](#).

### 7.88.3.9 to\_fallback\_value()

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

### 7.88.3.10 value\_type()

```
config_value::type hocon::simple_config_list::value_type ( ) const [inline], [override],
[virtual]
```

The type of the value; matches the JSON type schema.

## Returns

value's type

Implements [hocon::config\\_value](#).

Definition at line 20 of file simple\_config\_list.hpp.

### 7.88.3.11 value\_type\_name()

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

#### Returns

value's type's name

Definition at line 92 of file config\_value.hpp.

### 7.88.3.12 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

## Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

## Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

## 7.88.3.13 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```

Returns a.  
`config_value`

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single [config\\_value](#).

## Parameters

<i>origin</i>	the origin set on the returned value
---------------	--------------------------------------

## Returns

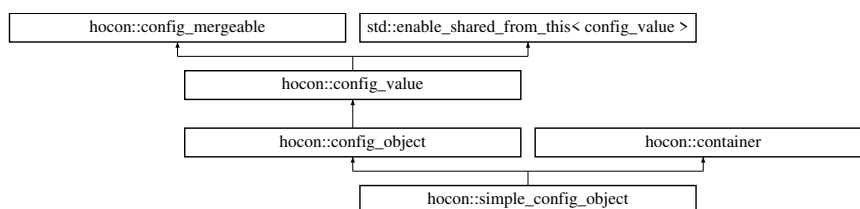
the new [config\\_value](#) with the given origin

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

- `internal/values/simple_config_list.hpp`

## 7.89 hocon::simple\_config\_object Class Reference

Inheritance diagram for `hocon::simple_config_object`:



## Public Types

- using **iterator** = std::unordered\_map< std::string, shared\_value >::const\_iterator
- enum class **type** {  
**OBJECT** , **LIST** , **NUMBER** , **BOOLEAN** ,  
**CONFIG\_NULL** , **STRING** , **UNSPECIFIED** }

*The type of a configuration value (following the [JSON](#) type schema).*

## Public Member Functions

- **simple\_config\_object** (shared\_origin [origin](#), std::unordered\_map< std::string, shared\_value > [value](#), resolve\_status status, bool ignores\_fallbacks)
- **simple\_config\_object** (shared\_origin [origin](#), std::unordered\_map< std::string, shared\_value > [value](#))
- shared\_value [attempt\\_peek\\_with\\_partial\\_resolve](#) (std::string const &key) const override

*Look up the key on an only-partially-resolved object, with no transformation or type conversion of any kind; if 'this' is not resolved then try to look up the key anyway if possible.*

- bool **is\_empty** () const override
- size\_t **size** () const override
- shared\_value **operator[]** (std::string const &key) const override
- iterator **begin** () const override
- iterator **end** () const override
- unwrapped\_value **unwrapped** () const override
- shared\_value **get** (std::string const &key) const override
- std::unordered\_map< std::string, shared\_value > const & **entry\_set** () const override
- resolve\_status **get\_resolve\_status** () const override
- bool **ignores\_fallbacks** () const override
- shared\_value **with\_fallbacks\_ignored** () const override
- shared\_value **merged\_with\_object** (shared\_object fallback) const override
- shared\_object **with\_value** ([path](#) raw\_path, shared\_value [value](#)) const override
- shared\_object **with\_value** (std::string key, shared\_value [value](#)) const override
- shared\_object **without\_path** ([path](#) raw\_path) const override
- shared\_object **with\_only\_path** ([path](#) raw\_path) const override
- shared\_object **with\_only\_path\_or\_null** ([path](#) raw\_path) const override

*Gets the object with only the path if the path exists, otherwise null if it doesn't.*

- shared\_value [replace\\_child](#) (shared\_value const &child, shared\_value replacement) const override

*Replace a child of this value.*

- bool [has\\_descendant](#) (shared\_value const &descendant) const override

*Super-expensive full traversal to see if descendant is anywhere underneath this container.*

- std::vector< std::string > [key\\_set](#) () const override

*Construct a list of keys in the \_value map.*

- std::vector< shared\_value > [value\\_set](#) (std::unordered\_map< std::string, shared\_value > m) const

*Construct a list of the values from the provided map.*

- bool **operator==** ([config\\_value](#) const &other) const override
- virtual std::shared\_ptr< const [config](#) > [to\\_config](#) () const

*Converts this object to a Config instance, enabling you to use path expressions to find values in the object.*

- [config\\_value::type](#) [value\\_type](#) () const override

*The type of the value; matches the JSON type schema.*

- virtual shared\_origin const & [origin](#) () const

*The origin of the value (file, line number, etc.), for debugging and error messages.*

- char const \* [value\\_type\\_name](#) () const

*The printable name of the value type.*

- virtual std::string [render](#) () const



- Renders the config value as a HOCON string.*
  - virtual std::string **render** (config\_render\_options options) const
  - Renders the config value to a string, using the provided options.*
- shared\_config **at\_key** (std::string const &key) const
  - Places the value inside a config at the given key.*
- shared\_config **at\_path** (std::string const &path\_expression) const
  - Places the value inside a config at the given path.*
- virtual shared\_value **with\_origin** (shared\_origin origin) const
  - Returns a.*
- virtual shared\_value **relativized** (std::string prefix) const
  - This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.*
- std::shared\_ptr< const config\_mergeable > **with\_fallback** (std::shared\_ptr< const config\_mergeable > other) const override
  - Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.*
- virtual std::string **transform\_to\_string** () const

## Static Public Member Functions

- static std::shared\_ptr< simple\_config\_object > **empty** ()
- static std::shared\_ptr< simple\_config\_object > **empty** (shared\_origin origin)
- static std::shared\_ptr< simple\_config\_object > **empty\_instance** ()
- static char const \* **type\_name** (type t)

## Protected Member Functions

- resolve\_result**< shared\_value > **resolve\_substitutions** (resolve\_context const &context, resolve\_source const &source) const override
- shared\_value **new\_copy** (shared\_origin) const override
- void **render** (std::string &s, int indent, bool at\_root, config\_render\_options options) const override
- shared\_value **peek\_path** (path desired\_path) const
- shared\_value **peek\_assuming\_resolved** (std::string const &key, path original\_path) const
- shared\_value **construct\_delayed\_merge** (shared\_origin origin, std::vector< shared\_value > stack) const override
- virtual void **render** (std::string &result, int indent, bool at\_root, std::string const &at\_key, config\_render\_options options) const
- shared\_config **at\_key** (shared\_origin origin, std::string const &key) const
- shared\_config **at\_path** (shared\_origin origin, path raw\_path) const
- void **require\_not\_ignoring\_fallbacks** () const
- shared\_value **merged\_with\_the\_unmergeable** (std::vector< shared\_value > stack, std::shared\_ptr< const unmergeable > fallback) const
- shared\_value **merged\_with\_the\_unmergeable** (std::shared\_ptr< const unmergeable > fallback) const
- shared\_value **merged\_with\_object** (std::vector< shared\_value > stack, shared\_object fallback) const
- shared\_value **merged\_with\_non\_object** (std::vector< shared\_value > stack, shared\_value fallback) const
- shared\_value **merged\_with\_non\_object** (shared\_value fallback) const
- shared\_value **to\_fallback\_value** () const override
- Converts a config to its root object and a config\_value to itself.*

## Static Protected Member Functions

- static shared\_value **peek\_path** (const [config\\_object](#) \*self, [path](#) desired\_path)
- static shared\_origin **merge\_origins** (std::vector< shared\_value > const &stack)
- static void **indent** (std::string &result, int indent, [config\\_render\\_options](#) const &options)
- static std::vector< shared\_value > **replace\_child\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &child, shared\_value replacement)
- static bool **has\_descendant\_in\_list** (std::vector< shared\_value > const &values, shared\_value const &descendant)
- template<typename T >  
static bool **equals** ([config\\_value](#) const &other, std::function< bool(T const &)> checker)

### 7.89.1 Detailed Description

Definition at line 11 of file simple\_config\_object.hpp.

### 7.89.2 Member Enumeration Documentation

#### 7.89.2.1 type

```
enum hocon::config\_value::type [strong], [inherited]
```

The type of a configuration value (following the [JSON](#) type schema).

Definition at line 56 of file config\_value.hpp.

### 7.89.3 Member Function Documentation

#### 7.89.3.1 at\_key()

```
shared_config hocon::config_value::at_key (  
    std::string const & key ) const [inherited]
```

Places the value inside a [config](#) at the given key.

See also `config_value#at_path(string)`.

#### Parameters

<i>key</i>	key to store this value at.
------------	-----------------------------

**Returns**

a  
config

instance containing this value at the given key.

**7.89.3.2 at\_path()**

```
shared_config hocon::config_value::at_path (
    std::string const & path_expression ) const [inherited]
```

Places the value inside a [config](#) at the given path.

See also `config_value#at_key(String)`.

**Parameters**

<i>path</i>	path to store this value at.
-------------	------------------------------

**Returns**

a  
config

instance containing this value at the given path.

**7.89.3.3 attempt\_peek\_with\_partial\_resolve()**

```
shared_value hocon::simple_config_object::attempt_peek_with_partial_resolve (
    std::string const & key ) const [override], [virtual]
```

Look up the key on an only-partially-resolved object, with no transformation or type conversion of any kind; if 'this' is not resolved then try to look up the key anyway if possible.

**Parameters**

<i>key</i>	key to look up
------------	----------------

**Returns**

the value of the key, or null if known not to exist

**Exceptions**

<a href="#">config_exception</a>	if can't figure out key's value (or existence) without more resolving
----------------------------------	---

Implements [hocon::config\\_object](#).

#### 7.89.3.4 has\_descendant()

```
bool hocon::simple_config_object::has_descendant (
    shared_value const & descendant ) const [override], [virtual]
```

Super-expensive full traversal to see if descendant is anywhere underneath this container.

Implements [hocon::container](#).

#### 7.89.3.5 key\_set()

```
std::vector<std::string> hocon::simple_config_object::key_set ( ) const [override], [virtual]
```

Construct a list of keys in the `_value` map.

Use a vector rather than set, because most of the time we just want to iterate over them.

Implements [hocon::config\\_object](#).

#### 7.89.3.6 origin()

```
virtual shared_origin const& hocon::config_value::origin ( ) const [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

##### Returns

where the value came from

#### 7.89.3.7 relativized()

```
virtual shared_value hocon::config_value::relativized (
    std::string prefix ) const [inline], [virtual], [inherited]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at `foo.bar` in the parent, and the included file as a substitution `$(a.b.c)`, the included substitution now needs to be `$(foo.bar.a.b.c)` because we resolve substitutions globally only after parsing everything.

## Parameters

<i>prefix</i>	
---------------	--

## Returns

value relativized to the given path or the same value if nothing to do

Reimplemented in [hocon::config\\_concatenation](#), and [hocon::simple\\_config\\_list](#).

Definition at line 181 of file `config_value.hpp`.

**7.89.3.8 render()** [1/2]

```
virtual std::string hocon::config_value::render ( ) const [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to

```
render(config_render_options())
```

.

## Returns

the rendered value

**7.89.3.9 render()** [2/2]

```
virtual std::string hocon::config_value::render (
    config\_render\_options options ) const [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see [config#resolve](#)), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

**Parameters**

<i>options</i>	the rendering options
----------------	-----------------------

**Returns**

the rendered value

**7.89.3.10 replace\_child()**

```
shared_value hocon::simple_config_object::replace_child (
    shared_value const & child,
    shared_value replacement ) const [override], [virtual]
```

Replace a child of this value.

CAUTION if replacement is null, delete the child, which may also delete the parent, or make the parent into a non-container.

Implements [hocon::container](#).

**7.89.3.11 to\_config()**

```
virtual std::shared_ptr<const config> hocon::config_object::to_config ( ) const [virtual],
[inherited]
```

Converts this object to a Config instance, enabling you to use path expressions to find values in the object.

This is a constant-time operation (it is not proportional to the size of the object).

**Returns**

a Config with this object as its root

**7.89.3.12 to\_fallback\_value()**

```
shared_value hocon::config_value::to_fallback_value ( ) const [override], [protected], [virtual],
[inherited]
```

Converts a config to its root object and a [config\\_value](#) to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements [hocon::config\\_mergeable](#).

### 7.89.3.13 value\_set()

```
std::vector<shared_value> hocon::simple_config_object::value_set (
    std::unordered_map< std::string, shared_value > m ) const
```

Construct a list of the values from the provided map.

Equivalent to Java's `Collection.values()` method.

### 7.89.3.14 value\_type()

```
config_value::type hocon::config_object::value_type ( ) const [override], [virtual], [inherited]
```

The type of the value; matches the JSON type schema.

#### Returns

value's type

Implements [hocon::config\\_value](#).

### 7.89.3.15 value\_type\_name()

```
char const* hocon::config_value::value_type_name ( ) const [inline], [inherited]
```

The printable name of the value type.

#### Returns

value's type's name

Definition at line 92 of file `config_value.hpp`.

### 7.89.3.16 with\_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
    std::shared_ptr< const config_mergeable > other ) const [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the [spec for HOCON](#). Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

#### Parameters

<i>other</i>	an object whose keys should be used as fallbacks, if the keys are not present in this one
--------------	---

#### Returns

a new object (or the original one, if the fallback doesn't get used)

Implements [hocon::config\\_mergeable](#).

### 7.89.3.17 with\_only\_path\_or\_null()

```
shared_object hocon::simple_config_object::with_only_path_or_null (
    path raw_path ) const [override], [virtual]
```



Gets the object with only the path if the path exists, otherwise null if it doesn't.

this ensures that if we have { a : { b : 42 } } and do withOnlyPath("a.b.c") that we don't keep an empty "a" object.

Implements [hocon::config\\_object](#).

### 7.89.3.18 with\_origin()

```
virtual shared_value hocon::config_value::with_origin (
    shared_origin origin ) const [virtual], [inherited]
```

Returns a.

`config_value`

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single [config\\_value](#).

#### Parameters

<i>origin</i>	the origin set on the returned value
---------------	--------------------------------------

#### Returns

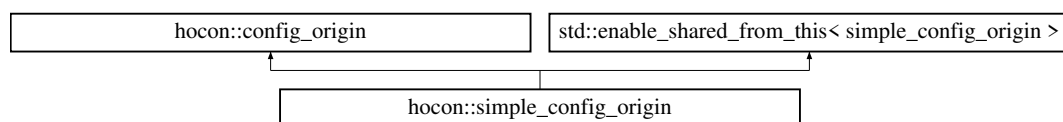
the new [config\\_value](#) with the given origin

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

- internal/values/simple\_config\_object.hpp

## 7.90 hocon::simple\_config\_origin Class Reference

Inheritance diagram for hocon::simple\_config\_origin:



### Public Member Functions

- **simple\_config\_origin** (std::string [description](#), int [line\\_number](#), int end\_line\_number, origin\_type org\_type, std::string resource\_or\_null, std::vector< std::string > comments\_or\_null)
- **simple\_config\_origin** (std::string [description](#), int [line\\_number](#)=-1, int end\_line\_number=-1, origin\_type org\_type=origin\_type::GENERIC)

*This constructor replaces the new\_simple method in the original library.*

- int [line\\_number](#) () const override

- Returns a line number where the value or exception originated.*
- `std::string const & description ()` const override
- Returns a string describing the origin of a value or exception.*
- `std::vector< std::string > const & comments ()` const override
- Returns any comments that appeared to "go with" this place in the file.*
- `shared_origin with_line_number (int line_number)` const override
- Returns a pointer to a copy of this origin with the specified line number as both starting and ending line.*
- `shared_origin with_comments (std::vector< std::string > comments)` const override
- Returns a.*
- `std::shared_ptr< const simple_config_origin > append_comments (std::vector< std::string > comments)` const
- `std::shared_ptr< const simple_config_origin > prepend_comments (std::vector< std::string > comments)` const
- `bool operator== (const simple_config_origin &other)` const
- `bool operator!= (const simple_config_origin &other)` const

## Static Public Member Functions

- static `shared_origin merge_origins (shared_origin a, shared_origin b)`
- static `shared_origin merge_origins (std::vector< shared_value > const &stack)`
- static `shared_origin merge_origins (std::vector< shared_origin > const &stack)`

### 7.90.1 Detailed Description

Definition at line 13 of file `simple_config_origin.hpp`.

### 7.90.2 Constructor & Destructor Documentation

#### 7.90.2.1 simple\_config\_origin()

```
hocon::simple_config_origin::simple_config_origin (
    std::string description,
    int line_number = -1,
    int end_line_number = -1,
    origin_type org_type = origin_type::GENERIC )
```

This constructor replaces the `new_simple` method in the original library.

### 7.90.3 Member Function Documentation

### 7.90.3.1 comments()

```
std::vector<std::string> const& hocon::simple_config_origin::comments ( ) const [override],  
[virtual]
```

Returns any comments that appeared to "go with" this place in the file.

Often an empty list, but never null. The details of this are subject to change, but at the moment comments that are immediately before an array element or object field, with no blank line after the comment, "go with" that element or field.

#### Returns

any comments that seemed to "go with" this origin, empty list if none

Implements [hocon::config\\_origin](#).

### 7.90.3.2 description()

```
std::string const& hocon::simple_config_origin::description ( ) const [override], [virtual]
```

Returns a string describing the origin of a value or exception.

This will never return null.

#### Returns

string describing the origin

Implements [hocon::config\\_origin](#).

### 7.90.3.3 line\_number()

```
int hocon::simple_config_origin::line_number ( ) const [override], [virtual]
```

Returns a line number where the value or exception originated.

This will return -1 if there's no meaningful line number.

#### Returns

line number or -1 if none is available

Implements [hocon::config\\_origin](#).

### 7.90.3.4 with\_comments()

```
shared_origin hocon::simple_config_origin::with_comments (
    std::vector< std::string > comments ) const [override], [virtual]
```

Returns a.  
`config_origin`

based on this one, but with the given comments. Does not modify this instance or any  
`config_value`

s with this origin (since they are immutable). To set the returned origin to a  
`config_value`

, use `config_value#with_origin`.

Note that when the given comments are equal to the comments on this object, a new instance may not be created and  
`this`

is returned directly.

Since

1.3.0

Parameters

<code>comments</code>	the comments used on the returned origin
-----------------------	--

Returns

the `config_origin` with the given comments

Implements `hocon::config_origin`.

### 7.90.3.5 with\_line\_number()

```
shared_origin hocon::simple_config_origin::with_line_number (
    int line_number ) const [override], [virtual]
```

Returns a pointer to a copy of this origin with the specified line number as both starting and ending line.

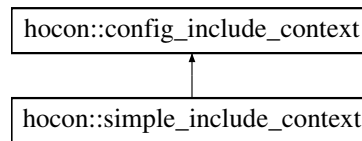
Implements `hocon::config_origin`.

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

- `internal/simple_config_origin.hpp`

## 7.91 hocon::simple\_include\_context Class Reference

Inheritance diagram for hocon::simple\_include\_context:



### Public Member Functions

- **simple\_include\_context** ([parseable](#) const &[parseable](#))
- shared\_parseable [relative\\_to](#) (std::string file\_name) const override  
*Tries to find a name relative to whatever is doing the including, for example in the same directory as the file doing the including.*
- [config\\_parse\\_options](#) [parse\\_options](#) () const override  
*Parse options to use (if you use another method to get a [config\\_parseable](#) then use [config\\_parseable#options\(\)](#) instead though).*
- void **set\_cur\_dir** (std::string dir) const
- std::string **get\_cur\_dir** () const

### Protected Attributes

- std::shared\_ptr< std::string > **\_cur\_dir**

#### 7.91.1 Detailed Description

Definition at line 8 of file simple\_include\_context.hpp.

#### 7.91.2 Member Function Documentation

##### 7.91.2.1 parse\_options()

```
config\_parse\_options hocon::simple_include_context::parse_options ( ) const [override], [virtual]
```

Parse options to use (if you use another method to get a [config\\_parseable](#) then use [config\\_parseable#options\(\)](#) instead though).

##### Returns

the parse options

Implements [hocon::config\\_include\\_context](#).

### 7.91.2.2 `relative_to()`

```
shared_parseable hocon::simple_include_context::relative_to (
    std::string file_name ) const [override], [virtual]
```

Tries to find a name relative to whatever is doing the including, for example in the same directory as the file doing the including.

Returns null if it can't meaningfully create a relative name. The returned parseable may not exist; this function is not required to do any IO, just compute what the name would be.

The passed-in filename has to be a complete name (with extension), not just a basename. (Include statements in config files are allowed to give just a basename.)

#### Parameters

<i>filename</i>	the name to make relative to the resource doing the including
-----------------	---

#### Returns

parseable item relative to the resource doing the including, or null

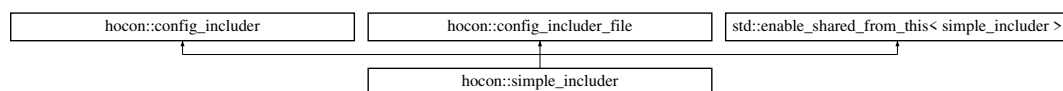
Implements [hocon::config\\_include\\_context](#).

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

- `internal/simple_include_context.hpp`

## 7.92 `hocon::simple_includer` Class Reference

Inheritance diagram for `hocon::simple_includer`:



### Public Member Functions

- **simple\_includer** (shared\_includer fallback)
- shared\_includer **with\_fallback** (shared\_includer fallback) const override  
*Returns a new includer that falls back to the given includer.*
- shared\_object **include** (shared\_include\_context context, std::string what) const override  
*Parses another item to be included.*
- shared\_object **include\_without\_fallback** (shared\_include\_context context, std::string what) const
- shared\_object **include\_file** (shared\_include\_context context, std::string what) const override  
*Parses another item to be included.*

## Static Public Member Functions

- static shared\_object **include\_file\_without\_fallback** (shared\_include\_context context, std::string what)
- static [config\\_parse\\_options](#) **clear\_for\_include** ([config\\_parse\\_options](#) const &options)
- static shared\_object **from\_basename** (std::shared\_ptr< [name\\_source](#) > source, std::string name, [config\\_parse\\_options](#) options)
- static std::shared\_ptr< const [full\\_includer](#) > **make\_full** (std::shared\_ptr< const [config\\_includer](#) > includer)

### 7.92.1 Detailed Description

Definition at line 12 of file simple\_includer.hpp.

### 7.92.2 Member Function Documentation

#### 7.92.2.1 include()

```
shared_object hocon::simple_includer::include (
    shared_include_context context,
    std::string what ) const [override], [virtual]
```

Parses another item to be included.

The returned object typically would not have substitutions resolved. You can throw a [config\\_exception](#) here to abort parsing, or return an empty object, but may not return null.

This method is used for a "heuristic" include statement that does not specify file, or URL resource. If the include statement does specify, then the same class implementing [config\\_includer](#) must also implement [config\\_includer\\_file](#) or [config\\_includer\\_URL](#) as needed, or a default includer will be used.

#### Parameters

<i>context</i>	some info about the include context
<i>what</i>	the include statement's argument

#### Returns

a non-null [config\\_object](#)

Implements [hocon::config\\_includer](#).

#### 7.92.2.2 include\_file()

```
shared_object hocon::simple_includer::include_file (
    shared_include_context context,
    std::string what ) const [override], [virtual]
```

Parses another item to be included.

The returned object typically would not have substitutions resolved. You can throw a [config\\_exception](#) here to abort parsing, or return an empty object, but may not return null.

#### Parameters

<i>context</i>	some info about the include context
<i>what</i>	the include statement's argument (a file path)

#### Returns

a non-null [config\\_object](#)

Implements [hocon::config\\_includer\\_file](#).

### 7.92.2.3 with\_fallback()

```
shared_includer hocon::simple_includer::with_fallback (
    shared_includer fallback ) const [override], [virtual]
```

Returns a new includer that falls back to the given includer.

This is how you can obtain the default includer; it will be provided as a fallback. It's up to your includer to chain to it if you want to. You might want to merge any files found by the fallback includer with any objects you load yourself.

It's important to handle the case where you already have the fallback with a "return this", i.e. this method should not create a new object if the fallback is the same one you already have. The same fallback may be added repeatedly.

#### Parameters

<i>fallback</i>	the previous includer for chaining
-----------------	------------------------------------

#### Returns

a new includer

Implements [hocon::config\\_includer](#).

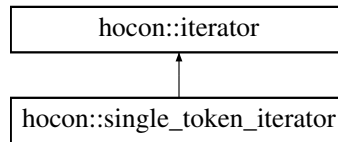
The documentation for this class was generated from the following file:

- internal/simple\_includer.hpp

## 7.93 hocon::single\_token\_iterator Class Reference

Inheritance diagram for `hocon::single_token_iterator`:





## Public Member Functions

- **single\_token\_iterator** (shared\_token [token](#))
- bool **has\_next** () override
- shared\_token **next** () override

### 7.93.1 Detailed Description

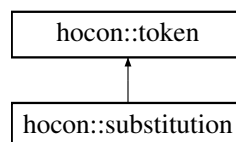
Definition at line 119 of file tokenizer.hpp.

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

- internal/tokenizer.hpp

## 7.94 hocon::substitution Class Reference

Inheritance diagram for hocon::substitution:



## Public Member Functions

- **substitution** (shared\_origin origin, bool optional, token\_list expression)
- bool **optional** () const
- token\_list const & **expression** () const
- std::string **token\_text** () const override
- std::string **to\_string** () const override
- bool **operator==** (const [token](#) &other) const override
- virtual token\_type **get\_token\_type** () const
- virtual shared\_origin const & **origin** () const
- int **line\_number** () const

### 7.94.1 Detailed Description

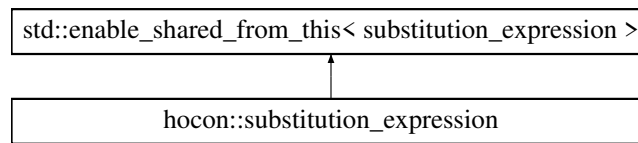
Definition at line 96 of file tokens.hpp.

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

- internal/tokens.hpp

## 7.95 hocon::substitution\_expression Class Reference

Inheritance diagram for hocon::substitution\_expression:



### Public Member Functions

- **substitution\_expression** ([path](#) the\_path, bool optional)
- [path](#) **get\_path** () const
- bool **optional** () const
- std::shared\_ptr< [substitution\\_expression](#) > **change\_path** ([path](#) new\_path)
- std::string **to\_string** () const
- bool **operator==** ([substitution\\_expression](#) const &other) const

### 7.95.1 Detailed Description

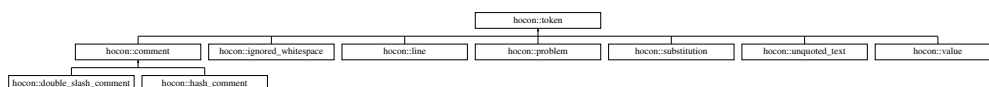
Definition at line 9 of file substitution\_expression.hpp.

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

- internal/substitution\_expression.hpp

## 7.96 hocon::token Class Reference

Inheritance diagram for hocon::token:



### Public Member Functions

- **token** (token\_type type, shared\_origin origin=nullptr, std::string token\_text="", std::string debug\_string="")
- virtual token\_type **get\_token\_type** () const
- virtual std::string **token\_text** () const
- virtual std::string **to\_string** () const
- virtual shared\_origin const & **origin** () const
- int **line\_number** () const
- virtual bool **operator==** (const [token](#) &other) const

### 7.96.1 Detailed Description

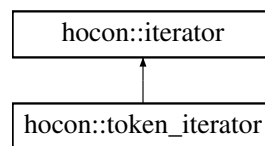
Definition at line 18 of file token.hpp.

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

- internal/token.hpp

## 7.97 hocon::token\_iterator Class Reference

Inheritance diagram for hocon::token\_iterator:



### Public Member Functions

- **token\_iterator** (shared\_origin origin, std::unique\_ptr< std::istream > input, bool allow\_comments)
- **token\_iterator** (shared\_origin origin, std::unique\_ptr< std::istream > input, config\_syntax flavor)
- bool **has\_next** () override
- shared\_token **next** () override

### Static Public Member Functions

- static std::string **render** (token\_list [tokens](#))

### 7.97.1 Detailed Description

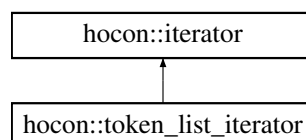
Definition at line 49 of file tokenizer.hpp.

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

- internal/tokenizer.hpp

## 7.98 hocon::token\_list\_iterator Class Reference

Inheritance diagram for hocon::token\_list\_iterator:



## Public Member Functions

- **token\_list\_iterator** (token\_list [tokens](#))
- bool **has\_next** () override
- shared\_token **next** () override

### 7.98.1 Detailed Description

Definition at line 131 of file tokenizer.hpp.

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

- internal/tokenizer.hpp

## 7.99 hocon::tokens Class Reference

### Static Public Member Functions

- static shared\_token const & [start\\_token](#) ()  
*Singleton tokens.*
- static shared\_token const & **end\_token** ()
- static shared\_token const & **comma\_token** ()
- static shared\_token const & **equals\_token** ()
- static shared\_token const & **colon\_token** ()
- static shared\_token const & **open\_curly\_token** ()
- static shared\_token const & **close\_curly\_token** ()
- static shared\_token const & **open\_square\_token** ()
- static shared\_token const & **close\_square\_token** ()
- static shared\_token const & **plus\_equals\_token** ()
- static bool **is\_newline** (shared\_token)
- static bool **is\_ignored\_whitespace** (shared\_token)
- static bool **is\_value\_with\_type** (shared\_token t, [config\\_value::type](#) type)
- static shared\_value **get\_value** (shared\_token t)

### 7.99.1 Detailed Description

Definition at line 113 of file tokens.hpp.

### 7.99.2 Member Function Documentation

### 7.99.2.1 start\_token()

```
static shared_token const& hocon::tokens::start_token ( ) [static]
```

Singleton tokens.

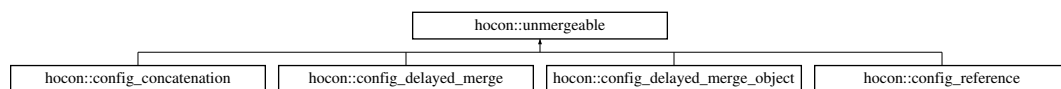
The documentation for this class was generated from the following file:

- internal/tokens.hpp

## 7.100 hocon::unmergeable Class Reference

Interface that tags a ConfigValue that is not mergeable until after substitutions are resolved.

Inheritance diagram for hocon::unmergeable:



### Public Member Functions

- virtual std::vector< shared\_value > **unmerged\_values** () const =0

### 7.100.1 Detailed Description

Interface that tags a ConfigValue that is not mergeable until after substitutions are resolved.

Basically these are special ConfigValue that never appear in a resolved tree, like ConfigSubstitution and ConfigDelayedMerge.

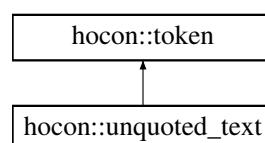
Definition at line 14 of file unmergeable.hpp.

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

- internal/unmergeable.hpp

## 7.101 hocon::unquoted\_text Class Reference

Inheritance diagram for hocon::unquoted\_text:



## Public Member Functions

- **unquoted\_text** (shared\_origin origin, std::string text)
- std::string **to\_string** () const override
- bool **operator==** (const [token](#) &other) const override
- virtual token\_type **get\_token\_type** () const
- virtual std::string **token\_text** () const
- virtual shared\_origin const & **origin** () const
- int **line\_number** () const

### 7.101.1 Detailed Description

Definition at line 33 of file tokens.hpp.

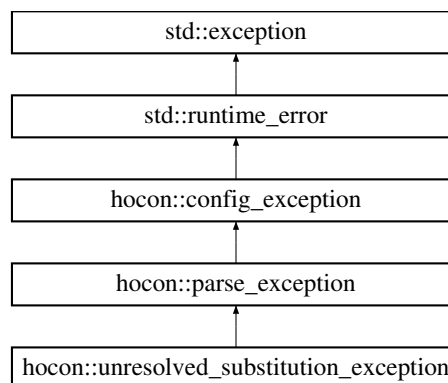
The documentation for this class was generated from the following file:

- internal/tokens.hpp

## 7.102 hocon::unresolved\_substitution\_exception Struct Reference

Exception indicating that a substitution did not resolve to anything.

Inheritance diagram for hocon::unresolved\_substitution\_exception:



## Public Member Functions

- **unresolved\_substitution\_exception** ([config\\_origin](#) const &origin, std::string const &detail)

### 7.102.1 Detailed Description

Exception indicating that a substitution did not resolve to anything.

Thrown by [config#resolve](#).

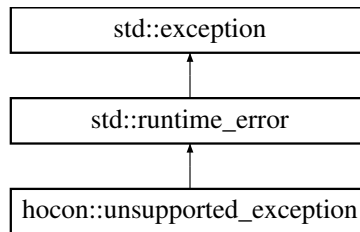
Definition at line 110 of file config\_exception.hpp.

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

- hocon/config\_exception.hpp

## 7.103 hocon::unsupported\_exception Struct Reference

Inheritance diagram for hocon::unsupported\_exception:



### Public Member Functions

- **unsupported\_exception** (std::string const &message)

#### 7.103.1 Detailed Description

Definition at line 14 of file token.hpp.

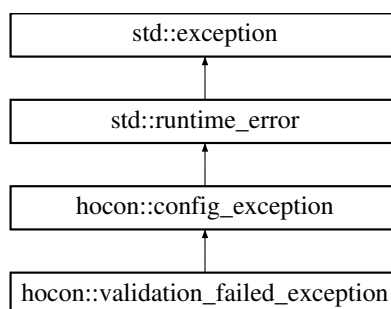
The documentation for this struct was generated from the following file:

- internal/token.hpp

## 7.104 hocon::validation\_failed\_exception Struct Reference

Exception indicating that [config#check\\_valid](#) found validity problems.

Inheritance diagram for hocon::validation\_failed\_exception:



### Public Member Functions

- **validation\_failed\_exception** (std::vector< [validation\\_problem](#) > problems\_)

### Public Attributes

- const std::vector< [validation\\_problem](#) > **problems**

### 7.104.1 Detailed Description

Exception indicating that [config#check\\_valid](#) found validity problems.

The problems are available via the `problems()` method. The `get_message()` of this exception is a potentially very long string listing all the problems found.

Definition at line 155 of file `config_exception.hpp`.

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

- `hocon/config_exception.hpp`

## 7.105 hocon::validation\_problem Struct Reference

Information about a problem that occurred in [config#check\\_valid](#).

### Public Member Functions

- **validation\_problem** (std::string path\_, shared\_origin origin\_, std::string problem\_)
- std::string **to\_string** ()

### Public Attributes

- const std::string **path**
- const shared\_origin **origin**
- const std::string **problem**

### 7.105.1 Detailed Description

Information about a problem that occurred in [config#check\\_valid](#).

A [validation\\_failed\\_exception](#) thrown from `check_valid()` includes a list of problems encountered.

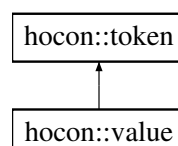
Definition at line 136 of file `config_exception.hpp`.

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

- `hocon/config_exception.hpp`

## 7.106 hocon::value Class Reference

Inheritance diagram for `hocon::value`:





## Public Member Functions

- **value** (shared\_value value)
- **value** (shared\_value value, std::string original\_text)
- std::string **to\_string** () const override
- shared\_origin const & **origin** () const override
- shared\_value **get\_value** () const
- bool **operator==** (const token &other) const override
- virtual token\_type **get\_token\_type** () const
- virtual std::string **token\_text** () const
- int **line\_number** () const

### 7.106.1 Detailed Description

Definition at line 8 of file tokens.hpp.

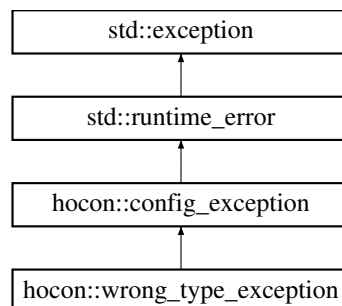
The documentation for this class was generated from the following file:

- internal/tokens.hpp

## 7.107 hocon::wrong\_type\_exception Struct Reference

Exception indicating that the type of a value does not match the type you requested.

Inheritance diagram for hocon::wrong\_type\_exception:



## Public Member Functions

- **wrong\_type\_exception** (config\_origin const &origin, std::string const &path, std::string const &expected, std::string const &actual)
- **config\_exception** (config\_origin const &origin, std::string const &message)
- **config\_exception** (std::string const &message)
- **config\_exception** (std::string const &message, std::exception const &e)

### 7.107.1 Detailed Description

Exception indicating that the type of a value does not match the type you requested.

Definition at line 27 of file config\_exception.hpp.

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

- hocon/config\_exception.hpp



## Chapter 8

# File Documentation

### 8.1 hocon/version.h File Reference

Declares macros for the cpp-hocon library version.

#### Macros

- `#define CPP_HOCON_VERSION_MAJOR 0`  
*The cpp-hocon library major version.*
- `#define CPP_HOCON_VERSION_MINOR 3`  
*The cpp-hocon library minor version.*
- `#define CPP_HOCON_VERSION_PATCH 0`  
*The cpp-hocon library patch version.*
- `#define CPP_HOCON_VERSION "0.3.0"`  
*The cpp-hocon library version as a string (without commit SHA1).*
- `#define CPP_HOCON_VERSION_WITH_COMMIT "0.3.0"`  
*The cpp-hocon library version as a string (with commit SHA1).*

#### 8.1.1 Detailed Description

Declares macros for the cpp-hocon library version.

#### 8.1.2 Macro Definition Documentation

##### 8.1.2.1 CPP\_HOCON\_VERSION

```
#define CPP_HOCON_VERSION "0.3.0"
```

The cpp-hocon library version as a string (without commit SHA1).

Definition at line 23 of file version.h.

#### 8.1.2.2 CPP\_HOCON\_VERSION\_MAJOR

```
#define CPP_HOCON_VERSION_MAJOR 0
```

The cpp-hocon library major version.

Definition at line 10 of file version.h.

#### 8.1.2.3 CPP\_HOCON\_VERSION\_MINOR

```
#define CPP_HOCON_VERSION_MINOR 3
```

The cpp-hocon library minor version.

Definition at line 14 of file version.h.

#### 8.1.2.4 CPP\_HOCON\_VERSION\_PATCH

```
#define CPP_HOCON_VERSION_PATCH 0
```

The cpp-hocon library patch version.

Definition at line 18 of file version.h.

#### 8.1.2.5 CPP\_HOCON\_VERSION\_WITH\_COMMIT

```
#define CPP_HOCON_VERSION_WITH_COMMIT "0.3.0"
```

The cpp-hocon library version as a string (with commit SHA1).

Definition at line 28 of file version.h.

# Index

- append\_includer
  - hocon::config\_parse\_options, 154
- at\_key
  - hocon::config, 30
  - hocon::config\_boolean, 44
  - hocon::config\_concatenation, 52
  - hocon::config\_delayed\_merge, 60
  - hocon::config\_delayed\_merge\_object, 68
  - hocon::config\_double, 80
  - hocon::config\_int, 92
  - hocon::config\_list, 100
  - hocon::config\_long, 107
  - hocon::config\_null, 130
  - hocon::config\_number, 137
  - hocon::config\_object, 144
  - hocon::config\_reference, 161
  - hocon::config\_string, 176
  - hocon::config\_value, 184
  - hocon::simple\_config\_list, 230
  - hocon::simple\_config\_object, 238
- at\_path
  - hocon::config, 30
  - hocon::config\_boolean, 45
  - hocon::config\_concatenation, 53
  - hocon::config\_delayed\_merge, 60
  - hocon::config\_delayed\_merge\_object, 68
  - hocon::config\_double, 80
  - hocon::config\_int, 93
  - hocon::config\_list, 100
  - hocon::config\_long, 108
  - hocon::config\_null, 130
  - hocon::config\_number, 137
  - hocon::config\_object, 144
  - hocon::config\_reference, 162
  - hocon::config\_string, 177
  - hocon::config\_value, 184
  - hocon::simple\_config\_list, 230
  - hocon::simple\_config\_object, 239
- attempt\_peek\_with\_partial\_resolve
  - hocon::config\_delayed\_merge\_object, 68
  - hocon::config\_object, 145
  - hocon::simple\_config\_object, 239
- check\_valid
  - hocon::config, 31
- comments
  - hocon::config\_origin, 150
  - hocon::simple\_config\_origin, 246
- concise
  - hocon::config\_render\_options, 168
- config\_parse\_options
  - hocon::config\_parse\_options, 154
- config\_render\_options
  - hocon::config\_render\_options, 168
- config\_resolve\_options
  - hocon::config\_resolve\_options, 172
- CPP\_HOCON\_VERSION
  - version.h, 263
- CPP\_HOCON\_VERSION\_MAJOR
  - version.h, 263
- CPP\_HOCON\_VERSION\_MINOR
  - version.h, 264
- CPP\_HOCON\_VERSION\_PATCH
  - version.h, 264
- CPP\_HOCON\_VERSION\_WITH\_COMMIT
  - version.h, 264
- defaults
  - hocon::config\_parse\_options, 154
- description
  - hocon::config\_origin, 150
  - hocon::simple\_config\_origin, 247
- duration
  - hocon, 19
- entry\_set
  - hocon::config, 32
- from\_any\_ref
  - hocon::config\_value\_factory, 189
- FwdListIter< T >, 195
- get\_allow\_missing
  - hocon::config\_parse\_options, 154
- get\_allow\_unresolved
  - hocon::config\_resolve\_options, 172
- get\_comments
  - hocon::config\_render\_options, 168
- get\_duration
  - hocon::config, 32
- get\_formatted
  - hocon::config\_render\_options, 169
- get\_includer
  - hocon::config\_parse\_options, 155
- get\_is\_null
  - hocon::config, 33
- get\_json
  - hocon::config\_render\_options, 169
- get\_origin\_comments
  - hocon::config\_render\_options, 169

- get\_origin\_description
  - hocon::config\_parse\_options, 155
- get\_syntax
  - hocon::config\_parse\_options, 155
- get\_use\_system\_environment
  - hocon::config\_resolve\_options, 173
- has\_descendant
  - hocon::config\_concatenation, 53
  - hocon::config\_delayed\_merge, 60
  - hocon::config\_delayed\_merge\_object, 69
  - hocon::container, 190
  - hocon::replaceable\_merge\_stack, 222
  - hocon::simple\_config\_list, 231
  - hocon::simple\_config\_object, 240
- has\_funky\_chars
  - hocon::path, 218
- has\_path
  - hocon::config, 33
  - hocon::config\_document, 75
  - hocon::simple\_config\_document, 225
- has\_path\_or\_null
  - hocon::config, 34
- hocon, 15
  - duration, 19
  - operator==, 19
- hocon/version.h, 263
- hocon::abstract\_config\_node, 21
  - render, 21
- hocon::abstract\_config\_node\_value, 22
  - render, 22
- hocon::bad\_path\_exception, 23
- hocon::bad\_value\_exception, 23
- hocon::bug\_or\_broken\_exception, 24
- hocon::comment, 25
- hocon::config, 25
  - at\_key, 30
  - at\_path, 30
  - check\_valid, 31
  - entry\_set, 32
  - get\_duration, 32
  - get\_is\_null, 33
  - has\_path, 33
  - has\_path\_or\_null, 34
  - is\_empty, 34
  - is\_resolved, 35
  - origin, 35
  - parse\_file\_any\_syntax, 35, 36
  - parse\_string, 36, 37
  - resolve, 37, 38
  - resolve\_with, 38, 39
  - root, 39
  - to\_fallback\_value, 39
  - with\_fallback, 40
  - with\_only\_path, 41
  - with\_value, 41
  - without\_path, 42
- hocon::config\_boolean, 42
  - at\_key, 44
  - at\_path, 45
  - origin, 45
  - relativized, 45
  - render, 47
  - to\_fallback\_value, 48
  - type, 44
  - value\_type, 48
  - value\_type\_name, 48
  - with\_fallback, 48
  - with\_origin, 49
- hocon::config\_concatenation, 50
  - at\_key, 52
  - at\_path, 53
  - has\_descendant, 53
  - origin, 53
  - relativized, 53
  - render, 54
  - replace\_child, 55
  - to\_fallback\_value, 55
  - type, 52
  - value\_type, 55
  - value\_type\_name, 55
  - with\_fallback, 56
  - with\_origin, 57
- hocon::config\_delayed\_merge, 57
  - at\_key, 60
  - at\_path, 60
  - has\_descendant, 60
  - origin, 61
  - relativized, 61
  - render, 61, 62
  - replace\_child, 62
  - to\_fallback\_value, 62
  - type, 59
  - value\_type, 63
  - value\_type\_name, 63
  - with\_fallback, 63
  - with\_origin, 64
- hocon::config\_delayed\_merge\_object, 65
  - at\_key, 68
  - at\_path, 68
  - attempt\_peek\_with\_partial\_resolve, 68
  - has\_descendant, 69
  - key\_set, 69
  - origin, 69
  - relativized, 70
  - render, 70
  - replace\_child, 71
  - to\_config, 71
  - to\_fallback\_value, 71
  - type, 67
  - value\_type, 72
  - value\_type\_name, 72
  - with\_fallback, 72
  - with\_origin, 73
- hocon::config\_document, 74
  - has\_path, 75
  - render, 75

- with\_value, 75
  - with\_value\_text, 76
  - without\_path, 77
- hocon::config\_document\_parser::parse\_context, 205
  - parse\_single\_value, 205
- hocon::config\_double, 77
  - at\_key, 80
  - at\_path, 80
  - origin, 81
  - relativized, 81
  - render, 81, 82
  - to\_fallback\_value, 82
  - type, 79
  - value\_type, 82
  - value\_type\_name, 83
  - with\_fallback, 83
  - with\_origin, 84
- hocon::config\_exception, 84
- hocon::config\_include\_context, 85
  - parse\_options, 86
  - relative\_to, 86
- hocon::config\_includer, 87
  - include, 87
  - with\_fallback, 88
- hocon::config\_includer\_file, 89
  - include\_file, 89
- hocon::config\_int, 90
  - at\_key, 92
  - at\_path, 93
  - origin, 93
  - relativized, 93
  - render, 94
  - to\_fallback\_value, 95
  - type, 92
  - value\_type, 95
  - value\_type\_name, 95
  - with\_fallback, 95
  - with\_origin, 96
- hocon::config\_list, 97
  - at\_key, 100
  - at\_path, 100
  - origin, 101
  - relativized, 101
  - render, 101, 102
  - to\_fallback\_value, 102
  - type, 100
  - value\_type, 103
  - value\_type\_name, 103
  - with\_fallback, 103
  - with\_origin, 104
- hocon::config\_long, 105
  - at\_key, 107
  - at\_path, 108
  - origin, 108
  - relativized, 108
  - render, 110
  - to\_fallback\_value, 111
  - type, 107
  - value\_type, 111
  - value\_type\_name, 111
  - with\_fallback, 111
  - with\_origin, 112
- hocon::config\_mergeable, 113
  - to\_fallback\_value, 114
  - with\_fallback, 114
- hocon::config\_node, 115
  - render, 115
- hocon::config\_node\_array, 116
  - render, 117
- hocon::config\_node\_comment, 117
  - render, 118
- hocon::config\_node\_complex\_value, 118
  - render, 119
- hocon::config\_node\_concatenation, 119
  - render, 120
- hocon::config\_node\_field, 120
  - render, 121
- hocon::config\_node\_include, 121
  - render, 122
- hocon::config\_node\_object, 122
  - render, 123
- hocon::config\_node\_path, 124
  - render, 124
- hocon::config\_node\_root, 125
  - render, 125
- hocon::config\_node\_simple\_value, 126
  - render, 126
- hocon::config\_node\_single\_token, 127
  - render, 127
- hocon::config\_null, 128
  - at\_key, 130
  - at\_path, 130
  - origin, 131
  - relativized, 131
  - render, 131, 132
  - to\_fallback\_value, 132
  - type, 129
  - value\_type, 132
  - value\_type\_name, 133
  - with\_fallback, 133
  - with\_origin, 134
- hocon::config\_number, 134
  - at\_key, 137
  - at\_path, 137
  - origin, 138
  - relativized, 138
  - render, 138, 139
  - to\_fallback\_value, 139
  - type, 136
  - value\_type, 139
  - value\_type\_name, 140
  - with\_fallback, 140
  - with\_origin, 141
- hocon::config\_object, 141
  - at\_key, 144
  - at\_path, 144

- attempt\_peek\_with\_partial\_resolve, 145
- key\_set, 145
- origin, 145
- relativized, 146
- render, 146
- to\_config, 147
- to\_fallback\_value, 147
- type, 144
- value\_type, 147
- value\_type\_name, 148
- with\_fallback, 148
- with\_origin, 149
- hocon::config\_origin, 149
  - comments, 150
  - description, 150
  - line\_number, 151
  - with\_comments, 151
  - with\_line\_number, 152
- hocon::config\_parse\_options, 153
  - append\_includer, 154
  - config\_parse\_options, 154
  - defaults, 154
  - get\_allow\_missing, 154
  - get\_includer, 155
  - get\_origin\_description, 155
  - get\_syntax, 155
  - prepend\_includer, 155
  - set\_allow\_missing, 156
  - set\_includer, 156
  - set\_origin\_description, 156
  - set\_syntax, 157
- hocon::config\_parseable, 157
  - options, 158
  - origin, 158
  - parse, 159
- hocon::config\_parser::parse\_context, 206
- hocon::config\_reference, 159
  - at\_key, 161
  - at\_path, 162
  - origin, 162
  - relativized, 162
  - render, 164
  - to\_fallback\_value, 165
  - type, 161
  - value\_type, 165
  - value\_type\_name, 165
  - with\_fallback, 165
  - with\_origin, 166
- hocon::config\_render\_options, 167
  - concise, 168
  - config\_render\_options, 168
  - get\_comments, 168
  - get\_formatted, 169
  - get\_json, 169
  - get\_origin\_comments, 169
  - set\_comments, 169
  - set\_formatted, 170
  - set\_json, 170
  - set\_origin\_comments, 170
- hocon::config\_resolve\_options, 171
  - config\_resolve\_options, 172
  - get\_allow\_unresolved, 172
  - get\_use\_system\_environment, 173
  - set\_allow\_unresolved, 173
  - set\_use\_system\_environment, 173
- hocon::config\_string, 174
  - at\_key, 176
  - at\_path, 177
  - origin, 177
  - relativized, 177
  - render, 178
  - to\_fallback\_value, 179
  - type, 176
  - value\_type, 179
  - value\_type\_name, 179
  - with\_fallback, 179
  - with\_origin, 180
- hocon::config\_value, 181
  - at\_key, 184
  - at\_path, 184
  - origin, 185
  - relativized, 185
  - render, 185, 186
  - to\_fallback\_value, 186
  - type, 184
  - value\_type, 187
  - value\_type\_name, 187
  - with\_fallback, 187
  - with\_origin, 188
- hocon::config\_value::modifier, 201
- hocon::config\_value::no\_exceptions\_modifier, 202
- hocon::config\_value\_factory, 189
  - from\_any\_ref, 189
- hocon::container, 190
  - has\_descendant, 190
  - replace\_child, 191
- hocon::default\_transformer, 191
- hocon::double\_slash\_comment, 192
- hocon::file\_name\_source, 192
- hocon::full\_includer, 193
  - include, 193
  - include\_file, 194
  - with\_fallback, 194
- hocon::generic\_exception, 196
- hocon::hash\_comment, 196
- hocon::ignored\_whitespace, 197
- hocon::io\_exception, 197
- hocon::iterator, 198
- hocon::iterator\_wrapper< iter >, 198
- hocon::line, 199
- hocon::missing\_exception, 200
- hocon::name\_source, 201
- hocon::not\_possible\_to\_resolve\_exception, 203
- hocon::not\_resolved\_exception, 203
- hocon::null\_exception, 204
- hocon::parse\_exception, 206



- hocon::parseable, 207
    - options, 208
    - origin, 208
    - parse, 208
  - hocon::parseable\_file, 209
    - options, 210
    - origin, 210
    - parse, 210
  - hocon::parseable\_not\_found, 211
    - options, 212
    - origin, 212
    - parse, 212
  - hocon::parseable\_resources, 213
    - options, 214
    - origin, 214
    - parse, 214
  - hocon::parseable\_string, 215
    - options, 216
    - origin, 216
    - parse, 216
  - hocon::path, 217
    - has\_funky\_chars, 218
    - parent, 218
    - remainder, 218
    - render, 218
    - to\_string, 218
  - hocon::path\_builder, 219
    - result, 219
  - hocon::path\_parser, 219
  - hocon::problem, 220
  - hocon::problem\_exception, 220
  - hocon::relative\_name\_source, 221
  - hocon::replaceable\_merge\_stack, 221
    - has\_descendant, 222
    - replace\_child, 222
  - hocon::resolve\_context, 223
  - hocon::resolve\_result< V >, 223
  - hocon::resolve\_source, 224
  - hocon::resolve\_source::result\_with\_path, 224
  - hocon::simple\_config\_document, 225
    - has\_path, 225
    - render, 226
    - with\_value, 226
    - with\_value\_text, 226
    - without\_path, 227
  - hocon::simple\_config\_list, 228
    - at\_key, 230
    - at\_path, 230
    - has\_descendant, 231
    - origin, 231
    - relativized, 231
    - render, 232
    - replace\_child, 233
    - to\_fallback\_value, 233
    - type, 230
    - value\_type, 233
    - value\_type\_name, 233
    - with\_fallback, 234
    - with\_origin, 235
  - hocon::simple\_config\_object, 235
    - at\_key, 238
    - at\_path, 239
    - attempt\_peek\_with\_partial\_resolve, 239
    - has\_descendant, 240
    - key\_set, 240
    - origin, 240
    - relativized, 240
    - render, 241
    - replace\_child, 242
    - to\_config, 242
    - to\_fallback\_value, 242
    - type, 238
    - value\_set, 242
    - value\_type, 243
    - value\_type\_name, 243
    - with\_fallback, 243
    - with\_only\_path\_or\_null, 244
    - with\_origin, 245
  - hocon::simple\_config\_origin, 245
    - comments, 246
    - description, 247
    - line\_number, 247
    - simple\_config\_origin, 246
    - with\_comments, 247
    - with\_line\_number, 248
  - hocon::simple\_include\_context, 249
    - parse\_options, 249
    - relative\_to, 249
  - hocon::simple\_includer, 250
    - include, 251
    - include\_file, 251
    - with\_fallback, 252
  - hocon::single\_token\_iterator, 252
  - hocon::substitution, 253
  - hocon::substitution\_expression, 254
  - hocon::token, 254
  - hocon::token\_iterator, 255
  - hocon::token\_list\_iterator, 255
  - hocon::tokens, 256
    - start\_token, 256
  - hocon::unmergeable, 257
  - hocon::unquoted\_text, 257
  - hocon::unresolved\_substitution\_exception, 258
  - hocon::unsupported\_exception, 259
  - hocon::validation\_failed\_exception, 259
  - hocon::validation\_problem, 260
  - hocon::value, 260
  - hocon::wrong\_type\_exception, 261
- include
    - hocon::config\_includer, 87
    - hocon::full\_includer, 193
    - hocon::simple\_includer, 251
  - include\_file
    - hocon::config\_includer\_file, 89
    - hocon::full\_includer, 194
    - hocon::simple\_includer, 251

- is\_empty
  - hocon::config, 34
- is\_resolved
  - hocon::config, 35
- key\_set
  - hocon::config\_delayed\_merge\_object, 69
  - hocon::config\_object, 145
  - hocon::simple\_config\_object, 240
- line\_number
  - hocon::config\_origin, 151
  - hocon::simple\_config\_origin, 247
- List< T >, 200
- operator==
  - hocon, 19
- options
  - hocon::config\_parseable, 158
  - hocon::parseable, 208
  - hocon::parseable\_file, 210
  - hocon::parseable\_not\_found, 212
  - hocon::parseable\_resources, 214
  - hocon::parseable\_string, 216
- origin
  - hocon::config, 35
  - hocon::config\_boolean, 45
  - hocon::config\_concatenation, 53
  - hocon::config\_delayed\_merge, 61
  - hocon::config\_delayed\_merge\_object, 69
  - hocon::config\_double, 81
  - hocon::config\_int, 93
  - hocon::config\_list, 101
  - hocon::config\_long, 108
  - hocon::config\_null, 131
  - hocon::config\_number, 138
  - hocon::config\_object, 145
  - hocon::config\_parseable, 158
  - hocon::config\_reference, 162
  - hocon::config\_string, 177
  - hocon::config\_value, 185
  - hocon::parseable, 208
  - hocon::parseable\_file, 210
  - hocon::parseable\_not\_found, 212
  - hocon::parseable\_resources, 214
  - hocon::parseable\_string, 216
  - hocon::simple\_config\_list, 231
  - hocon::simple\_config\_object, 240
- OutListIter< T >, 205
- parent
  - hocon::path, 218
- parse
  - hocon::config\_parseable, 159
  - hocon::parseable, 208
  - hocon::parseable\_file, 210
  - hocon::parseable\_not\_found, 212
  - hocon::parseable\_resources, 214
  - hocon::parseable\_string, 216
- parse\_file\_any\_syntax
  - hocon::config, 35, 36
- parse\_options
  - hocon::config\_include\_context, 86
  - hocon::simple\_include\_context, 249
- parse\_single\_value
  - hocon::config\_document\_parser::parse\_context, 205
- parse\_string
  - hocon::config, 36, 37
- prepend\_includer
  - hocon::config\_parse\_options, 155
- relative\_to
  - hocon::config\_include\_context, 86
  - hocon::simple\_include\_context, 249
- relativized
  - hocon::config\_boolean, 45
  - hocon::config\_concatenation, 53
  - hocon::config\_delayed\_merge, 61
  - hocon::config\_delayed\_merge\_object, 70
  - hocon::config\_double, 81
  - hocon::config\_int, 93
  - hocon::config\_list, 101
  - hocon::config\_long, 108
  - hocon::config\_null, 131
  - hocon::config\_number, 138
  - hocon::config\_object, 146
  - hocon::config\_reference, 162
  - hocon::config\_string, 177
  - hocon::config\_value, 185
  - hocon::simple\_config\_list, 231
  - hocon::simple\_config\_object, 240
- remainder
  - hocon::path, 218
- render
  - hocon::abstract\_config\_node, 21
  - hocon::abstract\_config\_node\_value, 22
  - hocon::config\_boolean, 47
  - hocon::config\_concatenation, 54
  - hocon::config\_delayed\_merge, 61, 62
  - hocon::config\_delayed\_merge\_object, 70
  - hocon::config\_document, 75
  - hocon::config\_double, 81, 82
  - hocon::config\_int, 94
  - hocon::config\_list, 101, 102
  - hocon::config\_long, 110
  - hocon::config\_node, 115
  - hocon::config\_node\_array, 117
  - hocon::config\_node\_comment, 118
  - hocon::config\_node\_complex\_value, 119
  - hocon::config\_node\_concatenation, 120
  - hocon::config\_node\_field, 121
  - hocon::config\_node\_include, 122
  - hocon::config\_node\_object, 123
  - hocon::config\_node\_path, 124
  - hocon::config\_node\_root, 125
  - hocon::config\_node\_simple\_value, 126
  - hocon::config\_node\_single\_token, 127

- hocon::config\_null, [131](#), [132](#)
- hocon::config\_number, [138](#), [139](#)
- hocon::config\_object, [146](#)
- hocon::config\_reference, [164](#)
- hocon::config\_string, [178](#)
- hocon::config\_value, [185](#), [186](#)
- hocon::path, [218](#)
- hocon::simple\_config\_document, [226](#)
- hocon::simple\_config\_list, [232](#)
- hocon::simple\_config\_object, [241](#)
- replace\_child
  - hocon::config\_concatenation, [55](#)
  - hocon::config\_delayed\_merge, [62](#)
  - hocon::config\_delayed\_merge\_object, [71](#)
  - hocon::container, [191](#)
  - hocon::replaceable\_merge\_stack, [222](#)
  - hocon::simple\_config\_list, [233](#)
  - hocon::simple\_config\_object, [242](#)
- resolve
  - hocon::config, [37](#), [38](#)
- resolve\_with
  - hocon::config, [38](#), [39](#)
- result
  - hocon::path\_builder, [219](#)
- root
  - hocon::config, [39](#)
- set\_allow\_missing
  - hocon::config\_parse\_options, [156](#)
- set\_allow\_unresolved
  - hocon::config\_resolve\_options, [173](#)
- set\_comments
  - hocon::config\_render\_options, [169](#)
- set\_formatted
  - hocon::config\_render\_options, [170](#)
- set\_includer
  - hocon::config\_parse\_options, [156](#)
- set\_json
  - hocon::config\_render\_options, [170](#)
- set\_origin\_comments
  - hocon::config\_render\_options, [170](#)
- set\_origin\_description
  - hocon::config\_parse\_options, [156](#)
- set\_syntax
  - hocon::config\_parse\_options, [157](#)
- set\_use\_system\_environment
  - hocon::config\_resolve\_options, [173](#)
- simple\_config\_origin
  - hocon::simple\_config\_origin, [246](#)
- start\_token
  - hocon::tokens, [256](#)
- to\_config
  - hocon::config\_delayed\_merge\_object, [71](#)
  - hocon::config\_object, [147](#)
  - hocon::simple\_config\_object, [242](#)
- to\_fallback\_value
  - hocon::config, [39](#)
  - hocon::config\_boolean, [48](#)
- hocon::config\_concatenation, [55](#)
- hocon::config\_delayed\_merge, [62](#)
- hocon::config\_delayed\_merge\_object, [71](#)
- hocon::config\_double, [82](#)
- hocon::config\_int, [95](#)
- hocon::config\_list, [102](#)
- hocon::config\_long, [111](#)
- hocon::config\_mergeable, [114](#)
- hocon::config\_null, [132](#)
- hocon::config\_number, [139](#)
- hocon::config\_object, [147](#)
- hocon::config\_reference, [165](#)
- hocon::config\_string, [179](#)
- hocon::config\_value, [186](#)
- hocon::simple\_config\_list, [233](#)
- hocon::simple\_config\_object, [242](#)
- to\_string
  - hocon::path, [218](#)
- type
  - hocon::config\_boolean, [44](#)
  - hocon::config\_concatenation, [52](#)
  - hocon::config\_delayed\_merge, [59](#)
  - hocon::config\_delayed\_merge\_object, [67](#)
  - hocon::config\_double, [79](#)
  - hocon::config\_int, [92](#)
  - hocon::config\_list, [100](#)
  - hocon::config\_long, [107](#)
  - hocon::config\_null, [129](#)
  - hocon::config\_number, [136](#)
  - hocon::config\_object, [144](#)
  - hocon::config\_reference, [161](#)
  - hocon::config\_string, [176](#)
  - hocon::config\_value, [184](#)
  - hocon::simple\_config\_list, [230](#)
  - hocon::simple\_config\_object, [238](#)
- value\_set
  - hocon::simple\_config\_object, [242](#)
- value\_type
  - hocon::config\_boolean, [48](#)
  - hocon::config\_concatenation, [55](#)
  - hocon::config\_delayed\_merge, [63](#)
  - hocon::config\_delayed\_merge\_object, [72](#)
  - hocon::config\_double, [82](#)
  - hocon::config\_int, [95](#)
  - hocon::config\_list, [103](#)
  - hocon::config\_long, [111](#)
  - hocon::config\_null, [132](#)
  - hocon::config\_number, [139](#)
  - hocon::config\_object, [147](#)
  - hocon::config\_reference, [165](#)
  - hocon::config\_string, [179](#)
  - hocon::config\_value, [187](#)
  - hocon::simple\_config\_list, [233](#)
  - hocon::simple\_config\_object, [243](#)
- value\_type\_name
  - hocon::config\_boolean, [48](#)
  - hocon::config\_concatenation, [55](#)
  - hocon::config\_delayed\_merge, [63](#)

- hocon::config\_delayed\_merge\_object, 72
- hocon::config\_double, 83
- hocon::config\_int, 95
- hocon::config\_list, 103
- hocon::config\_long, 111
- hocon::config\_null, 133
- hocon::config\_number, 140
- hocon::config\_object, 148
- hocon::config\_reference, 165
- hocon::config\_string, 179
- hocon::config\_value, 187
- hocon::simple\_config\_list, 233
- hocon::simple\_config\_object, 243
- version.h
  - CPP\_HOCON\_VERSION, 263
  - CPP\_HOCON\_VERSION\_MAJOR, 263
  - CPP\_HOCON\_VERSION\_MINOR, 264
  - CPP\_HOCON\_VERSION\_PATCH, 264
  - CPP\_HOCON\_VERSION\_WITH\_COMMIT, 264
- with\_comments
  - hocon::config\_origin, 151
  - hocon::simple\_config\_origin, 247
- with\_fallback
  - hocon::config, 40
  - hocon::config\_boolean, 48
  - hocon::config\_concatenation, 56
  - hocon::config\_delayed\_merge, 63
  - hocon::config\_delayed\_merge\_object, 72
  - hocon::config\_double, 83
  - hocon::config\_includer, 88
  - hocon::config\_int, 95
  - hocon::config\_list, 103
  - hocon::config\_long, 111
  - hocon::config\_mergeable, 114
  - hocon::config\_null, 133
  - hocon::config\_number, 140
  - hocon::config\_object, 148
  - hocon::config\_reference, 165
  - hocon::config\_string, 179
  - hocon::config\_value, 187
  - hocon::full\_includer, 194
  - hocon::simple\_config\_list, 234
  - hocon::simple\_config\_object, 243
  - hocon::simple\_includer, 252
- with\_line\_number
  - hocon::config\_origin, 152
  - hocon::simple\_config\_origin, 248
- with\_only\_path
  - hocon::config, 41
- with\_only\_path\_or\_null
  - hocon::simple\_config\_object, 244
- with\_origin
  - hocon::config\_boolean, 49
  - hocon::config\_concatenation, 57
  - hocon::config\_delayed\_merge, 64
  - hocon::config\_delayed\_merge\_object, 73
  - hocon::config\_double, 84
  - hocon::config\_int, 96
  - hocon::config\_list, 104
  - hocon::config\_long, 112
  - hocon::config\_null, 134
  - hocon::config\_number, 141
  - hocon::config\_object, 149
  - hocon::config\_reference, 166
  - hocon::config\_string, 180
  - hocon::config\_value, 188
  - hocon::simple\_config\_list, 235
  - hocon::simple\_config\_object, 245
- with\_value
  - hocon::config, 41
  - hocon::config\_document, 75
  - hocon::simple\_config\_document, 226
- with\_value\_text
  - hocon::config\_document, 76
  - hocon::simple\_config\_document, 226
- without\_path
  - hocon::config, 42
  - hocon::config\_document, 77
  - hocon::simple\_config\_document, 227