

cpp-hocon

0.3.0

Generated on Wed Jul 17 2024 00:00:00 for cpp-hocon by Doxygen 1.12.0

Wed Jul 17 2024 00:00:00

1 C++ HOCON Parser	1
1.1 Caveats	1
1.2 Build Requirements	1
1.3 Pre-Build	2
1.4 Building	2
1.5 Testing	2
2 Namespace Index	3
2.1 Namespace List	3
3 Hierarchical Index	5
3.1 Class Hierarchy	5
4 Class Index	9
4.1 Class List	9
5 File Index	13
5.1 File List	13
6 Namespace Documentation	15
6.1 hocon Namespace Reference	15
6.1.1 Detailed Description	19
6.1.2 Typedef Documentation	19
6.1.2.1 duration	19
6.1.2.2 shared_config	19
6.1.2.3 shared_container	19
6.1.2.4 shared_include_context	19
6.1.2.5 shared_includer	19
6.1.2.6 shared_list	19
6.1.2.7 shared_node	20
6.1.2.8 shared_node_array	20
6.1.2.9 shared_node_concatenation	20
6.1.2.10 shared_node_list	20
6.1.2.11 shared_node_object	20
6.1.2.12 shared_node_value	20
6.1.2.13 shared_object	20
6.1.2.14 shared_origin	20
6.1.2.15 shared_parseable	21
6.1.2.16 shared_string	21
6.1.2.17 shared_token	21
6.1.2.18 shared_value	21
6.1.2.19 token_list	21
6.1.2.20 unwrapped_value	21
6.1.3 Enumeration Type Documentation	21

6.1.3.1 config_include_kind	21
6.1.3.2 config_string_type	22
6.1.3.3 origin_type	22
6.1.3.4 resolve_status	22
6.1.3.5 time_unit	22
6.1.3.6 token_type	22
6.1.4 Function Documentation	22
6.1.4.1 make_resolve_result()	22
6.1.4.2 operator==()	22
6.2 std Namespace Reference	23
6.2.1 Detailed Description	27
6.2.2 Function Documentation	27
6.2.2.1 begin()	27
6.2.2.2 end()	27
7 Class Documentation	29
7.1 hocon::abstract_config_node Class Reference	29
7.1.1 Detailed Description	29
7.1.2 Member Function Documentation	29
7.1.2.1 render()	29
7.2 hocon::abstract_config_node_value Class Reference	30
7.2.1 Detailed Description	30
7.2.2 Member Function Documentation	30
7.2.2.1 render()	30
7.3 hocon::bad_path_exception Struct Reference	31
7.3.1 Detailed Description	31
7.3.2 Constructor & Destructor Documentation	31
7.3.2.1 bad_path_exception() [1/2]	31
7.3.2.2 bad_path_exception() [2/2]	31
7.4 hocon::bad_value_exception Struct Reference	32
7.4.1 Detailed Description	32
7.4.2 Constructor & Destructor Documentation	32
7.4.2.1 bad_value_exception() [1/2]	32
7.4.2.2 bad_value_exception() [2/2]	32
7.5 hocon::bug_or_broken_exception Struct Reference	33
7.5.1 Detailed Description	33
7.5.2 Constructor & Destructor Documentation	33
7.5.2.1 bug_or_broken_exception()	33
7.6 hocon::comment Class Reference	34
7.6.1 Detailed Description	34
7.6.2 Member Function Documentation	34
7.6.2.1 operator==()	34

7.6.2.2 to_string()	34
7.7 hocon::config Class Reference	35
7.7.1 Detailed Description	37
7.7.2 Member Function Documentation	38
7.7.2.1 at_key()	38
7.7.2.2 at_path()	39
7.7.2.3 check_valid()	39
7.7.2.4 entry_set()	40
7.7.2.5 get_duration()	40
7.7.2.6 get_homogeneous_unwrapped_list()	41
7.7.2.7 get_is_null()	41
7.7.2.8 has_path()	41
7.7.2.9 has_path_or_null()	42
7.7.2.10 is_empty()	42
7.7.2.11 is_resolved()	43
7.7.2.12 origin()	43
7.7.2.13 parse_file_any_syntax() [1/2]	43
7.7.2.14 parse_file_any_syntax() [2/2]	43
7.7.2.15 parse_string() [1/2]	44
7.7.2.16 parse_string() [2/2]	44
7.7.2.17 resolve() [1/2]	45
7.7.2.18 resolve() [2/2]	45
7.7.2.19 resolve_with() [1/2]	46
7.7.2.20 resolve_with() [2/2]	46
7.7.2.21 root()	46
7.7.2.22 to_fallback_value()	47
7.7.2.23 with_fallback()	47
7.7.2.24 with_only_path()	47
7.7.2.25 with_value()	48
7.7.2.26 without_path()	48
7.7.3 Friends And Related Symbol Documentation	49
7.7.3.1 config_object	49
7.7.3.2 config_parseable	49
7.7.3.3 config_value	49
7.7.3.4 parseable	49
7.8 hocon::config_boolean Class Reference	49
7.8.1 Detailed Description	51
7.8.2 Member Enumeration Documentation	51
7.8.2.1 type	51
7.8.3 Member Function Documentation	51
7.8.3.1 at_key()	51
7.8.3.2 at_path()	52

7.8.3.3 equals()	53
7.8.3.4 new_copy()	53
7.8.3.5 operator==()	53
7.8.3.6 origin()	53
7.8.3.7 relativized()	53
7.8.3.8 render() [1/2]	54
7.8.3.9 render() [2/2]	54
7.8.3.10 to_fallback_value()	55
7.8.3.11 transform_to_string()	55
7.8.3.12 type_name()	55
7.8.3.13 unwrapped()	55
7.8.3.14 value_type()	55
7.8.3.15 value_type_name()	55
7.8.3.16 with_fallback()	56
7.8.3.17 with_origin()	56
7.9 hocon::config_concatenation Class Reference	57
7.9.1 Detailed Description	59
7.9.2 Member Enumeration Documentation	59
7.9.2.1 type	59
7.9.3 Member Function Documentation	59
7.9.3.1 at_key()	59
7.9.3.2 at_path()	59
7.9.3.3 equals()	60
7.9.3.4 get_resolve_status()	60
7.9.3.5 has_descendant()	60
7.9.3.6 ignores_fallbacks()	60
7.9.3.7 new_copy()	60
7.9.3.8 operator==()	61
7.9.3.9 origin()	61
7.9.3.10 relativized()	61
7.9.3.11 render() [1/3]	61
7.9.3.12 render() [2/3]	62
7.9.3.13 render() [3/3]	62
7.9.3.14 replace_child()	62
7.9.3.15 resolve_substitutions()	63
7.9.3.16 to_fallback_value()	63
7.9.3.17 type_name()	63
7.9.3.18 unmerged_values()	63
7.9.3.19 unwrapped()	63
7.9.3.20 value_type()	63
7.9.3.21 value_type_name()	64
7.9.3.22 with_fallback()	64

7.9.3.23 with_origin()	64
7.10 hocon::config_delayed_merge Class Reference	65
7.10.1 Detailed Description	67
7.10.2 Member Enumeration Documentation	67
7.10.2.1 type	67
7.10.3 Member Function Documentation	67
7.10.3.1 at_key()	67
7.10.3.2 at_path()	67
7.10.3.3 equals()	68
7.10.3.4 get_resolve_status()	68
7.10.3.5 has_descendant()	68
7.10.3.6 ignores_fallbacks()	68
7.10.3.7 make_replacement()	68
7.10.3.8 new_copy()	69
7.10.3.9 operator==()	69
7.10.3.10 origin()	69
7.10.3.11 relativized()	69
7.10.3.12 render() [1/4]	70
7.10.3.13 render() [2/4]	70
7.10.3.14 render() [3/4]	70
7.10.3.15 render() [4/4]	70
7.10.3.16 replace_child()	71
7.10.3.17 resolve_substitutions()	71
7.10.3.18 to_fallback_value()	71
7.10.3.19 type_name()	71
7.10.3.20 unmerged_values()	72
7.10.3.21 unwrapped()	72
7.10.3.22 value_type()	72
7.10.3.23 value_type_name()	72
7.10.3.24 with_fallback()	73
7.10.3.25 with_origin()	73
7.11 hocon::config_delayed_merge_object Class Reference	74
7.11.1 Detailed Description	76
7.11.2 Member Typedef Documentation	76
7.11.2.1 iterator	76
7.11.3 Member Enumeration Documentation	76
7.11.3.1 type	76
7.11.4 Member Function Documentation	77
7.11.4.1 at_key()	77
7.11.4.2 at_path()	77
7.11.4.3 attempt_peek_with_partial_resolve()	77
7.11.4.4 begin()	78

7.11.4.5 construct_delayed_merge()	78
7.11.4.6 end()	78
7.11.4.7 entry_set()	78
7.11.4.8 equals()	78
7.11.4.9 get()	79
7.11.4.10 get_resolve_status()	79
7.11.4.11 has_descendant()	79
7.11.4.12 ignores_fallbacks()	79
7.11.4.13 is_empty()	79
7.11.4.14 key_set()	79
7.11.4.15 make_replacement()	80
7.11.4.16 new_copy() [1/2]	80
7.11.4.17 new_copy() [2/2]	80
7.11.4.18 operator==()	80
7.11.4.19 operator[]()	80
7.11.4.20 origin()	80
7.11.4.21 relativized()	80
7.11.4.22 render() [1/4]	81
7.11.4.23 render() [2/4]	81
7.11.4.24 render() [3/4]	81
7.11.4.25 render() [4/4]	82
7.11.4.26 replace_child()	82
7.11.4.27 resolve_substitutions()	82
7.11.4.28 size()	82
7.11.4.29 to_config()	83
7.11.4.30 to_fallback_value()	83
7.11.4.31 type_name()	83
7.11.4.32 unmerged_values()	83
7.11.4.33 unwrapped()	83
7.11.4.34 value_type()	84
7.11.4.35 value_type_name()	84
7.11.4.36 with_fallback()	84
7.11.4.37 with_only_path()	85
7.11.4.38 with_only_path_or_null()	85
7.11.4.39 with_origin()	85
7.11.4.40 with_value() [1/2]	85
7.11.4.41 with_value() [2/2]	86
7.11.4.42 without_path()	86
7.12 hocon::config_document Class Reference	86
7.12.1 Detailed Description	87
7.12.2 Member Function Documentation	87
7.12.2.1 has_path()	87

7.12.2.2 render()	87
7.12.2.3 with_value()	87
7.12.2.4 with_value_text()	88
7.12.2.5 without_path()	88
7.13 hocon::config_double Class Reference	89
7.13.1 Detailed Description	91
7.13.2 Member Enumeration Documentation	91
7.13.2.1 type	91
7.13.3 Member Function Documentation	91
7.13.3.1 at_key()	91
7.13.3.2 at_path()	91
7.13.3.3 double_value()	92
7.13.3.4 equals()	92
7.13.3.5 long_value()	92
7.13.3.6 new_copy()	92
7.13.3.7 operator==()	92
7.13.3.8 origin()	93
7.13.3.9 relativized()	93
7.13.3.10 render() [1/2]	93
7.13.3.11 render() [2/2]	94
7.13.3.12 to_fallback_value()	94
7.13.3.13 transform_to_string()	94
7.13.3.14 type_name()	94
7.13.3.15 unwrapped()	95
7.13.3.16 value_type()	95
7.13.3.17 value_type_name()	95
7.13.3.18 with_fallback()	95
7.13.3.19 with_origin()	96
7.13.4 Member Data Documentation	96
7.13.4.1 _original_text	96
7.14 hocon::config_exception Struct Reference	97
7.14.1 Detailed Description	97
7.14.2 Constructor & Destructor Documentation	98
7.14.2.1 config_exception() [1/3]	98
7.14.2.2 config_exception() [2/3]	98
7.14.2.3 config_exception() [3/3]	98
7.15 hocon::config_include_context Class Reference	98
7.15.1 Detailed Description	99
7.15.2 Constructor & Destructor Documentation	99
7.15.2.1 config_include_context()	99
7.15.3 Member Function Documentation	99
7.15.3.1 get_cur_dir()	99

7.15.3.2 parse_options()	100
7.15.3.3 relative_to()	100
7.15.3.4 set_cur_dir()	100
7.15.4 Member Data Documentation	101
7.15.4.1 _cur_dir	101
7.16 hocon::config_includer Class Reference	101
7.16.1 Detailed Description	101
7.16.2 Member Function Documentation	101
7.16.2.1 include()	101
7.16.2.2 with_fallback()	102
7.17 hocon::config_includer_file Class Reference	102
7.17.1 Detailed Description	103
7.17.2 Member Function Documentation	103
7.17.2.1 include_file()	103
7.18 hocon::config_int Class Reference	104
7.18.1 Detailed Description	106
7.18.2 Member Enumeration Documentation	106
7.18.2.1 type	106
7.18.3 Member Function Documentation	106
7.18.3.1 at_key()	106
7.18.3.2 at_path()	106
7.18.3.3 double_value()	107
7.18.3.4 equals()	107
7.18.3.5 long_value()	107
7.18.3.6 new_copy()	107
7.18.3.7 operator==()	107
7.18.3.8 origin()	108
7.18.3.9 relativized()	108
7.18.3.10 render() [1/2]	108
7.18.3.11 render() [2/2]	109
7.18.3.12 to_fallback_value()	109
7.18.3.13 transform_to_string()	109
7.18.3.14 type_name()	109
7.18.3.15 unwrapped()	110
7.18.3.16 value_type()	110
7.18.3.17 value_type_name()	110
7.18.3.18 with_fallback()	110
7.18.3.19 with_origin()	111
7.18.4 Member Data Documentation	111
7.18.4.1 _original_text	111
7.19 hocon::config_list Class Reference	111
7.19.1 Detailed Description	113

7.19.2 Member Typedef Documentation	114
7.19.2.1 iterator	114
7.19.3 Member Enumeration Documentation	114
7.19.3.1 type	114
7.19.4 Constructor & Destructor Documentation	114
7.19.4.1 config_list()	114
7.19.5 Member Function Documentation	114
7.19.5.1 at_key()	114
7.19.5.2 at_path()	114
7.19.5.3 equals()	115
7.19.5.4 origin()	115
7.19.5.5 relativized()	115
7.19.5.6 render() [1/2]	116
7.19.5.7 render() [2/2]	116
7.19.5.8 to_fallback_value()	116
7.19.5.9 type_name()	117
7.19.5.10 unwrapped()	117
7.19.5.11 value_type()	117
7.19.5.12 value_type_name()	117
7.19.5.13 with_fallback()	118
7.19.5.14 with_origin()	118
7.20 hocon::config_long Class Reference	119
7.20.1 Detailed Description	121
7.20.2 Member Enumeration Documentation	121
7.20.2.1 type	121
7.20.3 Member Function Documentation	121
7.20.3.1 at_key()	121
7.20.3.2 at_path()	121
7.20.3.3 double_value()	122
7.20.3.4 equals()	122
7.20.3.5 long_value()	122
7.20.3.6 new_copy()	122
7.20.3.7 operator==()	122
7.20.3.8 origin()	123
7.20.3.9 relativized()	123
7.20.3.10 render() [1/2]	123
7.20.3.11 render() [2/2]	124
7.20.3.12 to_fallback_value()	124
7.20.3.13 transform_to_string()	124
7.20.3.14 type_name()	124
7.20.3.15 unwrapped()	125
7.20.3.16 value_type()	125

7.20.3.17 value_type_name()	125
7.20.3.18 with_fallback()	125
7.20.3.19 with_origin()	126
7.20.4 Member Data Documentation	126
7.20.4.1 _original_text	126
7.21 hocon::config_mergeable Class Reference	127
7.21.1 Detailed Description	127
7.21.2 Member Function Documentation	128
7.21.2.1 to_fallback_value()	128
7.21.2.2 with_fallback()	128
7.21.3 Friends And Related Symbol Documentation	129
7.21.3.1 config_value	129
7.22 hocon::config_node Class Reference	129
7.22.1 Detailed Description	129
7.22.2 Member Function Documentation	130
7.22.2.1 render()	130
7.23 hocon::config_node_array Class Reference	130
7.23.1 Detailed Description	130
7.23.2 Member Function Documentation	131
7.23.2.1 get_tokens()	131
7.23.2.2 new_node()	131
7.23.2.3 render()	131
7.24 hocon::config_node_comment Class Reference	131
7.24.1 Detailed Description	132
7.24.2 Member Function Documentation	132
7.24.2.1 get_tokens()	132
7.24.2.2 render()	132
7.25 hocon::config_node_complex_value Class Reference	132
7.25.1 Detailed Description	133
7.25.2 Member Function Documentation	133
7.25.2.1 get_tokens()	133
7.25.2.2 render()	133
7.26 hocon::config_node_concatenation Class Reference	133
7.26.1 Detailed Description	134
7.26.2 Member Function Documentation	134
7.26.2.1 get_tokens()	134
7.26.2.2 new_node()	134
7.26.2.3 render()	134
7.27 hocon::config_node_field Class Reference	135
7.27.1 Detailed Description	135
7.27.2 Member Function Documentation	135
7.27.2.1 get_tokens()	135

7.27.2.2 render()	136
7.28 hocon::config_node_include Class Reference	136
7.28.1 Detailed Description	136
7.28.2 Member Function Documentation	137
7.28.2.1 get_tokens()	137
7.28.2.2 render()	137
7.29 hocon::config_node_object Class Reference	137
7.29.1 Detailed Description	138
7.29.2 Member Function Documentation	138
7.29.2.1 get_tokens()	138
7.29.2.2 new_node()	138
7.29.2.3 render()	139
7.30 hocon::config_node_path Class Reference	139
7.30.1 Detailed Description	139
7.30.2 Member Function Documentation	139
7.30.2.1 get_tokens()	139
7.30.2.2 render()	140
7.31 hocon::config_node_root Class Reference	140
7.31.1 Detailed Description	141
7.31.2 Member Function Documentation	141
7.31.2.1 get_tokens()	141
7.31.2.2 new_node()	141
7.31.2.3 render()	141
7.32 hocon::config_node_simple_value Class Reference	141
7.32.1 Detailed Description	142
7.32.2 Member Function Documentation	142
7.32.2.1 get_tokens()	142
7.32.2.2 render()	142
7.33 hocon::config_node_single_token Class Reference	142
7.33.1 Detailed Description	143
7.33.2 Member Function Documentation	143
7.33.2.1 get_tokens()	143
7.33.2.2 render()	143
7.34 hocon::config_null Class Reference	143
7.34.1 Detailed Description	145
7.34.2 Member Enumeration Documentation	145
7.34.2.1 type	145
7.34.3 Member Function Documentation	146
7.34.3.1 at_key()	146
7.34.3.2 at_path()	146
7.34.3.3 equals()	146
7.34.3.4 new_copy()	146

7.34.3.5 operator==()	147
7.34.3.6 origin()	147
7.34.3.7 relativized()	147
7.34.3.8 render() [1/3]	147
7.34.3.9 render() [2/3]	148
7.34.3.10 render() [3/3]	148
7.34.3.11 to_fallback_value()	148
7.34.3.12 transform_to_string()	149
7.34.3.13 type_name()	149
7.34.3.14 unwrapped()	149
7.34.3.15 value_type()	149
7.34.3.16 value_type_name()	149
7.34.3.17 with_fallback()	150
7.34.3.18 with_origin()	150
7.35 hocon::config_number Class Reference	151
7.35.1 Detailed Description	153
7.35.2 Member Enumeration Documentation	153
7.35.2.1 type	153
7.35.3 Member Function Documentation	153
7.35.3.1 at_key()	153
7.35.3.2 at_path()	153
7.35.3.3 equals()	154
7.35.3.4 operator==()	154
7.35.3.5 origin()	154
7.35.3.6 relativized()	154
7.35.3.7 render() [1/2]	155
7.35.3.8 render() [2/2]	155
7.35.3.9 to_fallback_value()	156
7.35.3.10 transform_to_string()	156
7.35.3.11 type_name()	156
7.35.3.12 value_type()	156
7.35.3.13 value_type_name()	156
7.35.3.14 with_fallback()	157
7.35.3.15 with_origin()	157
7.35.4 Member Data Documentation	158
7.35.4.1 _original_text	158
7.36 hocon::config_object Class Reference	158
7.36.1 Detailed Description	161
7.36.2 Member Typedef Documentation	161
7.36.2.1 iterator	161
7.36.3 Member Enumeration Documentation	161
7.36.3.1 type	161

7.36.4 Member Function Documentation	161
7.36.4.1 at_key()	161
7.36.4.2 at_path()	161
7.36.4.3 attempt_peek_with_partial_resolve()	162
7.36.4.4 construct_delayed_merge()	162
7.36.4.5 equals()	162
7.36.4.6 key_set()	163
7.36.4.7 new_copy()	163
7.36.4.8 origin()	163
7.36.4.9 relativized()	163
7.36.4.10 render() [1/2]	164
7.36.4.11 render() [2/2]	164
7.36.4.12 to_config()	164
7.36.4.13 to_fallback_value()	165
7.36.4.14 type_name()	165
7.36.4.15 value_type()	165
7.36.4.16 value_type_name()	165
7.36.4.17 with_fallback()	166
7.36.4.18 with_only_path_or_null()	166
7.36.4.19 with_origin()	166
7.36.5 Friends And Related Symbol Documentation	167
7.36.5.1 config	167
7.36.5.2 config_delayed_merge_object	167
7.36.5.3 config_value	167
7.36.5.4 resolve_source	167
7.36.5.5 simple_config_object	167
7.37 hocon::config_origin Class Reference	168
7.37.1 Detailed Description	168
7.37.2 Member Function Documentation	169
7.37.2.1 comments()	169
7.37.2.2 description()	169
7.37.2.3 line_number()	169
7.37.2.4 with_comments()	169
7.37.2.5 with_line_number()	170
7.38 hocon::config_parse_options Class Reference	170
7.38.1 Detailed Description	171
7.38.2 Constructor & Destructor Documentation	172
7.38.2.1 config_parse_options()	172
7.38.3 Member Function Documentation	172
7.38.3.1 append_includer()	172
7.38.3.2 defaults()	172
7.38.3.3 get_allow_missing()	173

7.38.3.4	get_includer()	173
7.38.3.5	get_origin_description()	173
7.38.3.6	get_syntax()	173
7.38.3.7	prepend_includer()	173
7.38.3.8	set_allow_missing()	174
7.38.3.9	set_includer()	174
7.38.3.10	set_origin_description()	174
7.38.3.11	set_syntax()	175
7.39	hocon::config_parseable Class Reference	175
7.39.1	Detailed Description	176
7.39.2	Member Function Documentation	176
7.39.2.1	options()	176
7.39.2.2	origin()	176
7.39.2.3	parse()	176
7.40	hocon::config_reference Class Reference	177
7.40.1	Detailed Description	179
7.40.2	Member Enumeration Documentation	179
7.40.2.1	type	179
7.40.3	Member Function Documentation	179
7.40.3.1	at_key()	179
7.40.3.2	at_path()	179
7.40.3.3	equals()	180
7.40.3.4	get_resolve_status()	180
7.40.3.5	ignores_fallbacks()	180
7.40.3.6	new_copy()	180
7.40.3.7	operator==()	180
7.40.3.8	origin()	180
7.40.3.9	relativized()	180
7.40.3.10	render() [1/3]	181
7.40.3.11	render() [2/3]	181
7.40.3.12	render() [3/3]	181
7.40.3.13	resolve_substitutions()	182
7.40.3.14	to_fallback_value()	182
7.40.3.15	type_name()	182
7.40.3.16	unmerged_values()	182
7.40.3.17	unwrapped()	182
7.40.3.18	value_type()	183
7.40.3.19	value_type_name()	183
7.40.3.20	with_fallback()	183
7.40.3.21	with_origin()	184
7.41	hocon::config_render_options Class Reference	184
7.41.1	Detailed Description	185

7.41.2 Constructor & Destructor Documentation	185
7.41.2.1 config_render_options()	185
7.41.3 Member Function Documentation	185
7.41.3.1 concise()	185
7.41.3.2 get_comments()	186
7.41.3.3 get_formatted()	186
7.41.3.4 get_json()	186
7.41.3.5 get_origin_comments()	186
7.41.3.6 set_comments()	186
7.41.3.7 set_formatted()	187
7.41.3.8 set_json()	187
7.41.3.9 set_origin_comments()	187
7.42 hocon::config_resolve_options Class Reference	188
7.42.1 Detailed Description	188
7.42.2 Constructor & Destructor Documentation	189
7.42.2.1 config_resolve_options()	189
7.42.3 Member Function Documentation	189
7.42.3.1 get_allow_unresolved()	189
7.42.3.2 get_use_system_environment()	189
7.42.3.3 set_allow_unresolved()	189
7.42.3.4 set_use_system_environment()	190
7.43 hocon::config_string Class Reference	190
7.43.1 Detailed Description	192
7.43.2 Member Enumeration Documentation	192
7.43.2.1 type	192
7.43.3 Member Function Documentation	192
7.43.3.1 at_key()	192
7.43.3.2 at_path()	193
7.43.3.3 equals()	194
7.43.3.4 new_copy()	194
7.43.3.5 operator==()	194
7.43.3.6 origin()	194
7.43.3.7 relativized()	194
7.43.3.8 render() [1/3]	195
7.43.3.9 render() [2/3]	195
7.43.3.10 render() [3/3]	195
7.43.3.11 to_fallback_value()	196
7.43.3.12 transform_to_string()	196
7.43.3.13 type_name()	196
7.43.3.14 unwrapped()	196
7.43.3.15 value_type()	196
7.43.3.16 value_type_name()	197

7.43.3.17 with_fallback()	197
7.43.3.18 with_origin()	197
7.44 hocon::config_value Class Reference	198
7.44.1 Detailed Description	200
7.44.2 Member Enumeration Documentation	200
7.44.2.1 type	200
7.44.3 Member Function Documentation	201
7.44.3.1 at_key()	201
7.44.3.2 at_path()	201
7.44.3.3 equals()	201
7.44.3.4 origin()	202
7.44.3.5 relativized()	202
7.44.3.6 render() [1/2]	202
7.44.3.7 render() [2/2]	203
7.44.3.8 to_fallback_value()	203
7.44.3.9 type_name()	203
7.44.3.10 value_type()	203
7.44.3.11 value_type_name()	204
7.44.3.12 with_fallback()	204
7.44.3.13 with_origin()	204
7.44.4 Friends And Related Symbol Documentation	205
7.44.4.1 config	205
7.44.4.2 config_concatenation	205
7.44.4.3 config_delayed_merge	205
7.44.4.4 config_delayed_merge_object	205
7.44.4.5 config_object	205
7.44.4.6 default_transformer	205
7.44.4.7 resolve_context	206
7.44.4.8 simple_config_list	206
7.44.4.9 simple_config_object	206
7.44.4.10 token	206
7.44.4.11 value	206
7.45 hocon::config_value_factory Class Reference	206
7.45.1 Detailed Description	206
7.45.2 Member Function Documentation	207
7.45.2.1 from_any_ref()	207
7.46 hocon::container Class Reference	207
7.46.1 Detailed Description	208
7.46.2 Member Function Documentation	208
7.46.2.1 has_descendant()	208
7.46.2.2 replace_child()	208
7.47 hocon::default_transformer Class Reference	208

7.47.1 Detailed Description	209
7.48 hocon::double_slash_comment Class Reference	209
7.48.1 Detailed Description	209
7.48.2 Member Function Documentation	209
7.48.2.1 operator==()	209
7.48.2.2 to_string()	210
7.48.2.3 token_text()	210
7.49 hocon::file_name_source Class Reference	210
7.49.1 Detailed Description	210
7.49.2 Member Function Documentation	210
7.49.2.1 context_initialized()	210
7.49.2.2 get_context()	211
7.49.2.3 name_to_parseable()	211
7.49.2.4 set_context()	211
7.50 hocon::full_includer Class Reference	211
7.50.1 Detailed Description	211
7.50.2 Member Function Documentation	212
7.50.2.1 include()	212
7.50.2.2 include_file()	212
7.50.2.3 with_fallback()	212
7.51 FwdListIter< T > Class Template Reference	213
7.51.1 Detailed Description	213
7.51.2 Constructor & Destructor Documentation	213
7.51.2.1 FwdListIter() [1/2]	213
7.51.2.2 FwdListIter() [2/2]	214
7.51.3 Member Function Documentation	214
7.51.3.1 operator!=(())	214
7.51.3.2 operator*()	214
7.51.3.3 operator++()	214
7.51.3.4 operator==(())	214
7.52 hocon::generic_exception Struct Reference	215
7.52.1 Detailed Description	215
7.52.2 Constructor & Destructor Documentation	215
7.52.2.1 generic_exception()	215
7.53 hocon::hash_comment Class Reference	215
7.53.1 Detailed Description	216
7.53.2 Member Function Documentation	216
7.53.2.1 operator==()	216
7.53.2.2 to_string()	216
7.53.2.3 token_text()	216
7.54 hocon::ignored_whitespace Class Reference	216
7.54.1 Detailed Description	217

7.54.2 Member Function Documentation	217
7.54.2.1 operator==()	217
7.54.2.2 to_string()	217
7.55 hocon::io_exception Struct Reference	217
7.55.1 Detailed Description	218
7.55.2 Constructor & Destructor Documentation	218
7.55.2.1 io_exception()	218
7.56 hocon::iterator Class Reference	218
7.56.1 Detailed Description	218
7.57 hocon::iterator_wrapper< iter > Class Template Reference	219
7.57.1 Detailed Description	219
7.57.2 Constructor & Destructor Documentation	219
7.57.2.1 iterator_wrapper()	219
7.57.3 Member Function Documentation	219
7.57.3.1 has_next()	219
7.57.3.2 next()	220
7.58 hocon::line Class Reference	220
7.58.1 Detailed Description	220
7.58.2 Member Function Documentation	220
7.58.2.1 operator==()	220
7.58.2.2 to_string()	221
7.59 List< T > Class Template Reference	221
7.59.1 Detailed Description	221
7.59.2 Constructor & Destructor Documentation	221
7.59.2.1 List() [1/3]	221
7.59.2.2 List() [2/3]	222
7.59.2.3 List() [3/3]	222
7.59.3 Member Function Documentation	222
7.59.3.1 forEach()	222
7.59.3.2 front()	222
7.59.3.3 headCount()	222
7.59.3.4 insertedAt()	222
7.59.3.5 isEmpty()	223
7.59.3.6 member()	223
7.59.3.7 popped_front()	223
7.59.3.8 pushed_front()	223
7.59.3.9 removed()	223
7.59.3.10 removed1()	223
7.59.3.11 take()	224
7.59.4 Friends And Related Symbol Documentation	224
7.59.4.1 FwdListIter< T >	224
7.60 hocon::missing_exception Struct Reference	224

7.60.1 Detailed Description	225
7.60.2 Constructor & Destructor Documentation	225
7.60.2.1 missing_exception()	225
7.60.3 Member Function Documentation	225
7.60.3.1 config_exception() [1/3]	225
7.60.3.2 config_exception() [2/3]	225
7.60.3.3 config_exception() [3/3]	225
7.61 hocon::config_value::modifier Class Reference	226
7.61.1 Detailed Description	226
7.62 hocon::name_source Class Reference	226
7.62.1 Detailed Description	226
7.62.2 Constructor & Destructor Documentation	227
7.62.2.1 name_source() [1/2]	227
7.62.2.2 name_source() [2/2]	227
7.62.3 Member Function Documentation	227
7.62.3.1 context_initialized()	227
7.62.3.2 get_context()	227
7.62.3.3 set_context()	227
7.63 hocon::config_value::no_exceptions_modifier Class Reference	227
7.63.1 Detailed Description	228
7.63.2 Member Function Documentation	228
7.63.2.1 modify_child_may_throw()	228
7.64 hocon::not_possible_to_resolve_exception Struct Reference	228
7.64.1 Detailed Description	228
7.64.2 Constructor & Destructor Documentation	229
7.64.2.1 not_possible_to_resolve_exception()	229
7.65 hocon::not_resolved_exception Struct Reference	229
7.65.1 Detailed Description	229
7.65.2 Constructor & Destructor Documentation	230
7.65.2.1 not_resolved_exception()	230
7.66 hocon::null_exception Struct Reference	230
7.66.1 Detailed Description	230
7.66.2 Constructor & Destructor Documentation	231
7.66.2.1 null_exception()	231
7.66.3 Member Function Documentation	231
7.66.3.1 config_exception() [1/3]	231
7.66.3.2 config_exception() [2/3]	231
7.66.3.3 config_exception() [3/3]	231
7.67 OutListIter< T > Class Template Reference	231
7.67.1 Detailed Description	232
7.67.2 Constructor & Destructor Documentation	232
7.67.2.1 OutListIter()	232

7.67.3 Member Function Documentation	232
7.67.3.1 getList()	232
7.67.3.2 operator*()	232
7.67.3.3 operator++()	232
7.68 hocon::config_document_parser::parse_context Class Reference	233
7.68.1 Detailed Description	233
7.68.2 Member Function Documentation	233
7.68.2.1 parse_single_value()	233
7.69 hocon::config_parser::parse_context Class Reference	233
7.69.1 Detailed Description	233
7.69.2 Member Data Documentation	234
7.69.2.1 array_count	234
7.70 hocon::parse_exception Struct Reference	234
7.70.1 Detailed Description	234
7.70.2 Constructor & Destructor Documentation	235
7.70.2.1 parse_exception()	235
7.71 hocon::parseable Class Reference	235
7.71.1 Detailed Description	236
7.71.2 Member Function Documentation	236
7.71.2.1 options()	236
7.71.2.2 origin()	236
7.71.2.3 parse()	236
7.72 hocon::parseable_file Class Reference	237
7.72.1 Detailed Description	237
7.72.2 Member Function Documentation	238
7.72.2.1 create_origin()	238
7.72.2.2 guess_syntax()	238
7.72.2.3 options()	238
7.72.2.4 origin()	238
7.72.2.5 parse()	238
7.72.2.6 reader()	239
7.73 hocon::parseable_not_found Class Reference	239
7.73.1 Detailed Description	240
7.73.2 Member Function Documentation	240
7.73.2.1 create_origin()	240
7.73.2.2 options()	240
7.73.2.3 origin()	240
7.73.2.4 parse()	240
7.73.2.5 reader()	241
7.74 hocon::parseable_resources Class Reference	241
7.74.1 Detailed Description	242
7.74.2 Member Function Documentation	242

7.74.2.1 create_origin()	242
7.74.2.2 options()	242
7.74.2.3 origin()	242
7.74.2.4 parse()	242
7.74.2.5 reader()	243
7.75 hocon::parseable_string Class Reference	243
7.75.1 Detailed Description	244
7.75.2 Member Function Documentation	244
7.75.2.1 create_origin()	244
7.75.2.2 options()	244
7.75.2.3 origin()	244
7.75.2.4 parse()	244
7.75.2.5 reader()	245
7.76 hocon::path Class Reference	245
7.76.1 Detailed Description	246
7.76.2 Member Function Documentation	246
7.76.2.1 has_funky_chars()	246
7.76.2.2 parent()	246
7.76.2.3 remainder()	246
7.76.2.4 render()	246
7.76.2.5 to_string()	246
7.77 hocon::path_builder Class Reference	247
7.77.1 Detailed Description	247
7.77.2 Member Function Documentation	247
7.77.2.1 result()	247
7.78 hocon::path_parser Class Reference	247
7.78.1 Detailed Description	247
7.79 hocon::problem Class Reference	248
7.79.1 Detailed Description	248
7.79.2 Member Function Documentation	248
7.79.2.1 operator==()	248
7.79.2.2 to_string()	248
7.80 hocon::problem_exception Class Reference	249
7.80.1 Detailed Description	249
7.81 hocon::relative_name_source Class Reference	249
7.81.1 Detailed Description	249
7.81.2 Member Function Documentation	250
7.81.2.1 context_initialized()	250
7.81.2.2 get_context()	250
7.81.2.3 name_to_parseable()	250
7.81.2.4 set_context()	250
7.82 hocon::replaceable_merge_stack Class Reference	250

7.82.1 Detailed Description	251
7.82.2 Member Function Documentation	251
7.82.2.1 has_descendant()	251
7.82.2.2 replace_child()	251
7.83 hocon::resolve_context Class Reference	252
7.83.1 Detailed Description	252
7.84 hocon::resolve_result< V > Struct Template Reference	252
7.84.1 Detailed Description	252
7.84.2 Constructor & Destructor Documentation	253
7.84.2.1 resolve_result()	253
7.84.3 Member Data Documentation	253
7.84.3.1 context	253
7.84.3.2 value	253
7.85 hocon::resolve_source Class Reference	253
7.85.1 Detailed Description	254
7.85.2 Member Typedef Documentation	254
7.85.2.1 node	254
7.86 hocon::resolve_source::result_with_path Struct Reference	254
7.86.1 Detailed Description	254
7.86.2 Member Data Documentation	255
7.86.2.1 path_from_root	255
7.86.2.2 result	255
7.87 hocon::simple_config_document Class Reference	255
7.87.1 Detailed Description	256
7.87.2 Member Function Documentation	256
7.87.2.1 has_path()	256
7.87.2.2 render()	256
7.87.2.3 with_value()	256
7.87.2.4 with_value_text()	257
7.87.2.5 without_path()	257
7.88 hocon::simple_config_list Class Reference	258
7.88.1 Detailed Description	260
7.88.2 Member Typedef Documentation	260
7.88.2.1 iterator	260
7.88.3 Member Enumeration Documentation	260
7.88.3.1 type	260
7.88.4 Member Function Documentation	260
7.88.4.1 at_key()	260
7.88.4.2 at_path()	261
7.88.4.3 begin()	261
7.88.4.4 contains()	261
7.88.4.5 end()	261

7.88.4.6 equals()	262
7.88.4.7 get()	262
7.88.4.8 get_resolve_status()	262
7.88.4.9 has_descendant()	262
7.88.4.10 index_of()	262
7.88.4.11 is_empty()	262
7.88.4.12 new_copy()	263
7.88.4.13 operator==()	263
7.88.4.14 operator[]()	263
7.88.4.15 origin()	263
7.88.4.16 relativized()	263
7.88.4.17 render() [1/3]	264
7.88.4.18 render() [2/3]	264
7.88.4.19 render() [3/3]	264
7.88.4.20 replace_child()	265
7.88.4.21 resolve_substitutions()	265
7.88.4.22 size()	265
7.88.4.23 to_fallback_value()	265
7.88.4.24 type_name()	265
7.88.4.25 unwrapped()	266
7.88.4.26 value_type()	266
7.88.4.27 value_type_name()	266
7.88.4.28 with_fallback()	266
7.88.4.29 with_origin()	267
7.89 hocon::simple_config_object Class Reference	267
7.89.1 Detailed Description	270
7.89.2 Member Typedef Documentation	270
7.89.2.1 iterator	270
7.89.3 Member Enumeration Documentation	270
7.89.3.1 type	270
7.89.4 Member Function Documentation	270
7.89.4.1 at_key()	270
7.89.4.2 at_path()	270
7.89.4.3 attempt_peek_with_partial_resolve()	271
7.89.4.4 begin()	271
7.89.4.5 construct_delayed_merge()	271
7.89.4.6 end()	272
7.89.4.7 entry_set()	272
7.89.4.8 equals()	272
7.89.4.9 get()	272
7.89.4.10 get_resolve_status()	272
7.89.4.11 has_descendant()	272

7.89.4.12 ignores_fallbacks()	273
7.89.4.13 is_empty()	273
7.89.4.14 key_set()	273
7.89.4.15 merged_with_object()	273
7.89.4.16 new_copy()	273
7.89.4.17 operator==()	273
7.89.4.18 operator[]()	274
7.89.4.19 origin()	274
7.89.4.20 relativized()	274
7.89.4.21 render() [1/3]	275
7.89.4.22 render() [2/3]	275
7.89.4.23 render() [3/3]	275
7.89.4.24 replace_child()	276
7.89.4.25 resolve_substitutions()	276
7.89.4.26 size()	276
7.89.4.27 to_config()	276
7.89.4.28 to_fallback_value()	276
7.89.4.29 type_name()	277
7.89.4.30 unwrapped()	277
7.89.4.31 value_set()	277
7.89.4.32 value_type()	277
7.89.4.33 value_type_name()	277
7.89.4.34 with_fallback()	278
7.89.4.35 with_fallbacks_ignored()	278
7.89.4.36 with_only_path()	279
7.89.4.37 with_only_path_or_null()	279
7.89.4.38 with_origin()	279
7.89.4.39 with_value() [1/2]	279
7.89.4.40 with_value() [2/2]	279
7.89.4.41 without_path()	280
7.90 hocon::simple_config_origin Class Reference	280
7.90.1 Detailed Description	281
7.90.2 Constructor & Destructor Documentation	281
7.90.2.1 simple_config_origin()	281
7.90.3 Member Function Documentation	281
7.90.3.1 comments()	281
7.90.3.2 description()	281
7.90.3.3 line_number()	282
7.90.3.4 with_comments()	282
7.90.3.5 with_line_number()	282
7.91 hocon::simple_include_context Class Reference	283
7.91.1 Detailed Description	283

7.91.2 Member Function Documentation	283
7.91.2.1 get_cur_dir()	283
7.91.2.2 parse_options()	283
7.91.2.3 relative_to()	284
7.91.2.4 set_cur_dir()	284
7.91.3 Member Data Documentation	284
7.91.3.1 _cur_dir	284
7.92 hocon::simple_includer Class Reference	284
7.92.1 Detailed Description	285
7.92.2 Member Function Documentation	285
7.92.2.1 include()	285
7.92.2.2 include_file()	286
7.92.2.3 with_fallback()	286
7.93 hocon::single_token_iterator Class Reference	287
7.93.1 Detailed Description	287
7.93.2 Member Function Documentation	287
7.93.2.1 has_next()	287
7.93.2.2 next()	287
7.94 hocon::substitution Class Reference	287
7.94.1 Detailed Description	288
7.94.2 Member Function Documentation	288
7.94.2.1 operator==()	288
7.94.2.2 to_string()	288
7.94.2.3 token_text()	288
7.95 hocon::substitution_expression Class Reference	288
7.95.1 Detailed Description	289
7.96 hocon::token Class Reference	289
7.96.1 Detailed Description	289
7.97 hocon::token_iterator Class Reference	290
7.97.1 Detailed Description	290
7.97.2 Member Function Documentation	290
7.97.2.1 has_next()	290
7.97.2.2 next()	290
7.98 hocon::token_list_iterator Class Reference	291
7.98.1 Detailed Description	291
7.98.2 Member Function Documentation	291
7.98.2.1 has_next()	291
7.98.2.2 next()	291
7.99 hocon::tokens Class Reference	292
7.99.1 Detailed Description	292
7.99.2 Member Function Documentation	292
7.99.2.1 start_token()	292

7.100 hocon::unmergeable Class Reference	292
7.100.1 Detailed Description	293
7.101 hocon::unquoted_text Class Reference	293
7.101.1 Detailed Description	293
7.101.2 Member Function Documentation	293
7.101.2.1 operator==()	293
7.101.2.2 to_string()	294
7.102 hocon::unresolved_substitution_exception Struct Reference	294
7.102.1 Detailed Description	294
7.102.2 Constructor & Destructor Documentation	295
7.102.2.1 unresolved_substitution_exception()	295
7.103 hocon::unsupported_exception Struct Reference	295
7.103.1 Detailed Description	295
7.104 hocon::validation_failed_exception Struct Reference	295
7.104.1 Detailed Description	296
7.104.2 Constructor & Destructor Documentation	296
7.104.2.1 validation_failed_exception()	296
7.104.3 Member Data Documentation	296
7.104.3.1 problems	296
7.105 hocon::validation_problem Struct Reference	296
7.105.1 Detailed Description	297
7.105.2 Constructor & Destructor Documentation	297
7.105.2.1 validation_problem()	297
7.105.3 Member Function Documentation	297
7.105.3.1 to_string()	297
7.105.4 Member Data Documentation	297
7.105.4.1 origin	297
7.105.4.2 path	298
7.105.4.3 problem	298
7.106 hocon::value Class Reference	298
7.106.1 Detailed Description	298
7.106.2 Member Function Documentation	298
7.106.2.1 operator==()	298
7.106.2.2 origin()	299
7.106.2.3 to_string()	299
7.107 hocon::wrong_type_exception Struct Reference	299
7.107.1 Detailed Description	299
7.107.2 Constructor & Destructor Documentation	300
7.107.2.1 wrong_type_exception()	300
7.107.3 Member Function Documentation	300
7.107.3.1 config_exception() [1/3]	300
7.107.3.2 config_exception() [2/3]	300

7.107.3.3 config_exception() [3/3]	300
8 File Documentation	301
8.1 config.hpp	301
8.2 config_exception.hpp	303
8.3 config_include_context.hpp	305
8.4 config_includer.hpp	305
8.5 config_includer_file.hpp	305
8.6 config_list.hpp	305
8.7 config_mergeable.hpp	306
8.8 config_object.hpp	306
8.9 config_origin.hpp	307
8.10 config_parse_options.hpp	307
8.11 config_parseable.hpp	308
8.12 config_render_options.hpp	308
8.13 config_resolve_options.hpp	309
8.14 config_syntax.hpp	309
8.15 config_value.hpp	309
8.16 config_value_factory.hpp	311
8.17 export.h	311
8.18 functional_list.hpp	312
8.19 config_document.hpp	316
8.20 config_document_factory.hpp	316
8.21 config_node.hpp	316
8.22 path.hpp	317
8.23 program_options.hpp	317
8.24 types.hpp	318
8.25 hocon/version.h File Reference	319
8.25.1 Detailed Description	319
8.25.2 Macro Definition Documentation	320
8.25.2.1 CPP_HOCON_VERSION	320
8.25.2.2 CPP_HOCON_VERSION_MAJOR	320
8.25.2.3 CPP_HOCON_VERSION_MINOR	320
8.25.2.4 CPP_HOCON_VERSION_PATCH	320
8.25.2.5 CPP_HOCON_VERSION_WITH_COMMIT	320
8.26 version.h	320
8.27 config_document_parser.hpp	321
8.28 config_parser.hpp	321
8.29 config_util.hpp	322
8.30 container.hpp	322
8.31 default_transformer.hpp	323
8.32 full_includer.hpp	323

8.33 abstract_config_node.hpp	323
8.34 abstract_config_node_value.hpp	323
8.35 config_node_array.hpp	324
8.36 config_node_comment.hpp	324
8.37 config_node_complex_value.hpp	324
8.38 config_node_concatenation.hpp	324
8.39 config_node_field.hpp	325
8.40 config_node_include.hpp	325
8.41 config_node_object.hpp	325
8.42 config_node_path.hpp	326
8.43 config_node_root.hpp	326
8.44 config_node_simple_value.hpp	327
8.45 config_node_single_token.hpp	327
8.46 parseable.hpp	327
8.47 path_builder.hpp	329
8.48 path_parser.hpp	329
8.49 replaceable_merge_stack.hpp	330
8.50 resolve_context.hpp	330
8.51 resolve_result.hpp	331
8.52 resolve_source.hpp	331
8.53 simple_config_document.hpp	332
8.54 simple_config_origin.hpp	333
8.55 simple_include_context.hpp	333
8.56 simple_includer.hpp	334
8.57 substitution_expression.hpp	335
8.58 token.hpp	336
8.59 tokenizer.hpp	336
8.60 tokens.hpp	338
8.61 unmergeable.hpp	339
8.62 config_boolean.hpp	340
8.63 config_concatenation.hpp	340
8.64 config_delayed_merge.hpp	341
8.65 config_delayed_merge_object.hpp	341
8.66 config_double.hpp	342
8.67 config_int.hpp	343
8.68 config_long.hpp	343
8.69 config_null.hpp	343
8.70 config_number.hpp	344
8.71 config_reference.hpp	344
8.72 config_string.hpp	345
8.73 simple_config_list.hpp	345
8.74 simple_config_object.hpp	346

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:

hocon	Factory for creating config_document instances	15
std	STL namespace	23

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

hocon::config_document	86
hocon::simple_config_document	255
hocon::config_include_context	98
hocon::simple_include_context	283
hocon::config_includer	101
hocon::full_includer	211
hocon::simple_includer	284
hocon::config_includer_file	102
hocon::full_includer	211
hocon::simple_includer	284
hocon::config_mergeable	127
hocon::config	35
hocon::config_value	198
hocon::config_boolean	49
hocon::config_concatenation	57
hocon::config_delayed_merge	65
hocon::config_list	111
hocon::simple_config_list	258
hocon::config_null	143
hocon::config_number	151
hocon::config_double	89
hocon::config_int	104
hocon::config_long	119
hocon::config_object	158
hocon::config_delayed_merge_object	74
hocon::simple_config_object	267
hocon::config_reference	177
hocon::config_string	190
hocon::config_node	129
hocon::abstract_config_node	29
hocon::abstract_config_node_value	30
hocon::config_node_complex_value	132
hocon::config_node_array	130

hocon::config_node_concatenation	133
hocon::config_node_object	137
hocon::config_node_root	140
hocon::config_node_field	135
hocon::config_node_simple_value	141
hocon::config_node_include	136
hocon::config_node_path	139
hocon::config_node_single_token	142
hocon::config_node_comment	131
hocon::config_origin	168
hocon::simple_config_origin	280
hocon::config_parse_options	170
hocon::config_parseable	175
hocon::parseable	235
hocon::parseable_file	237
hocon::parseable_not_found	239
hocon::parseable_resources	241
hocon::parseable_string	243
hocon::config_render_options	184
hocon::config_resolve_options	188
hocon::config_value_factory	206
hocon::container	207
hocon::config_concatenation	57
hocon::replaceable_merge_stack	250
hocon::config_delayed_merge	65
hocon::config_delayed_merge_object	74
hocon::simple_config_list	258
hocon::simple_config_object	267
hocon::default_transformer	208
std::enable_shared_from_this	
hocon::config	35
hocon::config_value	198
hocon::parseable	235
hocon::simple_config_origin	280
hocon::simple_includer	284
hocon::substitution_expression	288
std::exception	
std::runtime_error	
hocon::config_exception	97
hocon::bad_path_exception	31
hocon::bad_value_exception	32
hocon::bug_or_broken_exception	33
hocon::not_possible_to_resolve_exception	228
hocon::not_resolved_exception	229
hocon::generic_exception	215
hocon::io_exception	217
hocon::missing_exception	224
hocon::null_exception	230
hocon::parse_exception	234
hocon::unresolved_substitution_exception	294
hocon::validation_failed_exception	295
hocon::wrong_type_exception	299
hocon::problem_exception	249
hocon::unsupported_exception	295
hocon::iterator	218
hocon::iterator_wrapper< iter >	219

hocon::single_token_iterator	287
hocon::token_iterator	290
hocon::token_list_iterator	291
std::iterator	
FwdListIter< T >	213
OutListIter< T >	231
List< T >	221
List< shared_string >	221
hocon::config_value::modifier	226
hocon::config_value::no_exceptions_modifier	227
hocon::name_source	226
hocon::file_name_source	210
hocon::relative_name_source	249
hocon::config_document_parser::parse_context	233
hocon::config_parser::parse_context	233
hocon::path	245
hocon::path_builder	247
hocon::path_parser	247
hocon::resolve_context	252
hocon::resolve_result< V >	252
hocon::resolve_result< shared_value >	252
hocon::resolve_source	253
hocon::resolve_source::result_with_path	254
hocon::token	289
hocon::comment	34
hocon::double_slash_comment	209
hocon::hash_comment	215
hocon::ignored_whitespace	216
hocon::line	220
hocon::problem	248
hocon::substitution	287
hocon::unquoted_text	293
hocon::value	298
hocon::tokens	292
hocon::unmergeable	292
hocon::config_concatenation	57
hocon::config_delayed_merge	65
hocon::config_delayed_merge_object	74
hocon::config_reference	177
hocon::validation_problem	296

Chapter 4

Class Index

4.1 Class List

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

hocon::abstract_config_node	29
hocon::abstract_config_node_value	
This is used to classify certain abstract_config_node subclasses	30
hocon::bad_path_exception	
Exception indicating that a path expression was invalid	31
hocon::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	32
hocon::bug_or_broken_exception	
Exception indicating that there's a bug in something (possibly the library itself) or the runtime environment is broken	33
hocon::comment	34
hocon::config	
An immutable map from config paths to config values	35
hocon::config_boolean	49
hocon::config_concatenation	
A ConfigConcatenation represents a list of values to be concatenated (see the spec)	57
hocon::config_delayed_merge	65
hocon::config_delayed_merge_object	74
hocon::config_document	
Represents an individual HOCON or JSON file, preserving all formatting and syntax details	86
hocon::config_double	89
hocon::config_exception	
All exceptions thrown by the library are subclasses of config_exception	97
hocon::config_include_context	
Context provided to a config_includer ; this interface is only useful inside a config_includer implementation, and is not intended for apps to implement	98
hocon::config_includer	
Implement this interface and provide an instance to config_parse_options.set_includer() to customize handling of <code>include</code> statements in config files	101
hocon::config_includer_file	
Implement this <i>in addition to</i> config_includer if you want to support inclusion of files with the <code>include file("filename")</code> syntax	102
hocon::config_int	104
hocon::config_list	
Subtype of <code>ConfigValue</code> representing a list value, as in JSON's <code>[1, 2, 3]</code> syntax	111

hocon::config_long	119
hocon::config_mergeable	127
hocon::config_node	
A node in the syntax tree for a HOCON or JSON document	129
hocon::config_node_array	130
hocon::config_node_comment	131
hocon::config_node_complex_value	132
hocon::config_node_concatenation	133
hocon::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	135
hocon::config_node_include	
Represents an include statement of the form "include include_kind(include_path)"	136
hocon::config_node_object	137
hocon::config_node_path	139
hocon::config_node_root	140
hocon::config_node_simple_value	141
hocon::config_node_single_token	142
hocon::config_null	
This exists because sometimes null is not the same as missing	143
hocon::config_number	151
hocon::config_object	158
hocon::config_origin	
Represents the origin (such as filename and line number) of a config_value for use in error messages	168
hocon::config_parse_options	
A set of options related to parsing	170
hocon::config_parseable	
An opaque handle to something that can be parsed, obtained from config_include_context	175
hocon::config_reference	177
hocon::config_render_options	184
hocon::config_resolve_options	
A set of options related to resolving substitutions	188
hocon::config_string	190
hocon::config_value	
An immutable value, following the JSON type schema	198
hocon::config_value_factory	206
hocon::container	
An AbstractConfigValue which contains other values	207
hocon::default_transformer	208
hocon::double_slash_comment	209
hocon::file_name_source	210
hocon::full_includer	211
FwdListIter< T >	213
hocon::generic_exception	
Exception that doesn't fall into any other category	215
hocon::hash_comment	215
hocon::ignored_whitespace	216
hocon::io_exception	
Exception indicating that there was an IO error	217
hocon::iterator	218
hocon::iterator_wrapper< iter >	219
hocon::line	220
List< T >	221
hocon::missing_exception	
Exception indicates that the setting was never set to anything, not even null	224
hocon::config_value::modifier	226

hocon::name_source	226
hocon::config_value::no_exceptions_modifier	227
hocon::not_possible_to_resolve_exception	228
hocon::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, <code>config#resolve</code> was not called)	229
hocon::null_exception	
Exception indicates that the setting was treated as missing because it was set to null	230
OutListIter< T >	231
hocon::config_document_parser::parse_context	233
hocon::config_parser::parse_context	233
hocon::parse_exception	
Exception indicating that there was a parse error	234
hocon::parseable	235
hocon::parseable_file	237
hocon::parseable_not_found	239
hocon::parseable_resources	241
hocon::parseable_string	243
hocon::path	245
hocon::path_builder	247
hocon::path_parser	247
hocon::problem	248
hocon::problem_exception	249
hocon::relative_name_source	249
hocon::replaceable_merge_stack	250
hocon::resolve_context	252
hocon::resolve_result< V >	252
hocon::resolve_source	253
hocon::resolve_source::result_with_path	254
hocon::simple_config_document	255
hocon::simple_config_list	258
hocon::simple_config_object	267
hocon::simple_config_origin	280
hocon::simple_include_context	283
hocon::simple_includer	284
hocon::single_token_iterator	287
hocon::substitution	287
hocon::substitution_expression	288
hocon::token	289
hocon::token_iterator	290
hocon::token_list_iterator	291
hocon::tokens	292
hocon::unmergeable	
Interface that tags a ConfigValue that is not mergeable until after substitutions are resolved	292
hocon::unquoted_text	293
hocon::unresolved_substitution_exception	
Exception indicating that a substitution did not resolve to anything	294
hocon::unsupported_exception	295
hocon::validation_failed_exception	
Exception indicating that <code>config#check_valid</code> found validity problems	295
hocon::validation_problem	
Information about a problem that occurred in <code>config#check_valid</code>	296
hocon::value	298
hocon::wrong_type_exception	
Exception indicating that the type of a value does not match the type you requested	299

Chapter 5

File Index

5.1 File List

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

hocon/config.hpp	301
hocon/config_exception.hpp	303
hocon/config_include_context.hpp	305
hocon/config_includer.hpp	305
hocon/config_includer_file.hpp	305
hocon/config_list.hpp	305
hocon/config_mergeable.hpp	306
hocon/config_object.hpp	306
hocon/config_origin.hpp	307
hocon/config_parse_options.hpp	307
hocon/config_parseable.hpp	308
hocon/config_render_options.hpp	308
hocon/config_resolve_options.hpp	309
hocon/config_syntax.hpp	309
hocon/config_value.hpp	309
hocon/config_value_factory.hpp	311
hocon/export.h	311
hocon/functional_list.hpp	312
hocon/path.hpp	317
hocon/program_options.hpp	317
hocon/types.hpp	318
hocon/version.h	
Declares macros for the cpp-hocon library version	319
hocon/parser/config_document.hpp	316
hocon/parser/config_document_factory.hpp	316
hocon/parser/config_node.hpp	316
internal/config_document_parser.hpp	321
internal/config_parser.hpp	321
internal/config_util.hpp	322
internal/container.hpp	322
internal/default_transformer.hpp	323
internal/full_includer.hpp	323
internal/parseable.hpp	327
internal/path_builder.hpp	329
internal/path_parser.hpp	329

internal/replaceable_merge_stack.hpp	330
internal/resolve_context.hpp	330
internal/resolve_result.hpp	331
internal/resolve_source.hpp	331
internal/simple_config_document.hpp	332
internal/simple_config_origin.hpp	333
internal/simple_include_context.hpp	333
internal/simple_includer.hpp	334
internal/substitution_expression.hpp	335
internal/token.hpp	336
internal/tokenizer.hpp	336
internal/tokens.hpp	338
internal/unmergeable.hpp	339
internal/nodes/abstract_config_node.hpp	323
internal/nodes/abstract_config_node_value.hpp	323
internal/nodes/config_node_array.hpp	324
internal/nodes/config_node_comment.hpp	324
internal/nodes/config_node_complex_value.hpp	324
internal/nodes/config_node_concatenation.hpp	324
internal/nodes/config_node_field.hpp	325
internal/nodes/config_node_include.hpp	325
internal/nodes/config_node_object.hpp	325
internal/nodes/config_node_path.hpp	326
internal/nodes/config_node_root.hpp	326
internal/nodes/config_node_simple_value.hpp	327
internal/nodes/config_node_single_token.hpp	327
internal/values/config_boolean.hpp	340
internal/values/config_concatenation.hpp	340
internal/values/config_delayed_merge.hpp	341
internal/values/config_delayed_merge_object.hpp	341
internal/values/config_double.hpp	342
internal/values/config_int.hpp	343
internal/values/config_long.hpp	343
internal/values/config_null.hpp	343
internal/values/config_number.hpp	344
internal/values/config_reference.hpp	344
internal/values/config_string.hpp	345
internal/values/simple_config_list.hpp	345
internal/values/simple_config_object.hpp	346

Chapter 6

Namespace Documentation

6.1 hocon Namespace Reference

Factory for creating `config_document` instances.

Classes

- class `abstract_config_node`
- class `abstract_config_node_value`
This is used to classify certain `abstract_config_node` subclasses.
- struct `bad_path_exception`
Exception indicating that a path expression was invalid.
- 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 `bug_or_broken_exception`
Exception indicating that there's a bug in something (possibly the library itself) or the runtime environment is broken.
- class `comment`
- class `config`
An immutable map from config paths to config values.
- 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_document`
Represents an individual HOCON or JSON file, preserving all formatting and syntax details.
- class `config_double`
- struct `config_exception`
All exceptions thrown by the library are subclasses of `config_exception`.
- class `config_include_context`
Context provided to a `config_includer`; this interface is only useful inside a `config_includer` implementation, and is not intended for apps to implement.
- class `config_includer`
Implement this interface and provide an instance to `config_parse_options.set_includer()` to customize handling of `include` statements in config files.

- class [config_includer_file](#)

Implement this in addition to [config_includer](#) if you want to support inclusion of files with the `include file("filename")` syntax.

- class [config_int](#)
- class [config_list](#)

Subtype of `ConfigValue` representing a list value, as in JSON's `[1, 2, 3]` syntax.

- class [config_long](#)
- class [config_mergeable](#)
- class [config_node](#)

A node in the syntax tree for a HOCON or JSON document.

- 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 [config_null](#)

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

- class [config_number](#)
- 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_reference](#)
- class [config_render_options](#)
- class [config_resolve_options](#)

A set of options related to resolving substitutions.

- class [config_string](#)
- class [config_value](#)

An immutable value, following the `JSON` type schema.

- class [config_value_factory](#)
- class [container](#)

An `AbstractConfigValue` which contains other values.

- class [default_transformer](#)
- class [double_slash_comment](#)
- class [file_name_source](#)
- class [full_includer](#)
- struct [generic_exception](#)

Exception that doesn't fall into any other category.

- class [hash_comment](#)
- class [ignored_whitespace](#)
- struct [io_exception](#)

Exception indicating that there was an IO error.

- class [iterator](#)
- class [iterator_wrapper](#)
- class [line](#)
- struct [missing_exception](#)

Exception indicates that the setting was never set to anything, not even null.

- class [name_source](#)
- struct [not_possible_to_resolve_exception](#)
- 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 [null_exception](#)

Exception indicates that the setting was treated as missing because it was set to null.

- struct [parse_exception](#)

Exception indicating that there was a parse error.

- class [parseable](#)
- class [parseable_file](#)
- class [parseable_not_found](#)
- class [parseable_resources](#)
- class [parseable_string](#)
- class [path](#)
- class [path_builder](#)
- class [path_parser](#)
- class [problem](#)
- class [problem_exception](#)
- class [relative_name_source](#)
- class [replaceable_merge_stack](#)
- class [resolve_context](#)
- struct [resolve_result](#)
- class [resolve_source](#)
- class [simple_config_document](#)
- class [simple_config_list](#)
- class [simple_config_object](#)
- class [simple_config_origin](#)
- class [simple_include_context](#)
- class [simple_includer](#)
- class [single_token_iterator](#)
- class [substitution](#)
- class [substitution_expression](#)
- class [token](#)
- class [token_iterator](#)
- class [token_list_iterator](#)
- class [tokens](#)
- class [unmergeable](#)

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

- class [unquoted_text](#)
- struct [unresolved_substitution_exception](#)

Exception indicating that a substitution did not resolve to anything.

- struct [unsupported_exception](#)
- struct [validation_failed_exception](#)

Exception indicating that [config#check_valid](#) found validity problems.

- struct [validation_problem](#)

Information about a problem that occurred in [config#check_valid](#).

- class [value](#)
- struct [wrong_type_exception](#)

Exception indicating that the type of a value does not match the type you requested.

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::make_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 = 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](#).

6.1.2.2 shared_config

```
using hocon::shared_config = std::shared_ptr<const config>
```

Definition at line 24 of file [types.hpp](#).

6.1.2.3 shared_container

```
using hocon::shared_container = std::shared_ptr<const container>
```

Definition at line 45 of file [types.hpp](#).

6.1.2.4 shared_include_context

```
using hocon::shared_include_context = std::shared_ptr<const config_include_context>
```

Definition at line 59 of file [types.hpp](#).

6.1.2.5 shared_includer

```
using hocon::shared_includer = std::shared_ptr<const config_includer>
```

Definition at line 56 of file [types.hpp](#).

6.1.2.6 shared_list

```
using hocon::shared_list = std::shared_ptr<const config_list>
```

Definition at line 38 of file [types.hpp](#).

6.1.2.7 shared_node

```
using hocon::shared_node = std::shared_ptr<const abstract_config_node>
```

Definition at line 48 of file [types.hpp](#).

6.1.2.8 shared_node_array

```
using hocon::shared_node_array = std::shared_ptr<const config_node_array>
```

Definition at line 8 of file [config_node_array.hpp](#).

6.1.2.9 shared_node_concatenation

```
using hocon::shared_node_concatenation = std::shared_ptr<const config_node_concatenation>
```

Definition at line 14 of file [config_node_concatenation.hpp](#).

6.1.2.10 shared_node_list

```
using hocon::shared_node_list = std::vector<shared_node>
```

Definition at line 49 of file [types.hpp](#).

6.1.2.11 shared_node_object

```
using hocon::shared_node_object = std::shared_ptr<const config_node_object>
```

Definition at line 11 of file [config_node_object.hpp](#).

6.1.2.12 shared_node_value

```
using hocon::shared_node_value = std::shared_ptr<const abstract_config_node_value>
```

Definition at line 10 of file [abstract_config_node_value.hpp](#).

6.1.2.13 shared_object

```
using hocon::shared_object = std::shared_ptr<const config_object>
```

Definition at line 27 of file [types.hpp](#).

6.1.2.14 shared_origin

```
using hocon::shared_origin = std::shared_ptr<const config_origin>
```

Definition at line 30 of file [types.hpp](#).

6.1.2.15 shared_parseable

```
using hocon::shared_parseable = std::shared_ptr<const config_parseable>
```

Definition at line 62 of file [types.hpp](#).

6.1.2.16 shared_string

```
using hocon::shared_string = std::shared_ptr<const std::string>
```

Definition at line 51 of file [types.hpp](#).

6.1.2.17 shared_token

```
using hocon::shared_token = std::shared_ptr<const token>
```

Definition at line 42 of file [token.hpp](#).

6.1.2.18 shared_value

```
using hocon::shared_value = std::shared_ptr<const config_value>
```

Definition at line 35 of file [types.hpp](#).

6.1.2.19 token_list

```
using hocon::token_list = std::vector<shared_token>
```

Definition at line 43 of file [token.hpp](#).

6.1.2.20 unwrapped_value

```
typedef boost::make_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 hocon::unwrapped_value
```

Definition at line 42 of file [types.hpp](#).

6.1.3 Enumeration Type Documentation

6.1.3.1 config_include_kind

```
enum class hocon::config_include_kind [strong]
```

Definition at line 7 of file [config_node_include.hpp](#).

6.1.3.2 config_string_type

```
enum class hocon::config_string_type [strong]
```

Definition at line 8 of file [config_string.hpp](#).

6.1.3.3 origin_type

```
enum class hocon::origin_type [strong]
```

Definition at line 11 of file [simple_config_origin.hpp](#).

6.1.3.4 resolve_status

```
enum class hocon::resolve_status [strong]
```

Definition at line 22 of file [config_value.hpp](#).

6.1.3.5 time_unit

```
enum class hocon::time_unit [strong]
```

Definition at line 20 of file [config.hpp](#).

6.1.3.6 token_type

```
enum class hocon::token_type [strong]
```

Definition at line 9 of file [token.hpp](#).

6.1.4 Function Documentation

6.1.4.1 make_resolve_result()

```
template<typename T >  
static resolve_result< shared_value > hocon::make_resolve_result (  
    resolve_context context,  
    T value) [static]
```

Definition at line 17 of file [resolve_result.hpp](#).

6.1.4.2 operator==()

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

Config documents compare via rendered strings.

6.2 std Namespace Reference

STL namespace.

Classes

- class **allocator**
STL class.
- class **array**
STL class.
- class **atomic**
STL class.
- class **atomic_ref**
STL class.
- class **auto_ptr**
STL class.
- class **bad_alloc**
STL class.
- class **bad_cast**
STL class.
- class **bad_exception**
STL class.
- class **bad_typeid**
STL class.
- class **basic_fstream**
STL class.
- class **basic_ifstream**
STL class.
- class **basic_ios**
STL class.
- class **basic_iostream**
STL class.
- class **basic_istream**
STL class.
- class **basic_istreamstream**
STL class.
- class **basic_ofstream**
STL class.
- class **basic_ostream**
STL class.
- class **basic_ostreamstream**
STL class.
- class **basic_string**
STL class.
- class **basic_string_view**
STL class.
- class **basic_stringstream**
STL class.
- class **bitset**
STL class.

- class **complex**
STL class.
- class **deque**
STL class.
- class **domain_error**
STL class.
- class **error_category**
STL class.
- class **error_code**
STL class.
- class **error_condition**
STL class.
- class **exception**
STL class.
- class **forward_list**
STL class.
- class **fstream**
STL class.
- class **ifstream**
STL class.
- class **invalid_argument**
STL class.
- class **ios**
STL class.
- class **ios_base**
STL class.
- class **istream**
STL class.
- class **istreamstream**
STL class.
- class **jthread**
STL class.
- class **length_error**
STL class.
- class **list**
STL class.
- class **lock_guard**
STL class.
- class **logic_error**
STL class.
- class **map**
STL class.
- class **multimap**
STL class.
- class **multiset**
STL class.
- class **mutex**
STL class.
- class **ofstream**
STL class.
- class **ostream**

- STL class.*
- class **ostringstream**
 - STL class.*
- class **out_of_range**
 - STL class.*
- class **overflow_error**
 - STL class.*
- class **pair**
 - STL class.*
- class **priority_queue**
 - STL class.*
- class **queue**
 - STL class.*
- class **range_error**
 - STL class.*
- class **recursive_mutex**
 - STL class.*
- class **recursive_timed_mutex**
 - STL class.*
- class **runtime_error**
 - STL class.*
- class **set**
 - STL class.*
- class **shared_lock**
 - STL class.*
- class **shared_mutex**
 - STL class.*
- class **shared_ptr**
 - STL class.*
- class **shared_timed_mutex**
 - STL class.*
- class **smart_ptr**
 - STL class.*
- class **span**
 - STL class.*
- class **stack**
 - STL class.*
- class **string**
 - STL class.*
- class **string_view**
 - STL class.*
- class **stringstream**
 - STL class.*
- class **system_error**
 - STL class.*
- class **thread**
 - STL class.*
- class **timed_mutex**
 - STL class.*
- class **u16string**
 - STL class.*

- class **u16string_view**
STL class.
- class **u32string**
STL class.
- class **u32string_view**
STL class.
- class **u8string**
STL class.
- class **u8string_view**
STL class.
- class **underflow_error**
STL class.
- class **unique_lock**
STL class.
- class **unique_ptr**
STL class.
- class **unordered_map**
STL class.
- class **unordered_multimap**
STL class.
- class **unordered_multiset**
STL class.
- class **unordered_set**
STL class.
- class **valarray**
STL class.
- class **vector**
STL class.
- class **weak_ptr**
STL class.
- class **wfstream**
STL class.
- class **wifstream**
STL class.
- class **wios**
STL class.
- class **wistream**
STL class.
- class **wistringstream**
STL class.
- class **wofstream**
STL class.
- class **wostream**
STL class.
- class **wostreamstream**
STL class.
- class **wstring**
STL class.
- class **wstring_view**
STL class.
- class **wstringstream**
STL class.

Functions

- `template<class T >`
`FwdListIter< T > begin (List< T > const &lst)`
- `template<class T >`
`FwdListIter< T > end (List< T > const &lst)`

6.2.1 Detailed Description

STL namespace.

6.2.2 Function Documentation

6.2.2.1 begin()

```
template<class T >
FwdListIter< T > std::begin (
    List< T > const & lst)
```

Definition at line 162 of file [functional_list.hpp](#).

6.2.2.2 end()

```
template<class T >
FwdListIter< T > std::end (
    List< T > const & lst)
```

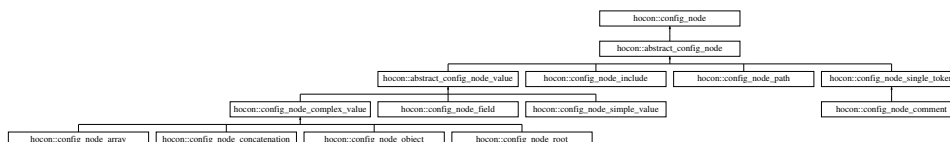
Definition at line 167 of file [functional_list.hpp](#).

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:

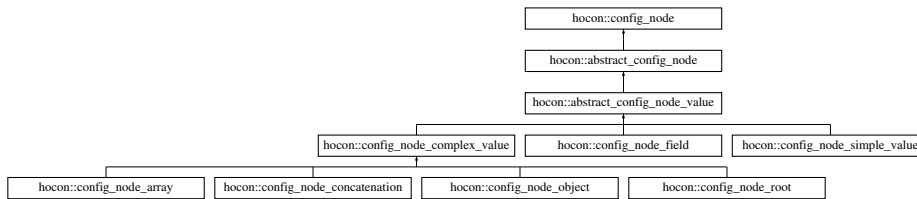
- `internal/nodes/abstract_config_node.hpp`

7.2 hocon::abstract_config_node_value Class Reference

This is used to classify certain [abstract_config_node](#) subclasses.

```
#include <abstract_config_node_value.hpp>
```

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.

```
#include <config_exception.hpp>
```

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](#).

7.3.2 Constructor & Destructor Documentation

7.3.2.1 bad_path_exception() [1/2]

```

hocon::bad_path_exception::bad_path_exception (
    config\_origin const & origin,
    std::string const & path,
    std::string const & message) [inline]
  
```

Definition at line 72 of file [config_exception.hpp](#).

7.3.2.2 bad_path_exception() [2/2]

```

hocon::bad_path_exception::bad_path_exception (
    std::string const & path,
    std::string const & message) [inline]
  
```

Definition at line 74 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.

```
#include <config_exception.hpp>
```

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](#).

7.4.2 Constructor & Destructor Documentation

7.4.2.1 bad_value_exception() [1/2]

```
hocon::bad_value_exception::bad_value_exception (
    config\_origin const & origin,
    std::string const & path,
    std::string const & message) [inline]
```

Definition at line 60 of file [config_exception.hpp](#).

7.4.2.2 bad_value_exception() [2/2]

```
hocon::bad_value_exception::bad_value_exception (
    std::string const & path,
    std::string const & message) [inline]
```

Definition at line 62 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.

```
#include <config_exception.hpp>
```

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](#).

7.5.2 Constructor & Destructor Documentation

7.5.2.1 bug_or_broken_exception()

```
hocon::bug_or_broken_exception::bug_or_broken_exception (
    std::string const & message) [inline]
```

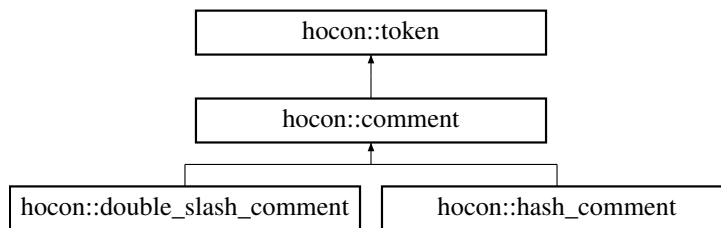
Definition at line 86 of file [config_exception.hpp](#).

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](#).

7.6.2 Member Function Documentation

7.6.2.1 operator==()

```
bool hocon::comment::operator== (
    const token & other) const  [override], [virtual]
```

Reimplemented from [hocon::token](#).

7.6.2.2 to_string()

```
std::string hocon::comment::to_string () const  [override], [virtual]
```

Reimplemented from [hocon::token](#).

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.

```
#include <config.hpp>
```

Inheritance diagram for hocon::config:



Public Member Functions

- virtual shared_object [root](#) () const
Gets the Config as a tree of ConfigObject.
- virtual shared_origin [origin](#) () const
Gets the origin of the Config, which may be a file, or a file with a line number, or just a descriptive phrase.
- 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 Config's root object contains no key-value pairs.
- 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 config at the given path.*
- virtual shared_config **at_key** (std::string const &key) const
 - Places the config inside a config at the given key.*
- virtual shared_config **with_value** (std::string const &path, std::shared_ptr< const config_value > value) const
 - Returns a config based on this one, but with the given path set to the given value.*
- 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_values` 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_me`. 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_homogeneous_unwrapped_list()

```
template<typename T >
std::vector< T > hocon::config::get_homogeneous_unwrapped_list (
    std::string const & path) const [inline]
```

Definition at line 578 of file [config.hpp](#).

7.7.2.7 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 `has_path_or_null(string)` returned true for the same path, so to avoid any possible exception check `has_path_or_null()` first. However, an exception for unset paths will usually be the right thing (because a `reference.conf` should exist that has the path set, the path should never be unset unless something is broken).

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

7.7.2.8 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

7.7.2.9 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

7.7.2.10 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

7.7.2.11 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.

7.7.2.12 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

7.7.2.13 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>fileBasename</i>	a filename with or without extension
---------------------	--------------------------------------

Returns

the parsed configuration

7.7.2.14 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 `fileBasename` already ends in a known extension, this method parses it according to that extension (the file's syntax must match its extension). If the `fileBasename` 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>fileBasename</i>	a filename with or without extension
<i>options</i>	parse options

Returns

the parsed configuration

7.7.2.15 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

7.7.2.16 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

7.7.2.17 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

7.7.2.18 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)

7.7.2.19 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

7.7.2.20 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)

7.7.2.21 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

7.7.2.22 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](#).

7.7.2.23 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](#).

7.7.2.24 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

7.7.2.25 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

7.7.2.26 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

7.7.3 Friends And Related Symbol Documentation

7.7.3.1 config_object

```
friend class config_object [friend]
```

Definition at line 173 of file [config.hpp](#).

7.7.3.2 config_parseable

```
friend class config_parseable [friend]
```

Definition at line 175 of file [config.hpp](#).

7.7.3.3 config_value

```
friend class config_value [friend]
```

Definition at line 174 of file [config.hpp](#).

7.7.3.4 parseable

```
friend class parseable [friend]
```

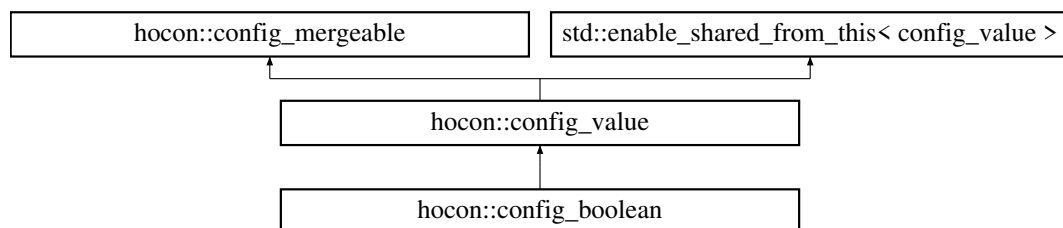
Definition at line 176 of file [config.hpp](#).

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 [config_value](#) based on this one, but with the given origin.
- 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 class 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 equals()

```
template<typename T >
static bool hocon::config_value::equals (
    config_value const & other,
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.8.3.4 new_copy()

```
shared_value hocon::config_boolean::new_copy (
    shared_origin ) const [override], [protected], [virtual]
```

Implements [hocon::config_value](#).

7.8.3.5 operator==()

```
bool hocon::config_boolean::operator== (
    config_value const & other) const [override], [virtual]
```

Implements [hocon::config_value](#).

7.8.3.6 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

7.8.3.7 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](#).

7.8.3.8 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

7.8.3.9 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

7.8.3.10 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](#).

7.8.3.11 transform_to_string()

```
std::string hocon::config_boolean::transform_to_string () const [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.8.3.12 type_name()

```
static char const * hocon::config_value::type_name (  
    type t) [inline], [static], [inherited]
```

Definition at line 59 of file [config_value.hpp](#).

7.8.3.13 unwrapped()

```
unwrapped_value hocon::config_boolean::unwrapped () const [override], [virtual]
```

Implements [hocon::config_value](#).

7.8.3.14 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](#).

7.8.3.15 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](#).

7.8.3.16 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](#).

7.8.3.17 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:

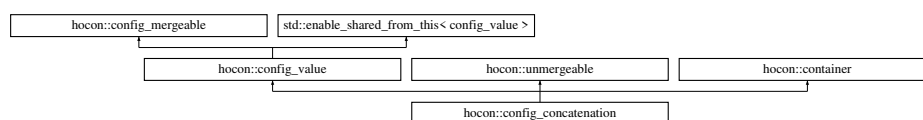
- internal/values/config_boolean.hpp

7.9 hocon::config_concatenation Class Reference

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

```
#include <config_concatenation.hpp>
```

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 `config_value` based on this one, but with the given origin.
- 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 class 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 equals()

```
template<typename T >
static bool hocon::config_value::equals (
    config_value const & other,
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.9.3.4 get_resolve_status()

```
resolve_status hocon::config_concatenation::get_resolve_status () const [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.9.3.5 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](#).

7.9.3.6 ignores_fallbacks()

```
bool hocon::config_concatenation::ignores_fallbacks () const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.9.3.7 new_copy()

```
shared_value hocon::config_concatenation::new_copy (
    shared_origin origin) const [override], [protected], [virtual]
```

Implements [hocon::config_value](#).

7.9.3.8 operator==(

```
bool hocon::config_concatenation::operator== (
    config_value const & other) const [override], [virtual]
```

Implements [hocon::config_value](#).

7.9.3.9 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

7.9.3.10 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](#).

7.9.3.11 render() [1/3]

```
void hocon::config_concatenation::render (
    std::string & result,
    int indent,
    bool at_root,
    config_render_options options) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.9.3.12 render() [2/3]

```
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

7.9.3.13 render() [3/3]

```
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

7.9.3.14 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](#).

7.9.3.15 resolve_substitutions()

```
resolve_result< shared_value > hocon::config_concatenation::resolve_substitutions (
    resolve_context const & context,
    resolve_source const & source) const [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.9.3.16 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](#).

7.9.3.17 type_name()

```
static char const * hocon::config_value::type_name (
    type t) [inline], [static], [inherited]
```

Definition at line 59 of file [config_value.hpp](#).

7.9.3.18 unmerged_values()

```
std::vector< shared_value > hocon::config_concatenation::unmerged_values () const [override],
[virtual]
```

Implements [hocon::unmergeable](#).

7.9.3.19 unwrapped()

```
unwrapped_value hocon::config_concatenation::unwrapped () const [override], [virtual]
```

Implements [hocon::config_value](#).

7.9.3.20 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](#).

7.9.3.21 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](#).

7.9.3.22 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](#).

7.9.3.23 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 `config_value` based on this one, but with the given origin.
- 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 class 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 equals()

```
template<typename T >
static bool hocon::config_value::equals (
    config_value const & other,
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.10.3.4 get_resolve_status()

```
resolve_status hocon::config_delayed_merge::get_resolve_status () const [inline], [override],
[virtual]
```

Reimplemented from [hocon::config_value](#).

Definition at line 30 of file [config_delayed_merge.hpp](#).

7.10.3.5 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](#).

7.10.3.6 ignores_fallbacks()

```
bool hocon::config_delayed_merge::ignores_fallbacks () const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.10.3.7 make_replacement()

```
shared_value hocon::config_delayed_merge::make_replacement (
    resolve_context const & context,
    int skipping) const [override], [virtual]
```

Implements [hocon::replaceable_merge_stack](#).

7.10.3.8 new_copy()

```
shared_value hocon::config_delayed_merge::new_copy (
    shared_origin ) const [override], [protected], [virtual]
```

Implements [hocon::config_value](#).

7.10.3.9 operator==(

```
bool hocon::config_delayed_merge::operator== (
    config_value const & other) const [override], [virtual]
```

Implements [hocon::config_value](#).

7.10.3.10 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

7.10.3.11 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](#).

7.10.3.12 render() [1/4]

```
virtual void hocon::config_delayed_merge::render (
    std::string & result,
    int indent,
    bool at_root,
    config_render_options options) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.10.3.13 render() [2/4]

```
virtual void hocon::config_delayed_merge::render (
    std::string & result,
    int indent,
    bool at_root,
    std::string const & at_key,
    config_render_options options) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.10.3.14 render() [3/4]

```
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

7.10.3.15 render() [4/4]

```
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

7.10.3.16 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](#).

7.10.3.17 resolve_substitutions()

```
resolve_result< shared_value > hocon::config_delayed_merge::resolve_substitutions (
    resolve_context const & context,
    resolve_source const & source) const [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.10.3.18 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](#).

7.10.3.19 type_name()

```
static char const * hocon::config_value::type_name (
    type t) [inline], [static], [inherited]
```

Definition at line 59 of file [config_value.hpp](#).

7.10.3.20 unmerged_values()

```
std::vector< shared_value > hocon::config_delayed_merge::unmerged_values () const [override],  
[virtual]
```

Implements [hocon::unmergeable](#).

7.10.3.21 unwrapped()

```
unwrapped_value hocon::config_delayed_merge::unwrapped () const [override], [virtual]
```

Implements [hocon::config_value](#).

7.10.3.22 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](#).

7.10.3.23 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](#).

7.10.3.24 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](#).

7.10.3.25 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

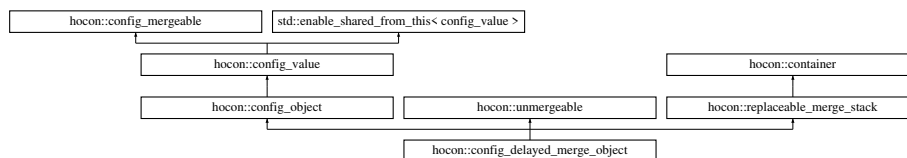
the new [config_value](#) with the given origin

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 `config_value` based on this one, but with the given origin.
- 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 Typedef Documentation

7.11.2.1 iterator

```
using hocon::config_object::iterator = std::unordered_map<std::string, shared_value>::const_iterator [inherited]
```

Definition at line 55 of file [config_object.hpp](#).

7.11.3 Member Enumeration Documentation

7.11.3.1 type

```
enum class 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](#).

7.11.4 Member Function Documentation

7.11.4.1 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.

7.11.4.2 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.

7.11.4.3 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

config_exception	if can't figure out key's value (or existence) without more resolving
----------------------------------	---

Implements [hocon::config_object](#).

7.11.4.4 begin()

```
iterator hocon::config_delayed_merge_object::begin () const [inline], [override], [virtual]
```

Implements [hocon::config_object](#).

Definition at line 30 of file [config_delayed_merge_object.hpp](#).

7.11.4.5 construct_delayed_merge()

```
shared_value hocon::config_object::construct_delayed_merge (
    shared_origin origin,
    std::vector< shared_value > stack) const [override], [protected], [virtual],
[inherited]
```

Reimplemented from [hocon::config_value](#).

7.11.4.6 end()

```
iterator hocon::config_delayed_merge_object::end () const [inline], [override], [virtual]
```

Implements [hocon::config_object](#).

Definition at line 31 of file [config_delayed_merge_object.hpp](#).

7.11.4.7 entry_set()

```
std::unordered_map< std::string, shared_value > const & hocon::config_delayed_merge_object←
::entry_set () const [override], [protected], [virtual]
```

Implements [hocon::config_object](#).

7.11.4.8 equals()

```
template<typename T >
static bool hocon::config_value::equals (
    config_value const & other,
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.11.4.9 get()

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

Implements [hocon::config_object](#).

Definition at line 29 of file [config_delayed_merge_object.hpp](#).

7.11.4.10 get_resolve_status()

```
resolve_status hocon::config_delayed_merge_object::get_resolve_status () const [inline],
[override], [virtual]
```

Reimplemented from [hocon::config_value](#).

Definition at line 21 of file [config_delayed_merge_object.hpp](#).

7.11.4.11 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](#).

7.11.4.12 ignores_fallbacks()

```
bool hocon::config_delayed_merge_object::ignores_fallbacks () const [override], [protected],
[virtual]
```

Reimplemented from [hocon::config_value](#).

7.11.4.13 is_empty()

```
bool hocon::config_delayed_merge_object::is_empty () const [inline], [override], [virtual]
```

Implements [hocon::config_object](#).

Definition at line 26 of file [config_delayed_merge_object.hpp](#).

7.11.4.14 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](#).

7.11.4.15 make_replacement()

```
shared_value hocon::config_delayed_merge_object::make_replacement (
    resolve_context const & context,
    int skipping) const [override], [virtual]
```

Implements [hocon::replaceable_merge_stack](#).

7.11.4.16 new_copy() [1/2]

```
shared_object hocon::config_delayed_merge_object::new_copy (
    resolve_status const & status,
    shared_origin origin) const [override], [protected], [virtual]
```

Implements [hocon::config_object](#).

7.11.4.17 new_copy() [2/2]

```
shared_value hocon::config_object::new_copy (
    shared_origin origin) const [override], [protected], [virtual], [inherited]
```

Implements [hocon::config_value](#).

7.11.4.18 operator==(

```
bool hocon::config_delayed_merge_object::operator== (
    config_value const & other) const [override], [virtual]
```

Implements [hocon::config_value](#).

7.11.4.19 operator[]()

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

Implements [hocon::config_object](#).

Definition at line 28 of file [config_delayed_merge_object.hpp](#).

7.11.4.20 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

7.11.4.21 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](#).

7.11.4.22 render() [1/4]

```
virtual void hocon::config_delayed_merge_object::render (
    std::string & result,
    int indent,
    bool at_root,
    config_render_options options) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.11.4.23 render() [2/4]

```
virtual void hocon::config_delayed_merge_object::render (
    std::string & result,
    int indent,
    bool at_root,
    std::string const & at_key,
    config_render_options options) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.11.4.24 render() [3/4]

```
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

7.11.4.25 render() [4/4]

```
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

7.11.4.26 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](#).

7.11.4.27 resolve_substitutions()

```
resolve_result< shared_value > hocon::config_delayed_merge_object::resolve_substitutions (
    resolve_context const & context,
    resolve_source const & source) const [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.11.4.28 size()

```
size_t hocon::config_delayed_merge_object::size () const [inline], [override], [virtual]
```

Implements [hocon::config_object](#).

Definition at line 27 of file [config_delayed_merge_object.hpp](#).

7.11.4.29 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

7.11.4.30 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`.

7.11.4.31 type_name()

```
static char const * hocon::config_value::type_name (  
    type t) [inline], [static], [inherited]
```

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

7.11.4.32 unmerged_values()

```
std::vector< shared_value > hocon::config_delayed_merge_object::unmerged_values () const  
[override], [virtual]
```

Implements `hocon::unmergeable`.

7.11.4.33 unwrapped()

```
unwrapped_value hocon::config_delayed_merge_object::unwrapped () const [override], [virtual]
```

Implements `hocon::config_value`.

7.11.4.34 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](#).

7.11.4.35 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](#).

7.11.4.36 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](#).

7.11.4.37 with_only_path()

```
shared_object hocon::config_delayed_merge_object::with_only_path (
    path raw_path) const [override], [protected], [virtual]
```

Implements [hocon::config_object](#).

7.11.4.38 with_only_path_or_null()

```
shared_object hocon::config_delayed_merge_object::with_only_path_or_null (
    path raw_path) const [override], [protected], [virtual]
```

Implements [hocon::config_object](#).

7.11.4.39 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

7.11.4.40 with_value() [1/2]

```
shared_object hocon::config_delayed_merge_object::with_value (
    path raw_path,
    shared_value value) const [override], [virtual]
```

Implements [hocon::config_object](#).

7.11.4.41 with_value() [2/2]

```
shared_object hocon::config_delayed_merge_object::with_value (
    std::string key,
    shared_value value) const [override], [virtual]
```

Implements [hocon::config_object](#).

7.11.4.42 without_path()

```
shared_object hocon::config_delayed_merge_object::without_path (
    path raw_path) const [override], [protected], [virtual]
```

Implements [hocon::config_object](#).

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

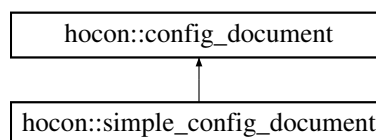
- internal/values/config_delayed_merge_object.hpp

7.12 hocon::config_document Class Reference

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

```
#include <config_document.hpp>
```

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 config_document .

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 config_node 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 newValue but the document was parsed with JSON, the first value in the concatenation will be parsed and inserted into the config_document .

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.

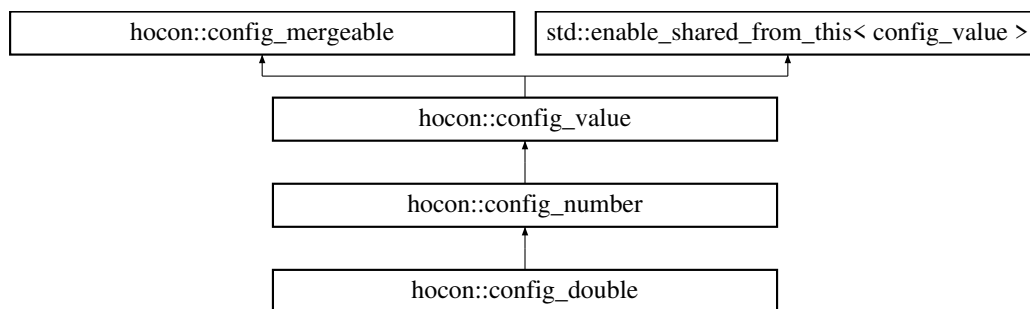
Implemented in [hocon::simple_config_document](#).

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 `config_value` based on this one, but with the given origin.
- 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 class 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 double_value()

```
double hocon::config_double::double_value () const [override], [virtual]
```

Implements [hocon::config_number](#).

7.13.3.4 equals()

```
template<typename T >
static bool hocon::config_value::equals (
    config_value const & other,
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.13.3.5 long_value()

```
int64_t hocon::config_double::long_value () const [override], [virtual]
```

Implements [hocon::config_number](#).

7.13.3.6 new_copy()

```
shared_value hocon::config_double::new_copy (
    shared_origin ) const [override], [protected], [virtual]
```

Implements [hocon::config_value](#).

7.13.3.7 operator==()

```
bool hocon::config_number::operator== (
    config_value const & other) const [override], [virtual], [inherited]
```

Implements [hocon::config_value](#).

7.13.3.8 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

7.13.3.9 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](#).

7.13.3.10 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

7.13.3.11 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

7.13.3.12 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](#).

7.13.3.13 transform_to_string()

```
std::string hocon::config_double::transform_to_string () const [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.13.3.14 type_name()

```
static char const * hocon::config_value::type_name (
    type t) [inline], [static], [inherited]
```

Definition at line 59 of file [config_value.hpp](#).

7.13.3.15 unwrapped()

```
unwrapped_value hocon::config_double::unwrapped () const [override], [virtual]
```

Implements [hocon::config_value](#).

7.13.3.16 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](#).

7.13.3.17 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](#).

7.13.3.18 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](#).

7.13.3.19 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

7.13.4 Member Data Documentation**7.13.4.1 _original_text**

```
std::string hocon::config_number::_original_text [protected], [inherited]
```

Definition at line 35 of file [config_number.hpp](#).

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

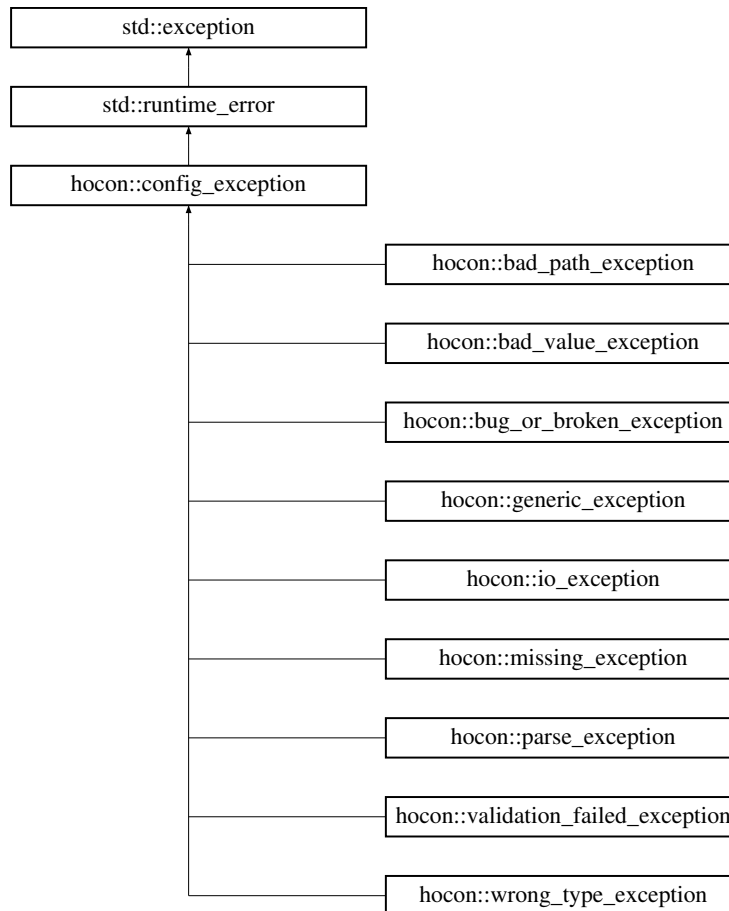
- `internal/values/config_double.hpp`

7.14 hocon::config_exception Struct Reference

All exceptions thrown by the library are subclasses of [config_exception](#).

```
#include <config_exception.hpp>
```

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](#).

7.14.2 Constructor & Destructor Documentation

7.14.2.1 `config_exception()` [1/3]

```
hocon::config_exception::config_exception (
    config\_origin const & origin,
    std::string const & message) [inline]
```

Definition at line 15 of file [config_exception.hpp](#).

7.14.2.2 `config_exception()` [2/3]

```
hocon::config_exception::config_exception (
    std::string const & message) [inline]
```

Definition at line 17 of file [config_exception.hpp](#).

7.14.2.3 `config_exception()` [3/3]

```
hocon::config_exception::config_exception (
    std::string const & message,
    std::exception const & e) [inline]
```

Definition at line 19 of file [config_exception.hpp](#).

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

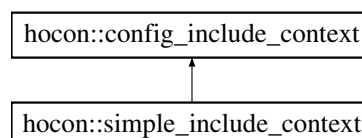
- [hocon/config_exception.hpp](#)

7.15 `hocon::config_include_context` Class Reference

Context provided to a [config_includer](#); this interface is only useful inside a `config_includer` implementation, and is not intended for apps to implement.

```
#include <config_include_context.hpp>
```

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 [config_includer](#) 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 Constructor & Destructor Documentation

7.15.2.1 [config_include_context\(\)](#)

```
hocon::config_include_context::config_include_context () [inline]
```

Definition at line 23 of file [config_include_context.hpp](#).

7.15.3 Member Function Documentation

7.15.3.1 [get_cur_dir\(\)](#)

```
std::string hocon::config_include_context::get_cur_dir () const [inline]
```

Definition at line 55 of file [config_include_context.hpp](#).

7.15.3.2 `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](#).

7.15.3.3 `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](#).

7.15.3.4 `set_cur_dir()`

```
void hocon::config_include_context::set_cur_dir (
    std::string dir) const [inline]
```

Definition at line 51 of file [config_include_context.hpp](#).

7.15.4 Member Data Documentation

7.15.4.1 `_cur_dir`

`std::shared_ptr<std::string> hocon::config_include_context::_cur_dir` [protected]

Definition at line 60 of file [config_include_context.hpp](#).

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

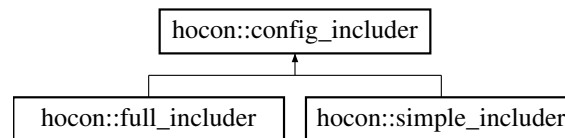
- [hocon/config_include_context.hpp](#)

7.16 hocon::config_includer Class Reference

Implement this interface and provide an instance to [config_parse_options.set_includer\(\)](#) to customize handling of `include` statements in config files.

```
#include <config_includer.hpp>
```

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:

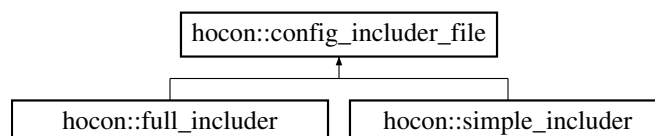
- [hocon/config_includer.hpp](#)

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 `include file("filename")` syntax.

```
#include <config_includer_file.hpp>
```

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](#)

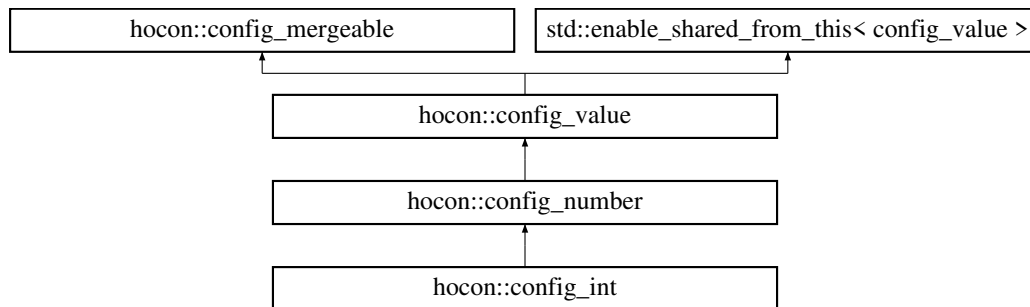
Implemented in [hocon::simple_includer](#).

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 [config_value](#) based on this one, but with the given origin.*
- 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 class 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 double_value()

```
double hocon::config_int::double_value () const [override], [virtual]
```

Implements [hocon::config_number](#).

7.18.3.4 equals()

```
template<typename T >
static bool hocon::config_value::equals (
    config_value const & other,
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.18.3.5 long_value()

```
int64_t hocon::config_int::long_value () const [override], [virtual]
```

Implements [hocon::config_number](#).

7.18.3.6 new_copy()

```
shared_value hocon::config_int::new_copy (
    shared_origin ) const [override], [protected], [virtual]
```

Implements [hocon::config_value](#).

7.18.3.7 operator==()

```
bool hocon::config_number::operator== (
    config_value const & other) const [override], [virtual], [inherited]
```

Implements [hocon::config_value](#).

7.18.3.8 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

7.18.3.9 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](#).

7.18.3.10 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

7.18.3.11 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

7.18.3.12 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](#).

7.18.3.13 transform_to_string()

```
std::string hocon::config_int::transform_to_string () const [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.18.3.14 type_name()

```
static char const * hocon::config_value::type_name (
    type t) [inline], [static], [inherited]
```

Definition at line 59 of file [config_value.hpp](#).

7.18.3.15 unwrapped()

```
unwrapped_value hocon::config_int::unwrapped () const [override], [virtual]
```

Implements [hocon::config_value](#).

7.18.3.16 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](#).

7.18.3.17 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](#).

7.18.3.18 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](#).

7.18.3.19 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

7.18.4 Member Data Documentation**7.18.4.1 _original_text**

```
std::string hocon::config_number::_original_text [protected], [inherited]
```

Definition at line 35 of file [config_number.hpp](#).

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

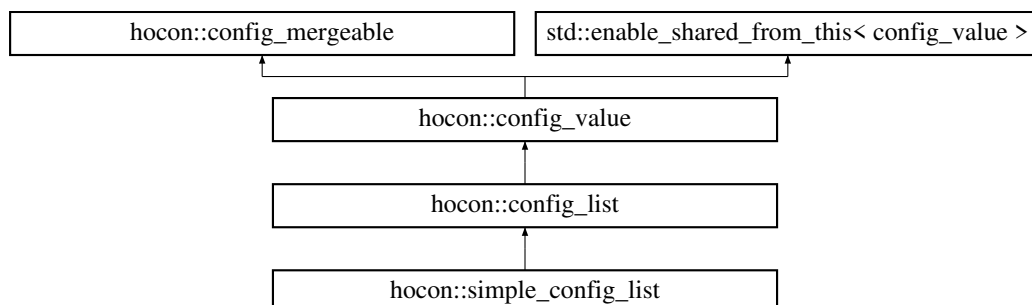
- `internal/values/config_int.hpp`

7.19 hocon::config_list Class Reference

Subtype of `ConfigValue` representing a list value, as in JSON's `[1, 2, 3]` syntax.

```
#include <config_list.hpp>
```

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 `config_value` based on this one, but with the given origin.
- 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 Typedef Documentation

7.19.2.1 iterator

```
using hocon::config_list::iterator = std::vector<shared_value>::const_iterator
```

Definition at line 42 of file [config_list.hpp](#).

7.19.3 Member Enumeration Documentation

7.19.3.1 type

```
enum class 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](#).

7.19.4 Constructor & Destructor Documentation

7.19.4.1 config_list()

```
hocon::config_list::config_list (
    shared_origin origin) [inline]
```

Definition at line 39 of file [config_list.hpp](#).

7.19.5 Member Function Documentation

7.19.5.1 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.

7.19.5.2 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.

7.19.5.3 equals()

```
template<typename T >
static bool hocon::config_value::equals (
    config_value const & other,
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.19.5.4 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

7.19.5.5 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](#).

7.19.5.6 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

7.19.5.7 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

7.19.5.8 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](#).

7.19.5.9 type_name()

```
static char const * hocon::config_value::type_name (  
    type t) [inline], [static], [inherited]
```

Definition at line 59 of file [config_value.hpp](#).

7.19.5.10 unwrapped()

```
virtual unwrapped_value hocon::config_list::unwrapped () const [pure virtual]
```

Implements [hocon::config_value](#).

7.19.5.11 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::config_boolean](#), [hocon::config_concatenation](#), [hocon::config_delayed_merge](#), [hocon::config_null](#), [hocon::config_number](#), [hocon::config_object](#), [hocon::config_reference](#), [hocon::config_string](#), and [hocon::simple_config_list](#).

7.19.5.12 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](#).

7.19.5.13 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](#).

7.19.5.14 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

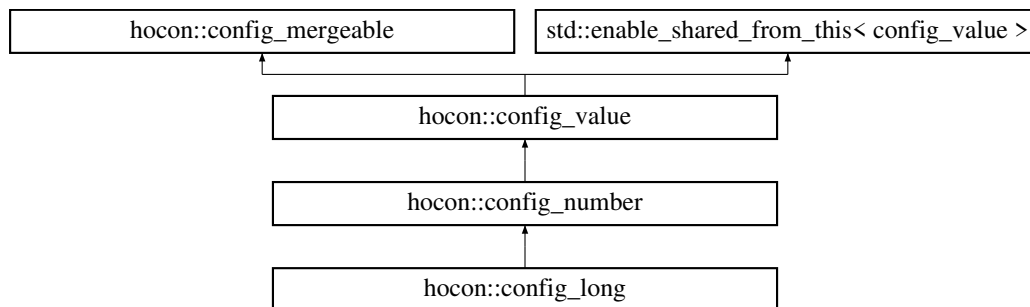
the new `config_value` with the given origin

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

- `hocon/config_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 *config_value* based on this one, but with the given origin.
- 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 class 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 `double_value()`

```
double hocon::config_long::double_value () const [override], [virtual]
```

Implements [hocon::config_number](#).

7.20.3.4 `equals()`

```
template<typename T >
static bool hocon::config_value::equals (
    config_value const & other,
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.20.3.5 `long_value()`

```
int64_t hocon::config_long::long_value () const [override], [virtual]
```

Implements [hocon::config_number](#).

7.20.3.6 `new_copy()`

```
shared_value hocon::config_long::new_copy (
    shared_origin ) const [override], [protected], [virtual]
```

Implements [hocon::config_value](#).

7.20.3.7 `operator==()`

```
bool hocon::config_number::operator== (
    config_value const & other) const [override], [virtual], [inherited]
```

Implements [hocon::config_value](#).

7.20.3.8 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

7.20.3.9 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](#).

7.20.3.10 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

7.20.3.11 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

7.20.3.12 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](#).

7.20.3.13 transform_to_string()

```
std::string hocon::config_long::transform_to_string () const [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.20.3.14 type_name()

```
static char const * hocon::config_value::type_name (
    type t) [inline], [static], [inherited]
```

Definition at line 59 of file [config_value.hpp](#).

7.20.3.15 unwrapped()

```
unwrapped_value hocon::config_long::unwrapped () const [override], [virtual]
```

Implements [hocon::config_value](#).

7.20.3.16 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](#).

7.20.3.17 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](#).

7.20.3.18 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](#).

7.20.3.19 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

7.20.4 Member Data Documentation**7.20.4.1 _original_text**

```
std::string hocon::config_number::_original_text [protected], [inherited]
```

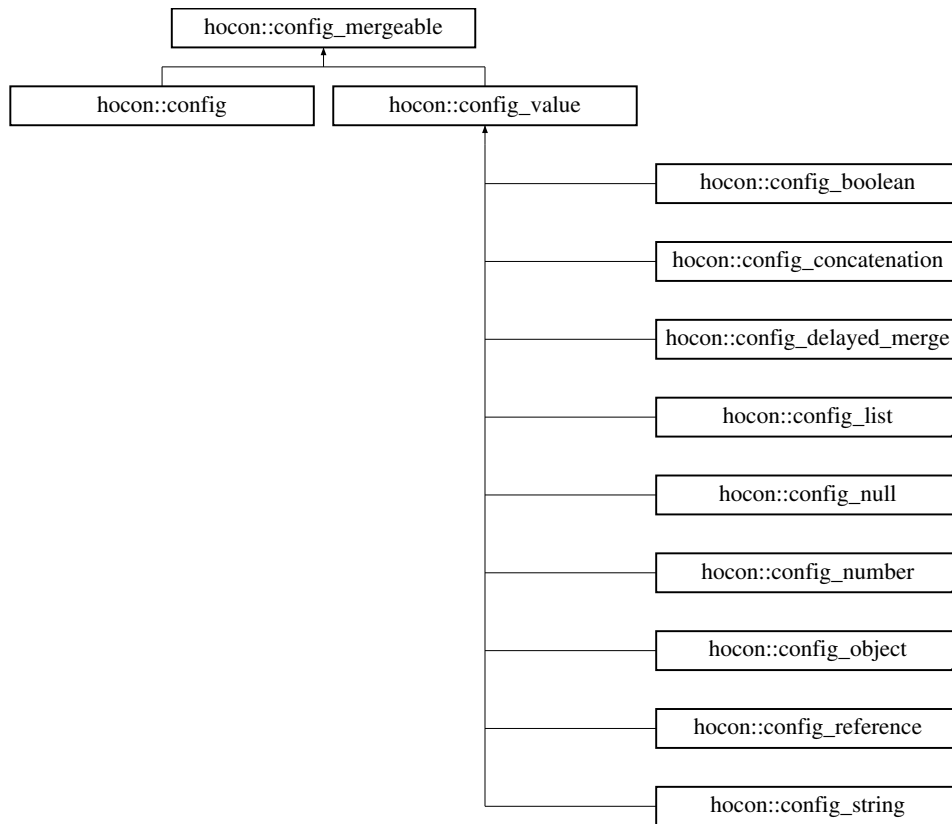
Definition at line 35 of file [config_number.hpp](#).

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](#), and [hocon::config_value](#).

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](#), and [hocon::config_value](#).

7.21.3 Friends And Related Symbol Documentation

7.21.3.1 config_value

```
friend class config_value [friend]
```

Definition at line 9 of file [config_mergeable.hpp](#).

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

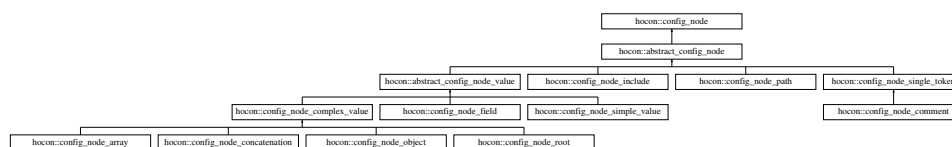
- [hocon/config_mergeable.hpp](#)

7.22 hocon::config_node Class Reference

A node in the syntax tree for a HOCON or JSON document.

```
#include <config_node.hpp>
```

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

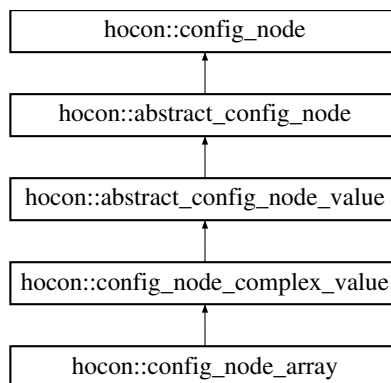
Implemented in [hocon::abstract_config_node](#).

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 get_tokens()

```
token_list hocon::config_node_complex_value::get_tokens () const [override], [virtual], [inherited]
```

Implements [hocon::abstract_config_node](#).

7.23.2.2 new_node()

```
std::shared_ptr< const config_node_complex_value > hocon::config_node_array::new_node (
    shared_node_list nodes) const [override], [virtual]
```

Implements [hocon::config_node_complex_value](#).

7.23.2.3 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

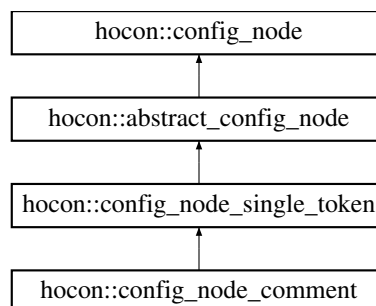
Implements [hocon::config_node](#).

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 get_tokens()

```
token_list hocon::config_node_single_token::get_tokens () const [override], [virtual], [inherited]
```

Implements [hocon::abstract_config_node](#).

7.24.2.2 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

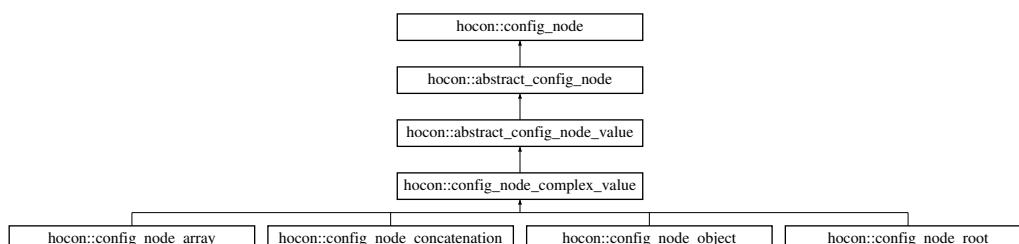
Implements [hocon::config_node](#).

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 get_tokens()

```
token_list hocon::config_node_complex_value::get_tokens () const [override], [virtual]
```

Implements [hocon::abstract_config_node](#).

7.25.2.2 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

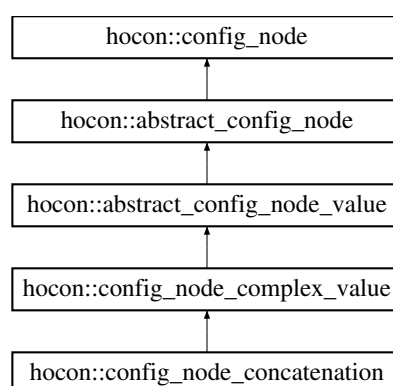
Implements [hocon::config_node](#).

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 get_tokens()

```
token_list hocon::config_node_complex_value::get_tokens () const [override], [virtual], [inherited]
```

Implements [hocon::abstract_config_node](#).

7.26.2.2 new_node()

```
std::shared_ptr< const config\_node\_complex\_value > hocon::config_node_concatenation::new_node
(
    shared_node_list nodes) const [override], [virtual]
```

Implements [hocon::config_node_complex_value](#).

7.26.2.3 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:

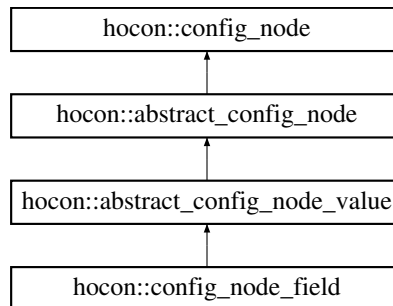
- internal/nodes/config_node_concatenation.hpp

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.

```
#include <config_node_field.hpp>
```

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 get_tokens()

```
token_list hocon::config_node_field::get_tokens () const [override], [virtual]
```

Implements [hocon::abstract_config_node](#).

7.27.2.2 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:

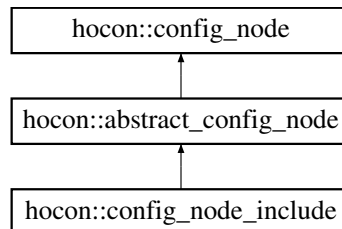
- internal/nodes/config_node_field.hpp

7.28 hocon::config_node_include Class Reference

Represents an include statement of the form "include include_kind(include_path)".

```
#include <config_node_include.hpp>
```

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 get_tokens()

```
token_list hocon::config_node_include::get_tokens () const [override], [virtual]
```

Implements [hocon::abstract_config_node](#).

7.28.2.2 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

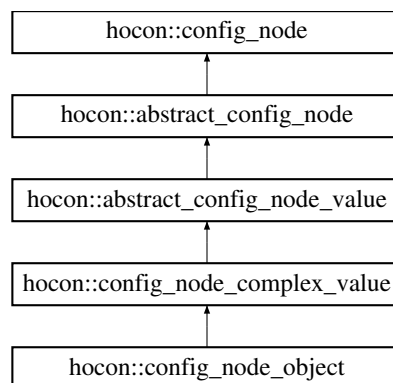
Implements [hocon::config_node](#).

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 get_tokens()

```
token_list hocon::config_node_complex_value::get_tokens () const [override], [virtual], [inherited]
```

Implements [hocon::abstract_config_node](#).

7.29.2.2 new_node()

```
std::shared_ptr< const config\_node\_complex\_value > hocon::config_node_object::new_node (
    shared_node_list nodes) const [override], [virtual]
```

Implements [hocon::config_node_complex_value](#).

7.29.2.3 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

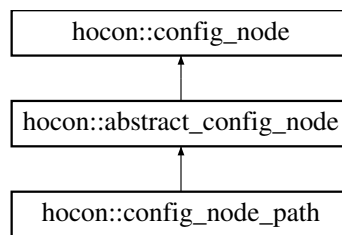
Implements [hocon::config_node](#).

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 get_tokens()

```
token_list hocon::config_node_path::get_tokens () const [override], [virtual]
```

Implements [hocon::abstract_config_node](#).

7.30.2.2 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

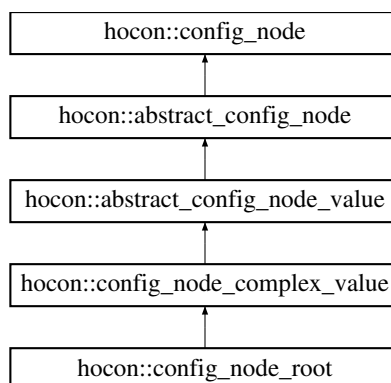
Implements [hocon::config_node](#).

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 get_tokens()

```
token_list hocon::config_node_complex_value::get_tokens () const [override], [virtual], [inherited]
```

Implements [hocon::abstract_config_node](#).

7.31.2.2 new_node()

```
std::shared_ptr< const config_node_complex_value > hocon::config_node_root::new_node (
    shared_node_list nodes) const [override], [virtual]
```

Implements [hocon::config_node_complex_value](#).

7.31.2.3 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

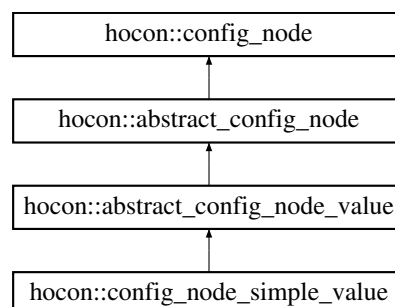
Implements [hocon::config_node](#).

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 get_tokens()

```
token_list hocon::config_node_simple_value::get_tokens () const [override], [virtual]
```

Implements [hocon::abstract_config_node](#).

7.32.2.2 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

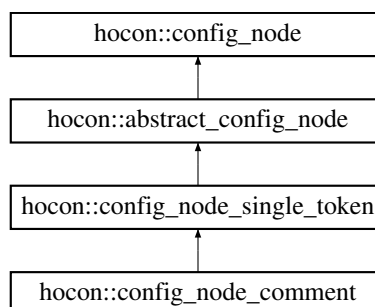
Implements [hocon::config_node](#).

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 get_tokens()

```
token_list hocon::config_node_single_token::get_tokens () const [override], [virtual]
```

Implements [hocon::abstract_config_node](#).

7.33.2.2 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:

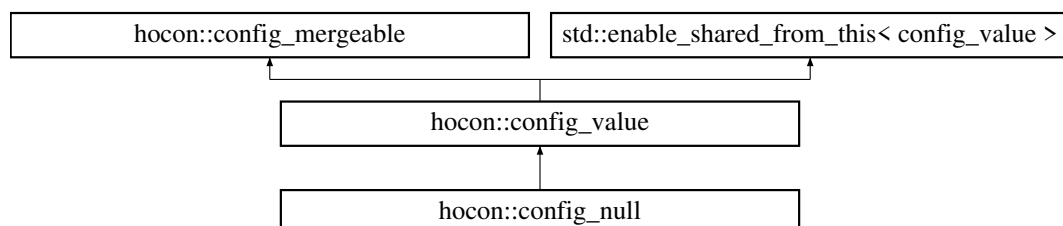
- internal/nodes/config_node_single_token.hpp

7.34 hocon::config_null Class Reference

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

```
#include <config_null.hpp>
```

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 *config_value* based on this one, but with the given origin.
- `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 class 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 equals()

```
template<typename T >
static bool hocon::config_value::equals (
    config_value const & other,
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.34.3.4 new_copy()

```
shared_value hocon::config_null::new_copy (
    shared_origin ) const [override], [protected], [virtual]
```

Implements [hocon::config_value](#).

7.34.3.5 operator==()

```
bool hocon::config_null::operator== (
    config_value const & other) const [override], [virtual]
```

Implements [hocon::config_value](#).

7.34.3.6 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

7.34.3.7 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](#).

7.34.3.8 render() [1/3]

```
void hocon::config_null::render (
    std::string & result,
    int indent,
    bool at_root,
    config_render_options options) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.34.3.9 render() [2/3]

```
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

7.34.3.10 render() [3/3]

```
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

7.34.3.11 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](#).

7.34.3.12 transform_to_string()

```
std::string hocon::config_null::transform_to_string () const [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.34.3.13 type_name()

```
static char const * hocon::config_value::type_name (  
    type t) [inline], [static], [inherited]
```

Definition at line 59 of file [config_value.hpp](#).

7.34.3.14 unwrapped()

```
unwrapped_value hocon::config_null::unwrapped () const [override], [virtual]
```

Implements [hocon::config_value](#).

7.34.3.15 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](#).

7.34.3.16 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](#).

7.34.3.17 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](#).

7.34.3.18 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

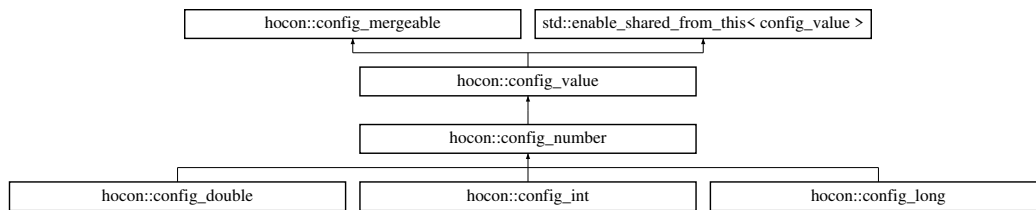
the new `config_value` with the given origin

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

- internal/values/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
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 `config_value` based on this one, but with the given origin.
- 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 class 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 equals()

```
template<typename T >
static bool hocon::config_value::equals (
    config_value const & other,
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.35.3.4 operator==()

```
bool hocon::config_number::operator== (
    config_value const & other) const [override], [virtual]
```

Implements [hocon::config_value](#).

7.35.3.5 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

7.35.3.6 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](#).

7.35.3.7 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

7.35.3.8 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

7.35.3.9 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](#).

7.35.3.10 transform_to_string()

```
std::string hocon::config_number::transform_to_string () const [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.35.3.11 type_name()

```
static char const * hocon::config_value::type_name (  
    type t) [inline], [static], [inherited]
```

Definition at line 59 of file [config_value.hpp](#).

7.35.3.12 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](#).

7.35.3.13 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](#).

7.35.3.14 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](#).

7.35.3.15 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

7.35.4 Member Data Documentation

7.35.4.1 `_original_text`

```
std::string hocon::config_number::_original_text [protected]
```

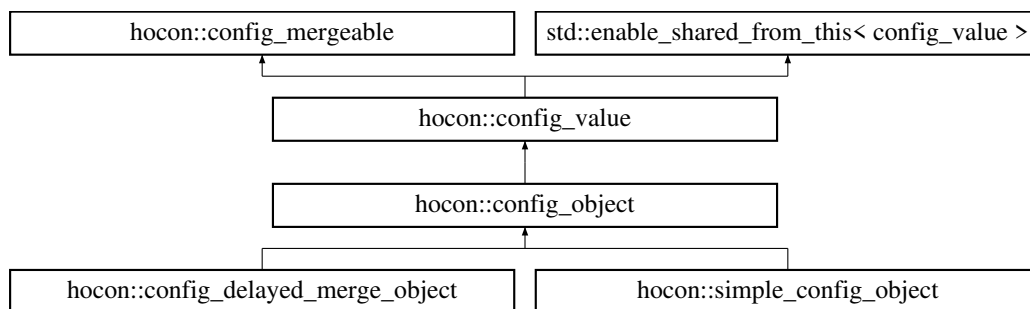
Definition at line 35 of file [config_number.hpp](#).

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 config_value based on this one, but with the given origin.
- 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 Typedef Documentation

7.36.2.1 iterator

```
using hocon::config_object::iterator = std::unordered_map<std::string, shared_value>::const_iterator
```

Definition at line 55 of file [config_object.hpp](#).

7.36.3 Member Enumeration Documentation

7.36.3.1 type

```
enum class 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](#).

7.36.4 Member Function Documentation

7.36.4.1 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.

7.36.4.2 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.

7.36.4.3 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

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

Implemented in [hocon::config_delayed_merge_object](#), and [hocon::simple_config_object](#).

7.36.4.4 construct_delayed_merge()

```
shared_value hocon::config_object::construct_delayed_merge (
    shared_origin origin,
    std::vector< shared_value > stack) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.36.4.5 equals()

```
template<typename T >
static bool hocon::config_value::equals (
    config_value const & other,
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.36.4.6 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::config_delayed_merge_object](#), and [hocon::simple_config_object](#).

7.36.4.7 new_copy()

```
shared_value hocon::config_object::new_copy (
    shared_origin origin) const [override], [protected], [virtual]
```

Implements [hocon::config_value](#).

7.36.4.8 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

7.36.4.9 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](#).

7.36.4.10 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

7.36.4.11 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

7.36.4.12 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

7.36.4.13 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](#).

7.36.4.14 type_name()

```
static char const * hocon::config_value::type_name (  
    type t) [inline], [static], [inherited]
```

Definition at line 59 of file [config_value.hpp](#).

7.36.4.15 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](#).

7.36.4.16 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](#).

7.36.4.17 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](#).

7.36.4.18 with_only_path_or_null()

```
virtual shared_object hocon::config_object::with_only_path_or_null (
    path raw_path) const [protected], [pure virtual]
```

Implemented in [hocon::simple_config_object](#).

7.36.4.19 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

7.36.5 Friends And Related Symbol Documentation

7.36.5.1 config

```
friend class config [friend]
```

Definition at line 12 of file [config_object.hpp](#).

7.36.5.2 config_delayed_merge_object

```
friend class config_delayed_merge_object [friend]
```

Definition at line 16 of file [config_object.hpp](#).

7.36.5.3 config_value

```
friend class config_value [friend]
```

Definition at line 13 of file [config_object.hpp](#).

7.36.5.4 resolve_source

```
friend class resolve_source [friend]
```

Definition at line 15 of file [config_object.hpp](#).

7.36.5.5 simple_config_object

```
friend class simple_config_object [friend]
```

Definition at line 14 of file [config_object.hpp](#).

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

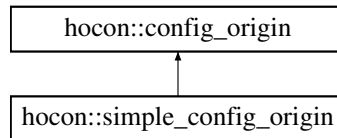
- [hocon/config_object.hpp](#)

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.

```
#include <config_origin.hpp>
```

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 ConfigOrigin based on this one, but with the given line number.
- 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 config_origin based on this one, but with the given comments.

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_values` 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.

```
#include <config_parse_options.hpp>
```

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 `config_includer#with_fallback` on the existing includer.

Parameters

<i>includer</i>	the includer to append (may not be null)
-----------------	--

Returns

new version of the parse options with different includer

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 config_origin
---------------------------	---

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 <code>nullptr</code> for best guess
---------------	---

Returns

options with the syntax set

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

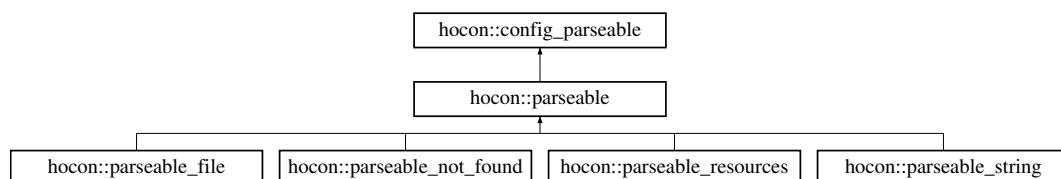
- hocon/config_parse_options.hpp

7.39 hocon::config_parseable Class Reference

An opaque handle to something that can be parsed, obtained from [config_include_context](#).

```
#include <config_parseable.hpp>
```

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 paresable 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 paresable 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

<code>options</code>	parse options, should be based on the ones from <code>config_parseable#options()</code>
----------------------	---

Returns

the parsed object

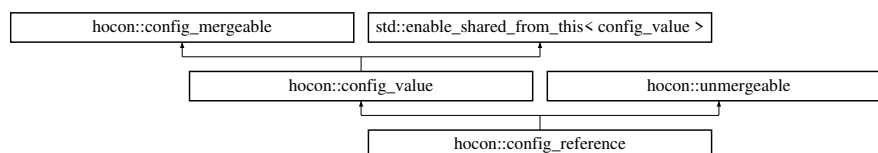
Implemented in [hocon::parseable](#).

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 [config_value](#) based on this one, but with the given origin.
- 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 class 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 equals()

```
template<typename T >
static bool hocon::config_value::equals (
    config_value const & other,
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.40.3.4 get_resolve_status()

```
resolve_status hocon::config_reference::get_resolve_status () const [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.40.3.5 ignores_fallbacks()

```
bool hocon::config_reference::ignores_fallbacks () const [inline], [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

Definition at line 31 of file [config_reference.hpp](#).

7.40.3.6 new_copy()

```
shared_value hocon::config_reference::new_copy (
    shared_origin origin) const [override], [protected], [virtual]
```

Implements [hocon::config_value](#).

7.40.3.7 operator==()

```
bool hocon::config_reference::operator== (
    config_value const & other) const [override], [virtual]
```

Implements [hocon::config_value](#).

7.40.3.8 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

7.40.3.9 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](#).

7.40.3.10 render() [1/3]

```
void hocon::config_reference::render (
    std::string & s,
    int indent,
    bool at_root,
    config_render_options options) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.40.3.11 render() [2/3]

```
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

7.40.3.12 render() [3/3]

```
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

7.40.3.13 resolve_substitutions()

```
resolve_result< shared_value > hocon::config_reference::resolve_substitutions (
    resolve_context const & context,
    resolve_source const & source) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.40.3.14 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](#).

7.40.3.15 type_name()

```
static char const * hocon::config_value::type_name (
    type t) [inline], [static], [inherited]
```

Definition at line 59 of file [config_value.hpp](#).

7.40.3.16 unmerged_values()

```
std::vector< shared_value > hocon::config_reference::unmerged_values () const [override],
[virtual]
```

Implements [hocon::unmergeable](#).

7.40.3.17 unwrapped()

```
unwrapped_value hocon::config_reference::unwrapped () const [override], [virtual]
```

Implements [hocon::config_value](#).

7.40.3.18 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](#).

7.40.3.19 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](#).

7.40.3.20 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](#).

7.40.3.21 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

```
#include <config_render_options.hpp>
```

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.

```
#include <config_resolve_options.hpp>
```

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)
```

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

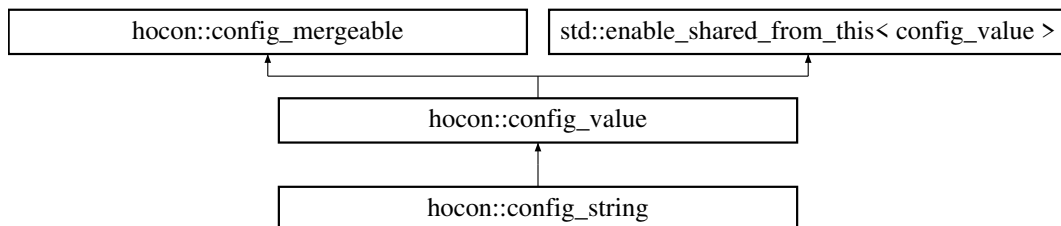
the resolve options with env variables disabled

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 [config_value](#) based on this one, but with the given origin.
- 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 class 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 equals()

```
template<typename T >
static bool hocon::config_value::equals (
    config_value const & other,
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.43.3.4 new_copy()

```
shared_value hocon::config_string::new_copy (
    shared_origin ) const [override], [protected], [virtual]
```

Implements [hocon::config_value](#).

7.43.3.5 operator==()

```
bool hocon::config_string::operator== (
    config_value const & other) const [override], [virtual]
```

Implements [hocon::config_value](#).

7.43.3.6 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

7.43.3.7 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](#).

7.43.3.8 render() [1/3]

```
void hocon::config_string::render (
    std::string & s,
    int indent,
    bool at_root,
    config_render_options options) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.43.3.9 render() [2/3]

```
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

7.43.3.10 render() [3/3]

```
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

7.43.3.11 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](#).

7.43.3.12 transform_to_string()

```
std::string hocon::config_string::transform_to_string () const [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.43.3.13 type_name()

```
static char const * hocon::config_value::type_name (
    type t) [inline], [static], [inherited]
```

Definition at line 59 of file [config_value.hpp](#).

7.43.3.14 unwrapped()

```
unwrapped_value hocon::config_string::unwrapped () const [override], [virtual]
```

Implements [hocon::config_value](#).

7.43.3.15 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](#).

7.43.3.16 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](#).

7.43.3.17 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](#).

7.43.3.18 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:

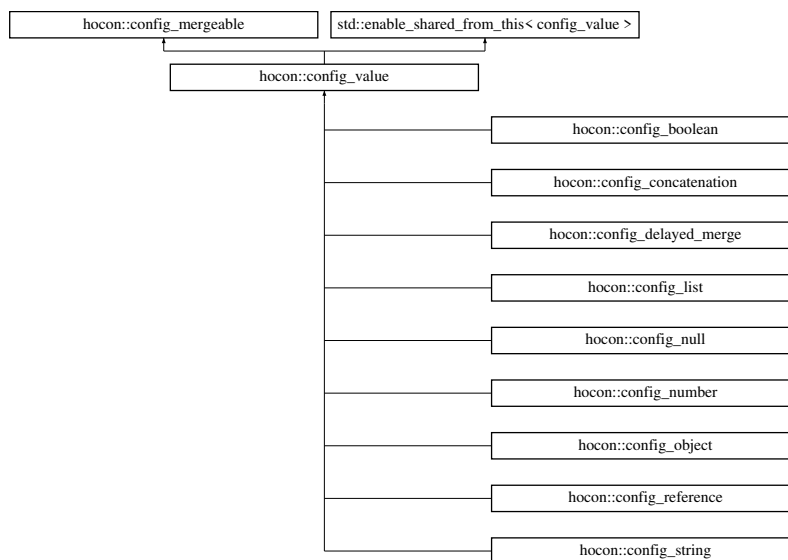
- `internal/values/config_string.hpp`

7.44 hocon::config_value Class Reference

An immutable value, following the `JSON` type schema.

```
#include <config_value.hpp>
```

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 `config_value` based on this one, but with the given origin.
- 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 class 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 equals()

```
template<typename T >  
static bool hocon::config_value::equals (  
    config_value const & other,  
    std::function< bool(T const &)> checker) [inline], [static], [protected]
```

Definition at line 250 of file [config_value.hpp](#).

7.44.3.4 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

7.44.3.5 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](#).

7.44.3.6 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

7.44.3.7 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

7.44.3.8 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](#).

7.44.3.9 type_name()

```
static char const * hocon::config_value::type_name (
    type t) [inline], [static]
```

Definition at line 59 of file [config_value.hpp](#).

7.44.3.10 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::config_boolean](#), [hocon::config_concatenation](#), [hocon::config_delayed_merge](#), [hocon::config_null](#), [hocon::config_number](#), [hocon::config_object](#), [hocon::config_reference](#), [hocon::config_string](#), and [hocon::simple_config_list](#).

7.44.3.11 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](#).

7.44.3.12 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](#).

7.44.3.13 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

7.44.4 Friends And Related Symbol Documentation

7.44.4.1 config

```
friend class config [friend]
```

Definition at line 43 of file [config_value.hpp](#).

7.44.4.2 config_concatenation

```
friend class config_concatenation [friend]
```

Definition at line 47 of file [config_value.hpp](#).

7.44.4.3 config_delayed_merge

```
friend class config_delayed_merge [friend]
```

Definition at line 49 of file [config_value.hpp](#).

7.44.4.4 config_delayed_merge_object

```
friend class config_delayed_merge_object [friend]
```

Definition at line 50 of file [config_value.hpp](#).

7.44.4.5 config_object

```
friend class config_object [friend]
```

Definition at line 44 of file [config_value.hpp](#).

7.44.4.6 default_transformer

```
friend class default_transformer [friend]
```

Definition at line 42 of file [config_value.hpp](#).

7.44.4.7 resolve_context

```
friend class resolve_context [friend]
```

Definition at line 48 of file [config_value.hpp](#).

7.44.4.8 simple_config_list

```
friend class simple_config_list [friend]
```

Definition at line 46 of file [config_value.hpp](#).

7.44.4.9 simple_config_object

```
friend class simple_config_object [friend]
```

Definition at line 45 of file [config_value.hpp](#).

7.44.4.10 token

```
friend class token [friend]
```

Definition at line 40 of file [config_value.hpp](#).

7.44.4.11 value

```
friend class value [friend]
```

Definition at line 41 of file [config_value.hpp](#).

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:

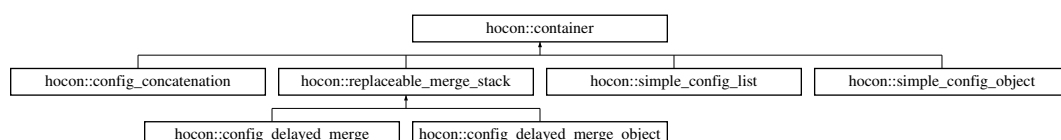
- hocon/config_value_factory.hpp

7.46 hocon::container Class Reference

An AbstractConfigValue which contains other values.

```
#include <container.hpp>
```

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::config_concatenation](#), [hocon::config_delayed_merge](#), [hocon::config_delayed_merge_object](#), [hocon::simple_config_list](#), and [hocon::simple_config_object](#).

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::config_concatenation](#), [hocon::config_delayed_merge](#), [hocon::config_delayed_merge_object](#), [hocon::simple_config_list](#), and [hocon::simple_config_object](#).

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

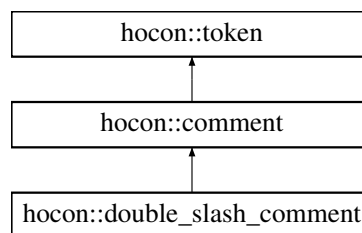
Definition at line 7 of file [default_transformer.hpp](#).

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](#).

7.48.2 Member Function Documentation

7.48.2.1 operator==()

```
bool hocon::comment::operator== (
    const token & other) const [override], [virtual], [inherited]
```

Reimplemented from [hocon::token](#).

7.48.2.2 to_string()

```
std::string hocon::comment::to_string () const [override], [virtual], [inherited]
```

Reimplemented from [hocon::token](#).

7.48.2.3 token_text()

```
std::string hocon::double_slash_comment::token_text () const [override], [virtual]
```

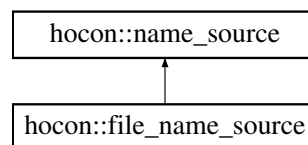
Reimplemented from [hocon::token](#).

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](#).

7.49.2 Member Function Documentation

7.49.2.1 context_initialized()

```
bool hocon::name_source::context_initialized () const [inline], [inherited]
```

Definition at line 81 of file [simple_includer.hpp](#).

7.49.2.2 get_context()

```
shared_include_context hocon::name_source::get_context () const [inline], [inherited]
```

Definition at line 77 of file [simple_includer.hpp](#).

7.49.2.3 name_to_parseable()

```
shared_parseable hocon::file_name_source::name_to_parseable (
    std::string name,
    config_parse_options parse_options) const [override], [virtual]
```

Implements [hocon::name_source](#).

7.49.2.4 set_context()

```
void hocon::name_source::set_context (
    shared_include_context context) [inline], [inherited]
```

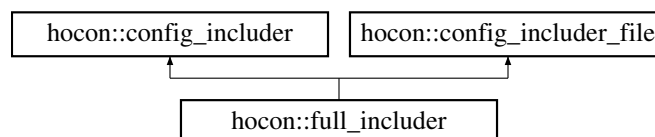
Definition at line 70 of file [simple_includer.hpp](#).

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

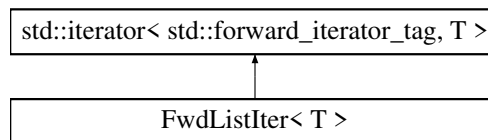
Implemented in [hocon::simple_includer](#).

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](#).

7.51.2 Constructor & Destructor Documentation

7.51.2.1 FwdListIter() [1/2]

```
template<class T>
FwdListIter< T >::FwdListIter () [inline]
```

Definition at line [120](#) of file [functional_list.hpp](#).

7.51.2.2 FwdListIter() [2/2]

```
template<class T >
FwdListIter< T >::FwdListIter (
    List< T > const & lst) [inline]
```

Definition at line 121 of file [functional_list.hpp](#).

7.51.3 Member Function Documentation

7.51.3.1 operator!=(())

```
template<class T >
bool FwdListIter< T >::operator!= (
    FwdListIter< T > const & other) [inline]
```

Definition at line 133 of file [functional_list.hpp](#).

7.51.3.2 operator*()

```
template<class T >
T FwdListIter< T >::operator* () const [inline]
```

Definition at line 123 of file [functional_list.hpp](#).

7.51.3.3 operator++()

```
template<class T >
FwdListIter & FwdListIter< T >::operator++ () [inline]
```

Definition at line 124 of file [functional_list.hpp](#).

7.51.3.4 operator==(())

```
template<class T >
bool FwdListIter< T >::operator== (
    FwdListIter< T > const & other) [inline]
```

Definition at line 129 of file [functional_list.hpp](#).

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

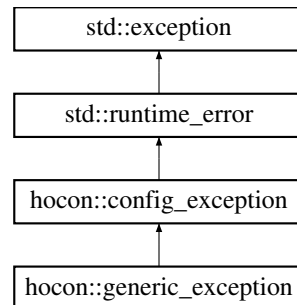
- [hocon/functional_list.hpp](#)

7.52 hocon::generic_exception Struct Reference

Exception that doesn't fall into any other category.

```
#include <config_exception.hpp>
```

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](#).

7.52.2 Constructor & Destructor Documentation

7.52.2.1 generic_exception()

```
hocon::generic_exception::generic_exception (
    std::string const & message) [inline]
```

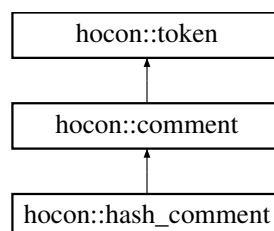
Definition at line 182 of file [config_exception.hpp](#).

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](#).

7.53.2 Member Function Documentation

7.53.2.1 operator==()

```
bool hocon::comment::operator== (
    const token & other) const    [override], [virtual], [inherited]
```

Reimplemented from [hocon::token](#).

7.53.2.2 to_string()

```
std::string hocon::comment::to_string () const    [override], [virtual], [inherited]
```

Reimplemented from [hocon::token](#).

7.53.2.3 token_text()

```
std::string hocon::hash_comment::token_text () const    [override], [virtual]
```

Reimplemented from [hocon::token](#).

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](#).

7.54.2 Member Function Documentation

7.54.2.1 operator==()

```
bool hocon::ignored_whitespace::operator== (
    const token & other) const    [override], [virtual]
```

Reimplemented from [hocon::token](#).

7.54.2.2 to_string()

```
std::string hocon::ignored_whitespace::to_string () const    [override], [virtual]
```

Reimplemented from [hocon::token](#).

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

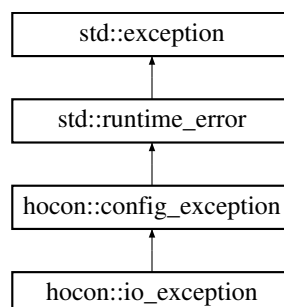
- internal/tokens.hpp

7.55 hocon::io_exception Struct Reference

Exception indicating that there was an IO error.

```
#include <config_exception.hpp>
```

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](#).

7.55.2 Constructor & Destructor Documentation

7.55.2.1 io_exception()

```
hocon::io_exception::io_exception (
    config\_origin const & origin,
    std::string const & message) [inline]
```

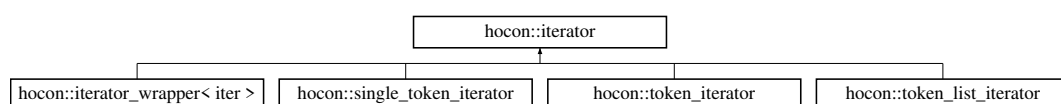
Definition at line 94 of file [config_exception.hpp](#).

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

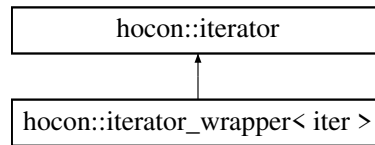
Definition at line 24 of file [tokenizer.hpp](#).

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](#).

7.57.2 Constructor & Destructor Documentation

7.57.2.1 iterator_wrapper()

```
template<typename iter >
hocon::iterator_wrapper< iter >::iterator_wrapper (
    iter begin,
    iter end) [inline]
```

Definition at line 33 of file [tokenizer.hpp](#).

7.57.3 Member Function Documentation

7.57.3.1 has_next()

```
template<typename iter >
bool hocon::iterator_wrapper< iter >::has_next () [inline], [override], [virtual]
```

Implements [hocon::iterator](#).

Definition at line 36 of file [tokenizer.hpp](#).

7.57.3.2 next()

```
template<typename iter >
shared_token hocon::iterator_wrapper< iter >::next () [inline], [override], [virtual]
```

Implements [hocon::iterator](#).

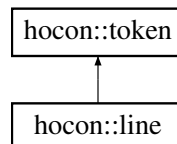
Definition at line 40 of file [tokenizer.hpp](#).

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](#).

7.58.2 Member Function Documentation

7.58.2.1 operator==()

```
bool hocon::line::operator== (
    token const & other) const [override], [virtual]
```

Reimplemented from [hocon::token](#).

7.58.2.2 to_string()

```
std::string hocon::line::to_string () const [override], [virtual]
```

Reimplemented from [hocon::token](#).

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](#).

7.59.2 Constructor & Destructor Documentation

7.59.2.1 List() [1/3]

```
template<class T >
List< T >::List () [inline]
```

Definition at line [31](#) of file [functional_list.hpp](#).

7.59.2.2 List() [2/3]

```
template<class T >
List< T >::List (
    T v,
    List< T > const & tail) [inline]
```

Definition at line 33 of file [functional_list.hpp](#).

7.59.2.3 List() [3/3]

```
template<class T >
List< T >::List (
    T v) [inline], [explicit]
```

Definition at line 36 of file [functional_list.hpp](#).

7.59.3 Member Function Documentation

7.59.3.1 forEach()

```
template<class T >
template<class F >
void List< T >::forEach (
    F f) const [inline]
```

Definition at line 89 of file [functional_list.hpp](#).

7.59.3.2 front()

```
template<class T >
T List< T >::front () const [inline]
```

Definition at line 39 of file [functional_list.hpp](#).

7.59.3.3 headCount()

```
template<class T >
int List< T >::headCount () const [inline]
```

Definition at line 101 of file [functional_list.hpp](#).

7.59.3.4 insertedAt()

```
template<class T >
List List< T >::insertedAt (
    int i,
    T v) const [inline]
```

Definition at line 59 of file [functional_list.hpp](#).

7.59.3.5 isEmpty()

```
template<class T >
bool List< T >::isEmpty () const [inline]
```

Definition at line 38 of file [functional_list.hpp](#).

7.59.3.6 member()

```
template<class T >
bool List< T >::member (
    T v) const [inline]
```

Definition at line 82 of file [functional_list.hpp](#).

7.59.3.7 popped_front()

```
template<class T >
List List< T >::popped_front () const [inline]
```

Definition at line 44 of file [functional_list.hpp](#).

7.59.3.8 pushed_front()

```
template<class T >
List List< T >::pushed_front (
    T v) const [inline]
```

Definition at line 50 of file [functional_list.hpp](#).

7.59.3.9 removed()

```
template<class T >
List List< T >::removed (
    T v) const [inline]
```

Definition at line 68 of file [functional_list.hpp](#).

7.59.3.10 removed1()

```
template<class T >
List List< T >::removed1 (
    T v) const [inline]
```

Definition at line 75 of file [functional_list.hpp](#).

7.59.3.11 take()

```
template<class T >
List List< T >::take (
    int n) [inline]
```

Definition at line 54 of file [functional_list.hpp](#).

7.59.4 Friends And Related Symbol Documentation

7.59.4.1 FwdListIter< T >

```
template<class T >
friend class FwdListIter< T > [friend]
```

Definition at line 89 of file [functional_list.hpp](#).

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

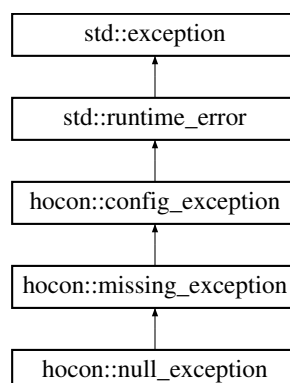
- [hocon/functional_list.hpp](#)

7.60 hocon::missing_exception Struct Reference

Exception indicates that the setting was never set to anything, not even null.

```
#include <config_exception.hpp>
```

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](#).

7.60.2 Constructor & Destructor Documentation

7.60.2.1 missing_exception()

```
hocon::missing_exception::missing_exception (  
    std::string const & path) [inline]
```

Definition at line 39 of file [config_exception.hpp](#).

7.60.3 Member Function Documentation

7.60.3.1 config_exception() [1/3]

```
hocon::config_exception::config_exception (  
    config_origin const & origin,  
    std::string const & message) [inline]
```

Definition at line 15 of file [config_exception.hpp](#).

7.60.3.2 config_exception() [2/3]

```
hocon::config_exception::config_exception (  
    std::string const & message) [inline]
```

Definition at line 17 of file [config_exception.hpp](#).

7.60.3.3 config_exception() [3/3]

```
hocon::config_exception::config_exception (  
    std::string const & message,  
    std::exception const & e) [inline]
```

Definition at line 19 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

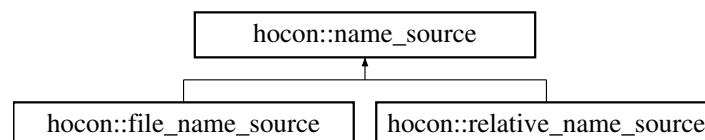
Definition at line 212 of file [config_value.hpp](#).

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](#).

7.62.2 Constructor & Destructor Documentation

7.62.2.1 name_source() [1/2]

```
hocon::name_source::name_source (
    shared_include_context context) [inline]
```

Definition at line 59 of file [simple_includer.hpp](#).

7.62.2.2 name_source() [2/2]

```
hocon::name_source::name_source () [inline]
```

Definition at line 66 of file [simple_includer.hpp](#).

7.62.3 Member Function Documentation

7.62.3.1 context_initialized()

```
bool hocon::name_source::context_initialized () const [inline]
```

Definition at line 81 of file [simple_includer.hpp](#).

7.62.3.2 get_context()

```
shared_include_context hocon::name_source::get_context () const [inline]
```

Definition at line 77 of file [simple_includer.hpp](#).

7.62.3.3 set_context()

```
void hocon::name_source::set_context (
    shared_include_context context) [inline]
```

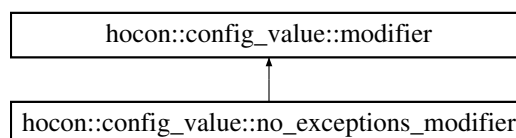
Definition at line 70 of file [simple_includer.hpp](#).

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](#).

7.63.2 Member Function Documentation

7.63.2.1 modify_child_may_throw()

```
shared_value hocon::config_value::no_exceptions_modifier::modify_child_may_throw (
    std::string const & key_or_null,
    shared_value v) [override], [virtual]
```

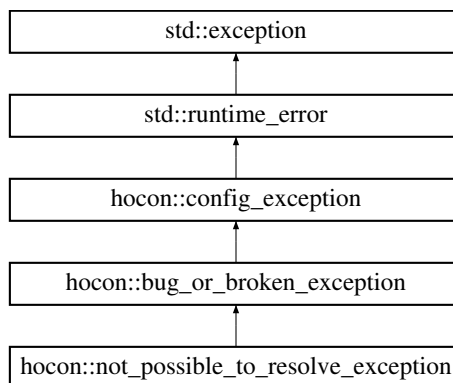
Implements [hocon::config_value::modifier](#).

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](#).

7.64.2 Constructor & Destructor Documentation

7.64.2.1 not_possible_to_resolve_exception()

```
hocon::not_possible_to_resolve_exception::not_possible_to_resolve_exception (
    std::string const & message) [inline]
```

Definition at line 128 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).

```
#include <config_exception.hpp>
```

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](#).

7.65.2 Constructor & Destructor Documentation

7.65.2.1 not_resolved_exception()

```
hocon::not_resolved_exception::not_resolved_exception (
    std::string const & message) [inline]
```

Definition at line 124 of file [config_exception.hpp](#).

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

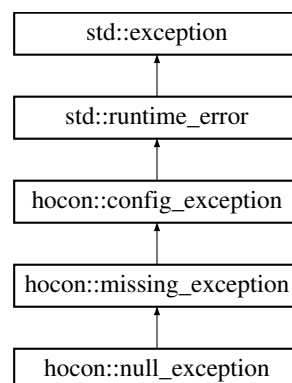
- [hocon/config_exception.hpp](#)

7.66 hocon::null_exception Struct Reference

Exception indicates that the setting was treated as missing because it was set to null.

```
#include <config_exception.hpp>
```

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](#).

7.66.2 Constructor & Destructor Documentation

7.66.2.1 null_exception()

```
hocon::null_exception::null_exception (
    config_origin const & origin,
    std::string const & path,
    std::string const & expected = "") [inline]
```

Definition at line 49 of file [config_exception.hpp](#).

7.66.3 Member Function Documentation

7.66.3.1 config_exception() [1/3]

```
hocon::config_exception::config_exception (
    config_origin const & origin,
    std::string const & message) [inline], [inherited]
```

Definition at line 15 of file [config_exception.hpp](#).

7.66.3.2 config_exception() [2/3]

```
hocon::config_exception::config_exception (
    std::string const & message) [inline], [inherited]
```

Definition at line 17 of file [config_exception.hpp](#).

7.66.3.3 config_exception() [3/3]

```
hocon::config_exception::config_exception (
    std::string const & message,
    std::exception const & e) [inline], [inherited]
```

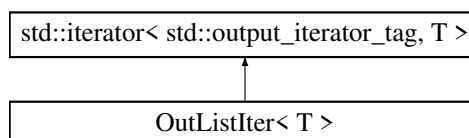
Definition at line 19 of file [config_exception.hpp](#).

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](#).

7.67.2 Constructor & Destructor Documentation

7.67.2.1 OutListIter()

```
template<class T >
OutListIter< T >::OutListIter () [inline]
```

Definition at line 145 of file [functional_list.hpp](#).

7.67.3 Member Function Documentation

7.67.3.1 getList()

```
template<class T >
List< T > OutListIter< T >::getList () const [inline]
```

Definition at line 152 of file [functional_list.hpp](#).

7.67.3.2 operator*()

```
template<class T >
T & OutListIter< T >::operator* () [inline]
```

Definition at line 146 of file [functional_list.hpp](#).

7.67.3.3 operator++()

```
template<class T >
OutListIter & OutListIter< T >::operator++ () [inline]
```

Definition at line 147 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](#).

7.69.2 Member Data Documentation

7.69.2.1 array_count

```
int hocon::config_parser::parse_context::array_count
```

Definition at line 42 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.

```
#include <config_exception.hpp>
```

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](#).

7.70.2 Constructor & Destructor Documentation

7.70.2.1 parse_exception()

```
hocon::parse_exception::parse_exception (
    config_origin const & origin,
    std::string const & message) [inline]
```

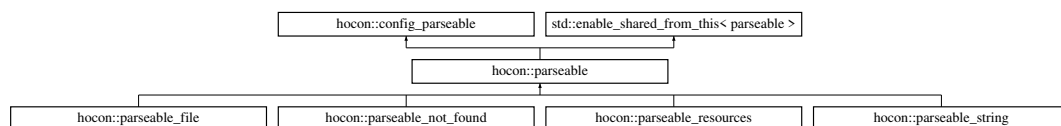
Definition at line 102 of file [config_exception.hpp](#).

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 paresable 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 config_parseable#options()
----------------	--

Returns

the parsed object

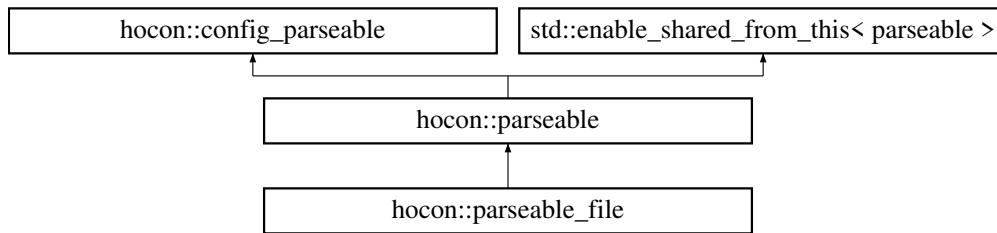
Implements [hocon::config_parseable](#).

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 create_origin()

```
shared_origin hocon::parseable_file::create_origin () const [override], [virtual]
```

Implements [hocon::parseable](#).

7.72.2.2 guess_syntax()

```
config_syntax hocon::parseable_file::guess_syntax () const [override], [virtual]
```

Reimplemented from [hocon::parseable](#).

7.72.2.3 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](#).

7.72.2.4 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](#).

7.72.2.5 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

<code>options</code>	parse options, should be based on the ones from <code>config_parseable#options()</code>
----------------------	---

Returns

the parsed object

Implements `hocon::config_parseable`.

7.72.2.6 reader()

```
std::unique_ptr< std::istream > hocon::parseable_file::reader () const [override], [virtual]
```

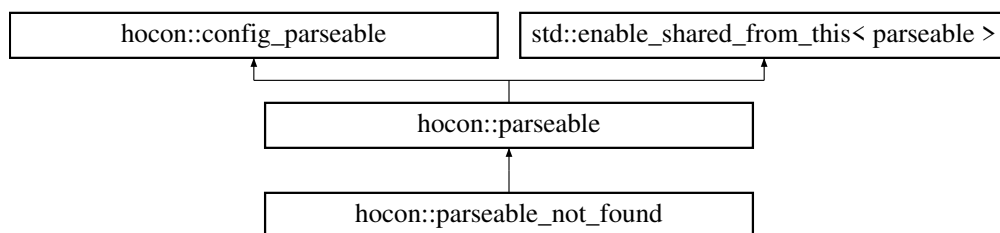
Implements `hocon::parseable`.

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 create_origin()

```
shared_origin hocon::parseable_not_found::create_origin () const [override], [virtual]
```

Implements [hocon::parseable](#).

7.73.2.2 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](#).

7.73.2.3 origin()

```
std::shared_ptr< const config\_origin > hocon::parseable::origin () const [override], [virtual], [inherited]
```

Returns a [config_origin](#) describing the origin of the paresable item.

Implements [hocon::config_parseable](#).

7.73.2.4 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 config_parseable#options()
----------------	--

Returns

the parsed object

Implements [hocon::config_parseable](#).

7.73.2.5 reader()

```
std::unique_ptr< std::istream > hocon::parseable_not_found::reader () const [override], [virtual]
```

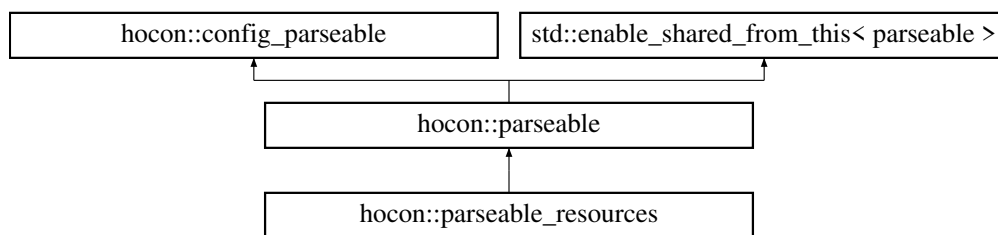
Implements [hocon::parseable](#).

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 create_origin()

```
shared_origin hocon::parseable_resources::create_origin () const [override], [virtual]
```

Implements [hocon::parseable](#).

7.74.2.2 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](#).

7.74.2.3 origin()

```
std::shared_ptr< const config\_origin > hocon::parseable::origin () const [override], [virtual], [inherited]
```

Returns a [config_origin](#) describing the origin of the paresable item.

Implements [hocon::config_parseable](#).

7.74.2.4 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 <code>config_parseable#options()</code>
----------------	---

Returns

the parsed object

Implements `hocon::config_parseable`.

7.74.2.5 reader()

```
std::unique_ptr< std::istream > hocon::parseable_resources::reader () const [override], [virtual]
```

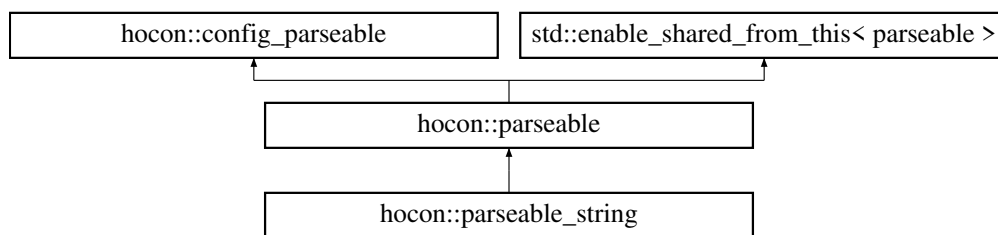
Implements `hocon::parseable`.

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 create_origin()

```
shared_origin hocon::parseable_string::create_origin () const [override], [virtual]
```

Implements [hocon::parseable](#).

7.75.2.2 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](#).

7.75.2.3 origin()

```
std::shared_ptr< const config\_origin > hocon::parseable::origin () const [override], [virtual], [inherited]
```

Returns a [config_origin](#) describing the origin of the paresable item.

Implements [hocon::config_parseable](#).

7.75.2.4 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 <code>config_parseable#options()</code>
----------------	---

Returns

the parsed object

Implements [hocon::config_parseable](#).

7.75.2.5 reader()

```
std::unique_ptr< std::istream > hocon::parseable_string::reader () const [override], [virtual]
```

Implements [hocon::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](#).

7.79.2 Member Function Documentation

7.79.2.1 operator==()

```
bool hocon::problem::operator== (
    const token & other) const    [override], [virtual]
```

Reimplemented from [hocon::token](#).

7.79.2.2 to_string()

```
std::string hocon::problem::to_string () const    [override], [virtual]
```

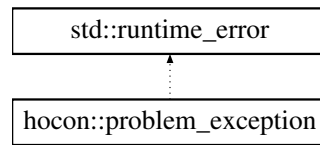
Reimplemented from [hocon::token](#).

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

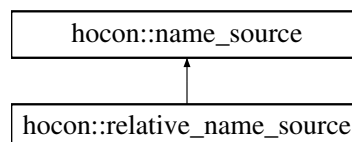
Definition at line 15 of file [tokenizer.hpp](#).

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](#).

7.81.2 Member Function Documentation

7.81.2.1 context_initialized()

```
bool hocon::name_source::context_initialized () const [inline], [inherited]
```

Definition at line 81 of file [simple_includer.hpp](#).

7.81.2.2 get_context()

```
shared_include_context hocon::name_source::get_context () const [inline], [inherited]
```

Definition at line 77 of file [simple_includer.hpp](#).

7.81.2.3 name_to_parseable()

```
shared_parseable hocon::relative_name_source::name_to_parseable (
    std::string name,
    config_parse_options parse_options) const [override], [virtual]
```

Implements [hocon::name_source](#).

7.81.2.4 set_context()

```
void hocon::name_source::set_context (
    shared_include_context context) [inline], [inherited]
```

Definition at line 70 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::config_concatenation](#), [hocon::config_delayed_merge](#), [hocon::config_delayed_merge_object](#), [hocon::simple_config_list](#), and [hocon::simple_config_object](#).

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::config_concatenation](#), [hocon::config_delayed_merge](#), [hocon::config_delayed_merge_object](#), [hocon::simple_config_list](#), and [hocon::simple_config_object](#).

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](#).

7.84.2 Constructor & Destructor Documentation

7.84.2.1 resolve_result()

```
template<typename V >
hocon::resolve_result< V >::resolve_result (
    resolve_context c,
    V v) [inline]
```

Definition at line 9 of file [resolve_result.hpp](#).

7.84.3 Member Data Documentation

7.84.3.1 context

```
template<typename V >
resolve_context hocon::resolve_result< V >::context
```

Definition at line 12 of file [resolve_result.hpp](#).

7.84.3.2 value

```
template<typename V >
V hocon::resolve_result< V >::value
```

Definition at line 13 of file [resolve_result.hpp](#).

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

- [hocon/config_value.hpp](#)
- [internal/resolve_context.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](#).

7.85.2 Member Typedef Documentation

7.85.2.1 node

```
std::list<std::shared_ptr<const container> > hocon::resolve_source::node
```

Definition at line 17 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](#).

7.86.2 Member Data Documentation

7.86.2.1 path_from_root

node hocon::resolve_source::result_with_path::path_from_root

Definition at line 21 of file [resolve_source.hpp](#).

7.86.2.2 result

[resolve_result](#)<shared_value> hocon::resolve_source::result_with_path::result

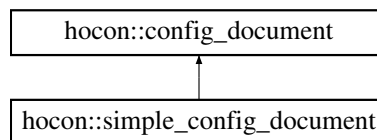
Definition at line 20 of file [resolve_source.hpp](#).

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 config_document .

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 config_node 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 newValue but the document was parsed with JSON, the first value in the concatenation will be parsed and inserted into the config_document .

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.

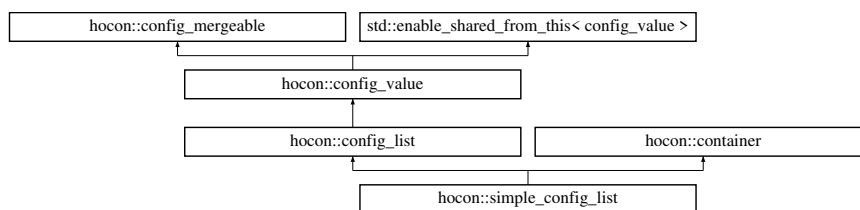
Implements [hocon::config_document](#).

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 **config_value** based on this one, but with the given origin.*
- 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 Typedef Documentation

7.88.2.1 iterator

```
using hocon::config_list::iterator = std::vector<shared_value>::const_iterator [inherited]
```

Definition at line 42 of file [config_list.hpp](#).

7.88.3 Member Enumeration Documentation

7.88.3.1 type

```
enum class 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](#).

7.88.4 Member Function Documentation

7.88.4.1 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.

7.88.4.2 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.

7.88.4.3 begin()

```
iterator hocon::simple_config_list::begin () const [inline], [override], [virtual]
```

Implements `hocon::config_list`.

Definition at line 45 of file `simple_config_list.hpp`.

7.88.4.4 contains()

```
bool hocon::simple_config_list::contains (  
    shared_value v) const [inline]
```

Definition at line 28 of file `simple_config_list.hpp`.

7.88.4.5 end()

```
iterator hocon::simple_config_list::end () const [inline], [override], [virtual]
```

Implements `hocon::config_list`.

Definition at line 46 of file `simple_config_list.hpp`.

7.88.4.6 equals()

```
template<typename T >
static bool hocon::config_value::equals (
    config_value const & other,
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.88.4.7 get()

```
shared_value hocon::simple_config_list::get (
    size_t index) const [inline], [override], [virtual]
```

Implements [hocon::config_list](#).

Definition at line 44 of file [simple_config_list.hpp](#).

7.88.4.8 get_resolve_status()

```
resolve_status hocon::simple_config_list::get_resolve_status () const [inline], [override],
[virtual]
```

Reimplemented from [hocon::config_value](#).

Definition at line 21 of file [simple_config_list.hpp](#).

7.88.4.9 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](#).

7.88.4.10 index_of()

```
int hocon::simple_config_list::index_of (
    shared_value v) [inline]
```

Definition at line 31 of file [simple_config_list.hpp](#).

7.88.4.11 is_empty()

```
bool hocon::simple_config_list::is_empty () const [inline], [override], [virtual]
```

Implements [hocon::config_list](#).

Definition at line 41 of file [simple_config_list.hpp](#).

7.88.4.12 new_copy()

```
shared_value hocon::simple_config_list::new_copy (
    shared_origin origin) const [override], [protected], [virtual]
```

Implements [hocon::config_value](#).

7.88.4.13 operator==(

```
bool hocon::simple_config_list::operator== (
    config_value const & other) const [override], [virtual]
```

Implements [hocon::config_value](#).

7.88.4.14 operator[]()

```
shared_value hocon::simple_config_list::operator[] (
    size_t index) const [inline], [override], [virtual]
```

Implements [hocon::config_list](#).

Definition at line 43 of file [simple_config_list.hpp](#).

7.88.4.15 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

7.88.4.16 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](#).

7.88.4.17 render() [1/3]

```
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

7.88.4.18 render() [2/3]

```
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

7.88.4.19 render() [3/3]

```
void hocon::simple_config_list::render (
    std::string & result,
    int indent,
    bool at_root,
    config_render_options options) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.88.4.20 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](#).

7.88.4.21 resolve_substitutions()

```
resolve_result< shared_value > hocon::simple_config_list::resolve_substitutions (
    resolve_context const & context,
    resolve_source const & source) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.88.4.22 size()

```
size_t hocon::simple_config_list::size () const [inline], [override], [virtual]
```

Implements [hocon::config_list](#).

Definition at line 42 of file [simple_config_list.hpp](#).

7.88.4.23 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](#).

7.88.4.24 type_name()

```
static char const * hocon::config_value::type_name (
    type t) [inline], [static], [inherited]
```

Definition at line 59 of file [config_value.hpp](#).

7.88.4.25 unwrapped()

```
unwrapped_value hocon::simple_config_list::unwrapped () const [override], [virtual]
```

Implements [hocon::config_list](#).

7.88.4.26 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](#).

7.88.4.27 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](#).

7.88.4.28 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](#).

7.88.4.29 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

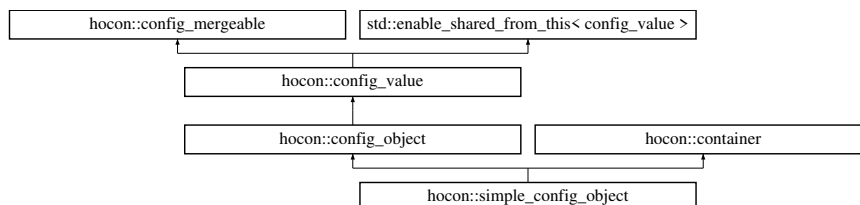
the new [config_value](#) with the given origin

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 config_value based on this one, but with the given origin.

- 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 Typedef Documentation

7.89.2.1 iterator

```
using hocon::config_object::iterator = std::unordered_map<std::string, shared_value>::const_iterator [inherited]
```

Definition at line 55 of file [config_object.hpp](#).

7.89.3 Member Enumeration Documentation

7.89.3.1 type

```
enum class 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](#).

7.89.4 Member Function Documentation

7.89.4.1 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.

7.89.4.2 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.

7.89.4.3 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

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

Implements [hocon::config_object](#).

7.89.4.4 begin()

```
iterator hocon::simple_config_object::begin () const [inline], [override], [virtual]
```

Implements [hocon::config_object](#).

Definition at line 24 of file [simple_config_object.hpp](#).

7.89.4.5 construct_delayed_merge()

```
shared_value hocon::config_object::construct_delayed_merge (
    shared_origin origin,
    std::vector< shared_value > stack) const [override], [protected], [virtual],
[inherited]
```

Reimplemented from [hocon::config_value](#).

7.89.4.6 end()

```
iterator hocon::simple_config_object::end () const [inline], [override], [virtual]
```

Implements [hocon::config_object](#).

Definition at line 25 of file [simple_config_object.hpp](#).

7.89.4.7 entry_set()

```
std::unordered_map< std::string, shared_value > const & hocon::simple_config_object::entry_set  
( ) const [override], [virtual]
```

Implements [hocon::config_object](#).

7.89.4.8 equals()

```
template<typename T >  
static bool hocon::config_value::equals (  
    config\_value const & other,  
    std::function< bool(T const &)> checker) [inline], [static], [protected], [inherited]
```

Definition at line 250 of file [config_value.hpp](#).

7.89.4.9 get()

```
shared_value hocon::simple_config_object::get (  
    std::string const & key) const [inline], [override], [virtual]
```

Implements [hocon::config_object](#).

Definition at line 28 of file [simple_config_object.hpp](#).

7.89.4.10 get_resolve_status()

```
resolve_status hocon::simple_config_object::get_resolve_status () const [inline], [override],  
[virtual]
```

Reimplemented from [hocon::config_value](#).

Definition at line 37 of file [simple_config_object.hpp](#).

7.89.4.11 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](#).

7.89.4.12 ignores_fallbacks()

```
bool hocon::simple_config_object::ignores_fallbacks () const [inline], [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

Definition at line 38 of file [simple_config_object.hpp](#).

7.89.4.13 is_empty()

```
bool hocon::simple_config_object::is_empty () const [inline], [override], [virtual]
```

Implements [hocon::config_object](#).

Definition at line 21 of file [simple_config_object.hpp](#).

7.89.4.14 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](#).

7.89.4.15 merged_with_object()

```
shared_value hocon::simple_config_object::merged_with_object (  
    shared_object fallback) const [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.89.4.16 new_copy()

```
shared_value hocon::simple_config_object::new_copy (  
    shared_origin ) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_object](#).

7.89.4.17 operator==(())

```
bool hocon::simple_config_object::operator== (  
    config\_value const & other) const [override], [virtual]
```

Implements [hocon::config_value](#).

7.89.4.18 operator[]()

```
shared_value hocon::simple_config_object::operator[] (
    std::string const & key) const [inline], [override], [virtual]
```

Implements [hocon::config_object](#).

Definition at line 23 of file [simple_config_object.hpp](#).

7.89.4.19 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

7.89.4.20 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](#).

7.89.4.21 render() [1/3]

```
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

7.89.4.22 render() [2/3]

```
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

7.89.4.23 render() [3/3]

```
void hocon::simple_config_object::render (
    std::string & s,
    int indent,
    bool at_root,
    config_render_options options) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.89.4.24 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](#).

7.89.4.25 resolve_substitutions()

```
resolve_result< shared_value > hocon::simple_config_object::resolve_substitutions (
    resolve_context const & context,
    resolve_source const & source) const [override], [protected], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.89.4.26 size()

```
size_t hocon::simple_config_object::size () const [inline], [override], [virtual]
```

Implements [hocon::config_object](#).

Definition at line 22 of file [simple_config_object.hpp](#).

7.89.4.27 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

7.89.4.28 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](#).

7.89.4.29 type_name()

```
static char const * hocon::config_value::type_name (
    type t) [inline], [static], [inherited]
```

Definition at line 59 of file [config_value.hpp](#).

7.89.4.30 unwrapped()

```
unwrapped_value hocon::simple_config_object::unwrapped () const [override], [virtual]
```

Implements [hocon::config_value](#).

7.89.4.31 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.

7.89.4.32 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](#).

7.89.4.33 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](#).

7.89.4.34 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](#).

7.89.4.35 with_fallbacks_ignored()

```
shared_value hocon::simple_config_object::with_fallbacks_ignored () const [override], [virtual]
```

Reimplemented from [hocon::config_value](#).

7.89.4.36 with_only_path()

```
shared_object hocon::simple_config_object::with_only_path (
    path raw_path) const [override], [virtual]
```

Implements [hocon::config_object](#).

7.89.4.37 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](#).

7.89.4.38 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

7.89.4.39 with_value() [1/2]

```
shared_object hocon::simple_config_object::with_value (
    path raw_path,
    shared_value value) const [override], [virtual]
```

Implements [hocon::config_object](#).

7.89.4.40 with_value() [2/2]

```
shared_object hocon::simple_config_object::with_value (
    std::string key,
    shared_value value) const [override], [virtual]
```

Implements [hocon::config_object](#).

7.89.4.41 without_path()

```
shared_object hocon::simple_config_object::without_path (
    path raw_path) const [override], [virtual]
```

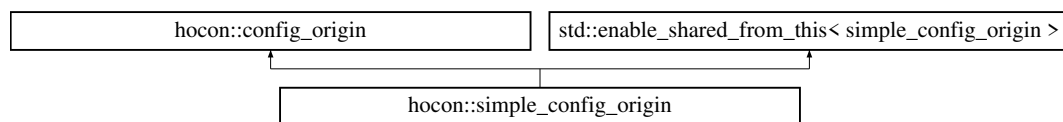
Implements [hocon::config_object](#).

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 config_origin based on this one, but with the given comments.
- 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_values` 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

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.

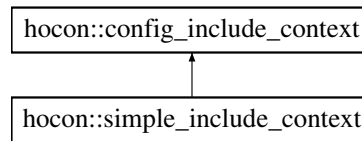
Implements [hocon::config_origin](#).

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 get_cur_dir()

```
std::string hocon::config_include_context::get_cur_dir () const [inline], [inherited]
```

Definition at line 55 of file [config_include_context.hpp](#).

7.91.2.2 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](#).

7.91.2.3 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](#).

7.91.2.4 set_cur_dir()

```
void hocon::config_include_context::set_cur_dir (
    std::string dir) const [inline], [inherited]
```

Definition at line 51 of file [config_include_context.hpp](#).

7.91.3 Member Data Documentation

7.91.3.1 _cur_dir

```
std::shared_ptr<std::string> hocon::config_include_context::_cur_dir [protected], [inherited]
```

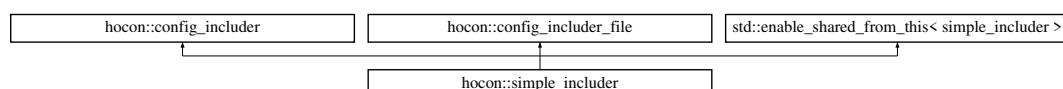
Definition at line 60 of file [config_include_context.hpp](#).

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

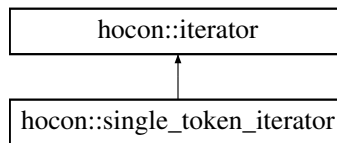
Implements [hocon::config_includer](#).

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](#).

7.93.2 Member Function Documentation

7.93.2.1 has_next()

```
bool hocon::single_token_iterator::has_next () [override], [virtual]
```

Implements [hocon::iterator](#).

7.93.2.2 next()

```
shared_token hocon::single_token_iterator::next () [override], [virtual]
```

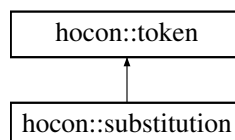
Implements [hocon::iterator](#).

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](#).

7.94.2 Member Function Documentation

7.94.2.1 operator==()

```
bool hocon::substitution::operator== (
    const token & other) const [override], [virtual]
```

Reimplemented from [hocon::token](#).

7.94.2.2 to_string()

```
std::string hocon::substitution::to_string () const [override], [virtual]
```

Reimplemented from [hocon::token](#).

7.94.2.3 token_text()

```
std::string hocon::substitution::token_text () const [override], [virtual]
```

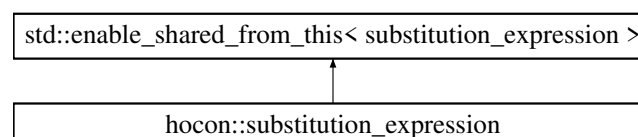
Reimplemented from [hocon::token](#).

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

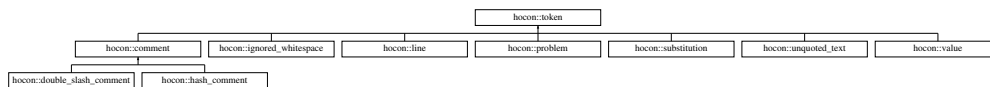
Definition at line 9 of file [substitution_expression.hpp](#).

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

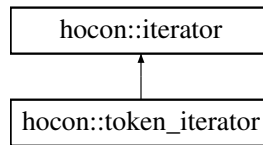
Definition at line 18 of file [token.hpp](#).

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](#).

7.97.2 Member Function Documentation

7.97.2.1 has_next()

```
bool hocon::token_iterator::has_next () [override], [virtual]
```

Implements [hocon::iterator](#).

7.97.2.2 next()

```
shared_token hocon::token_iterator::next () [override], [virtual]
```

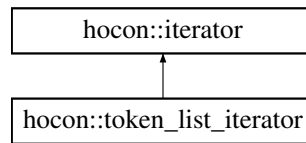
Implements [hocon::iterator](#).

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](#).

7.98.2 Member Function Documentation

7.98.2.1 has_next()

```
bool hocon::token_list_iterator::has_next () [override], [virtual]
```

Implements [hocon::iterator](#).

7.98.2.2 next()

```
shared_token hocon::token_list_iterator::next () [override], [virtual]
```

Implements [hocon::iterator](#).

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:

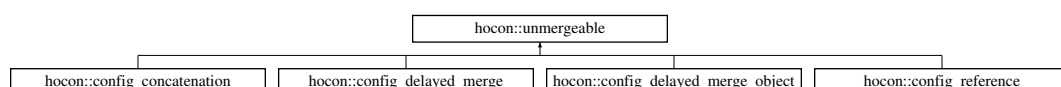
- internal/tokens.hpp

7.100 hocon::unmergeable Class Reference

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

```
#include <unmergeable.hpp>
```

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.

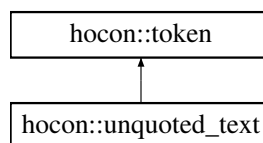
Definition at line 14 of file [unmergeable.hpp](#).

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](#).

7.101.2 Member Function Documentation

7.101.2.1 operator==()

```
bool hocon::unquoted_text::operator== (
    const token & other) const [override], [virtual]
```

Reimplemented from [hocon::token](#).

7.101.2.2 to_string()

```
std::string hocon::unquoted_text::to_string () const [override], [virtual]
```

Reimplemented from [hocon::token](#).

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

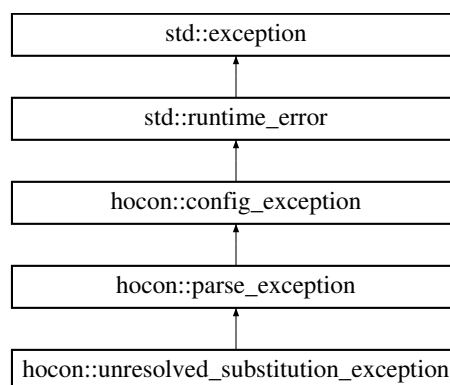
- [internal/tokens.hpp](#)

7.102 hocon::unresolved_substitution_exception Struct Reference

Exception indicating that a substitution did not resolve to anything.

```
#include <config_exception.hpp>
```

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](#).

7.102.2 Constructor & Destructor Documentation

7.102.2.1 unresolved_substitution_exception()

```
hocon::unresolved_substitution_exception::unresolved_substitution_exception (
    config_origin const & origin,
    std::string const & detail) [inline]
```

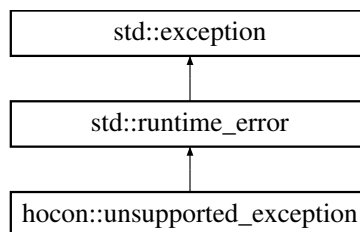
Definition at line 111 of file [config_exception.hpp](#).

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:

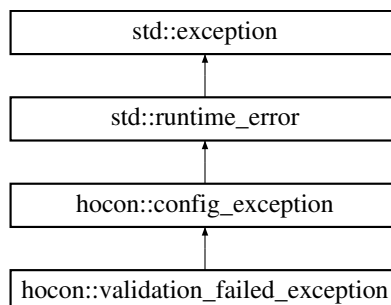
- [internal/token.hpp](#)

7.104 hocon::validation_failed_exception Struct Reference

Exception indicating that [config#check_valid](#) found validity problems.

```
#include <config_exception.hpp>
```

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](#).

7.104.2 Constructor & Destructor Documentation

7.104.2.1 validation_failed_exception()

```
hocon::validation_failed_exception::validation_failed_exception (
    std::vector< validation\_problem > problems_) [inline]
```

Definition at line [156](#) of file [config_exception.hpp](#).

7.104.3 Member Data Documentation

7.104.3.1 problems

```
const std::vector<validation\_problem> hocon::validation_failed_exception::problems
```

Definition at line [159](#) 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](#).

```
#include <config_exception.hpp>
```


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](#).

7.105.2 Constructor & Destructor Documentation

7.105.2.1 validation_problem()

```
hocon::validation_problem::validation_problem (  
    std::string path_,  
    shared_origin origin_,  
    std::string problem_) [inline]
```

Definition at line [137](#) of file [config_exception.hpp](#).

7.105.3 Member Function Documentation

7.105.3.1 to_string()

```
std::string hocon::validation_problem::to_string () [inline]
```

Definition at line [144](#) of file [config_exception.hpp](#).

7.105.4 Member Data Documentation

7.105.4.1 origin

```
const shared_origin hocon::validation_problem::origin
```

Definition at line [141](#) of file [config_exception.hpp](#).

7.105.4.2 path

```
const std::string hocon::validation_problem::path
```

Definition at line 140 of file [config_exception.hpp](#).

7.105.4.3 problem

```
const std::string hocon::validation_problem::problem
```

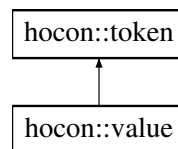
Definition at line 142 of file [config_exception.hpp](#).

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](#).

7.106.2 Member Function Documentation

7.106.2.1 operator==()

```
bool hocon::value::operator== (
    const token & other) const [override], [virtual]
```

Reimplemented from [hocon::token](#).

7.106.2.2 origin()

```
shared_origin const & hocon::value::origin () const [override], [virtual]
```

Reimplemented from [hocon::token](#).

7.106.2.3 to_string()

```
std::string hocon::value::to_string () const [override], [virtual]
```

Reimplemented from [hocon::token](#).

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

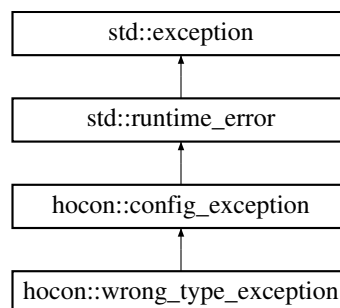
- internal/tokens.hpp

7.107 hocon::wrong_type_exception Struct Reference

Exception indicating that the type of a value does not match the type you requested.

```
#include <config_exception.hpp>
```

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](#).

7.107.2 Constructor & Destructor Documentation

7.107.2.1 wrong_type_exception()

```
hocon::wrong_type_exception::wrong_type_exception (
    config_origin const & origin,
    std::string const & path,
    std::string const & expected,
    std::string const & actual) [inline]
```

Definition at line 28 of file [config_exception.hpp](#).

7.107.3 Member Function Documentation

7.107.3.1 config_exception() [1/3]

```
hocon::config_exception::config_exception (
    config_origin const & origin,
    std::string const & message) [inline]
```

Definition at line 15 of file [config_exception.hpp](#).

7.107.3.2 config_exception() [2/3]

```
hocon::config_exception::config_exception (
    std::string const & message) [inline]
```

Definition at line 17 of file [config_exception.hpp](#).

7.107.3.3 config_exception() [3/3]

```
hocon::config_exception::config_exception (
    std::string const & message,
    std::exception const & e) [inline]
```

Definition at line 19 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 config.hpp

```
00001 #pragma once
00002
00003 #include "types.hpp"
00004 #include "config_mergeable.hpp"
00005 #include "config_origin.hpp"
00006 #include "config_object.hpp"
00007 #include "config_resolve_options.hpp"
00008 #include "config_value.hpp"
00009 #include "config_list.hpp"
00010 #include "config_exception.hpp"
00011 #include "path.hpp"
00012 #include <vector>
00013 #include <string>
00014 #include <set>
00015 #include "export.h"
00016 #include <leatherman/locale/locale.hpp>
00017
00018 namespace hocon {
00019
00020     enum class time_unit { NANoseconds, MICROseconds, MILLIseconds, SECONDS, MINUTES, HOURS, DAYS };
00021
00022     class LIBCPP_HOCON_EXPORT config : public config_mergeable, public
00023     std::enable_shared_from_this<config> {
00024     friend class config_object;
00025     friend class config_value;
00026     friend class config_parseable;
00027     friend class parseable;
00028     public:
00029     static shared_config parse_file_any_syntax(std::string file_basename, config_parse_options
00030 options);
00031
00032     static shared_config parse_file_any_syntax(std::string file_basename);
00033
00034     static shared_config parse_string(std::string s, config_parse_options options);
00035
00036     static shared_config parse_string(std::string s);
00037
00038     virtual shared_object root() const;
00039
00040     virtual shared_origin origin() const;
00041
00042     std::shared_ptr<const config_mergeable> with_fallback(std::shared_ptr<const config_mergeable>
00043 other) const override;
00044
00045     shared_value to_fallback_value() const override;
00046
00047     virtual shared_config resolve() const;
00048
00049     virtual shared_config resolve(config_resolve_options options) const;
00050
00051     virtual bool is_resolved() const;
00052
00053     virtual shared_config resolve_with(shared_config source) const;
00054
00055     virtual shared_config resolve_with(shared_config source, config_resolve_options options)
00056 const;
00057
00058 }
```

```

00448     virtual void check_valid(shared_config reference, std::vector<std::string> restrict_to_paths)
00449     const;
00471     virtual bool has_path(std::string const& path) const;
00472
00510     virtual bool has_path_or_null(std::string const& path) const;
00511
00518     virtual bool is_empty() const;
00519
00542     virtual std::set<std::pair<std::string, std::shared_ptr<const config_value>> entry_set()
00543     const;
00543
00565     virtual bool get_is_null(std::string const& path) const;
00566
00567     virtual bool get_bool(std::string const& path) const;
00568     virtual int get_int(std::string const& path) const;
00569     virtual int64_t get_long(std::string const& path) const;
00570     virtual double get_double(std::string const& path) const;
00571     virtual std::string get_string(std::string const& path) const;
00572     virtual std::shared_ptr<const config_object> get_object(std::string const& path) const;
00573     virtual shared_config get_config(std::string const& path) const;
00574     virtual unwrapped_value get_any_ref(std::string const& path) const;
00575     virtual std::shared_ptr<const config_value> get_value(std::string const& path) const;
00576
00577     template<typename T>
00578     std::vector<T> get_homogeneous_unwrapped_list(std::string const& path) const {
00579         auto list = boost::get<std::vector<unwrapped_value>(get_list(path)->unwrapped());
00580         std::vector<T> T_list;
00581         for (auto item : list) {
00582             try {
00583                 T_list.push_back(boost::get<T>(item));
00584             } catch (boost::bad_get &ex) {
00585                 throw config_exception(leatherman::locale::format("The list did not contain only
the desired type."));
00586             }
00587         }
00588         return T_list;
00589     }
00590
00591     virtual shared_list get_list(std::string const& path) const;
00592     virtual std::vector<bool> get_bool_list(std::string const& path) const;
00593     virtual std::vector<int> get_int_list(std::string const& path) const;
00594     virtual std::vector<int64_t> get_long_list(std::string const& path) const;
00595     virtual std::vector<double> get_double_list(std::string const& path) const;
00596     virtual std::vector<std::string> get_string_list(std::string const& path) const;
00597     virtual std::vector<shared_object> get_object_list(std::string const& path) const;
00598     virtual std::vector<shared_config> get_config_list(std::string const& path) const;
00599
00600     // TODO: memory parsing
00601
00610     virtual int64_t get_duration(std::string const& path, time_unit unit) const;
00611
00623     virtual shared_config with_only_path(std::string const& path) const;
00624
00635     virtual shared_config without_path(std::string const& path) const;
00636
00648     virtual shared_config at_path(std::string const& path) const;
00649
00660     virtual shared_config at_key(std::string const& key) const;
00661
00677     virtual shared_config with_value(std::string const& path, std::shared_ptr<const config_value>
value) const;
00678
00679     bool operator==(config const& other) const;
00680
00681     config(shared_object object);
00682
00683     static shared_object env_variables_as_config_object();
00684
00685     protected:
00686     shared_value find(std::string const& path_expression, config_value::type expected) const;
00687     shared_value find(path path_expression, config_value::type expected, path original_path)
const;
00688     shared_value find(path path_expression, config_value::type expected) const;
00689     shared_config at_key(shared_origin origin, std::string const& key) const;
00690
00691     static shared_includer default_includer();
00692
00693     // TODO: memory and duration parsing
00694
00695     private:
00706     static duration parse_duration(std::string input, shared_origin origin_for_exception,
std::string path_for_exception);
00707
00708     static duration convert(int64_t number, time_unit units);
00709     static duration convert(double number, time_unit units);
00710     static time_unit get_units(std::string const& unit_string);

```

```

00711         duration get_duration(std::string const& path) const;
00712
00713         shared_value has_path_peek(std::string const& path_expression) const;
00714         shared_value peek_path(path desired_path) const;
00715
00716         static void find_paths(std::set<std::pair<std::string, std::shared_ptr<const config_value>>&
entries,
00717                               path parent, shared_object obj);
00718         static shared_value throw_if_null(shared_value v, config_value::type expected, path
original_path);
00719         static shared_value find_key(shared_object self, std::string const& key,
00720                                     config_value::type expected, path original_path);
00721         static shared_value find_key_or_null(shared_object self, std::string const& key,
00722                                             config_value::type expected, path original_path);
00723         static shared_value find_or_null(shared_object self, path desired_path,
00724                                         config_value::type expected, path original_path);
00725         shared_value find_or_null(std::string const& path_expression, config_value::type expected)
const;
00726         shared_value find_or_null(path path_expression, config_value::type expected, path
original_path) const;
00727
00728         shared_object _object;
00729     };
00730
00731     template<>
00732     std::vector<int64_t> config::get_homogeneous_unwrapped_list(std::string const& path) const;
00733
00734 } // namespace hocon

```

8.2 config_exception.hpp

```

00001 #pragma once
00002
00003 #include <stdexcept>
00004 #include <string>
00005 #include "config_origin.hpp"
00006 #include <leatherman/locale/locale.hpp>
00007
00008 namespace hocon {
00009
00014     struct config_exception : public std::runtime_error {
00015         config_exception(config_origin const& origin, std::string const& message) :
00016             runtime_error(leatherman::locale::format("{1}: {2}", origin.description(), message)) { }
00017
00018         config_exception(std::string const& message) : runtime_error(message) { }
00019
00019         config_exception(std::string const& message, std::exception const& e) :
runtime_error(leatherman::locale::format("{1} {2}", message, e.what())) { }
00020     };
00021
00027     struct wrong_type_exception : public config_exception {
00028         wrong_type_exception(config_origin const& origin,
00029                             std::string const& path, std::string const& expected, std::string const&
actual) :
00030             config_exception(origin, leatherman::locale::format("{1} has type {2} rather than
{3}", path, actual, expected)) { }
00031         using config_exception::config_exception;
00032     };
00033
00038     struct missing_exception : public config_exception {
00039         missing_exception(std::string const& path) :
00040             config_exception(leatherman::locale::format("No configuration setting found for key
'{1}'", path)) { }
00041         using config_exception::config_exception;
00042     };
00043
00048     struct null_exception : public missing_exception {
00049         null_exception(config_origin const& origin, std::string const& path, std::string const&
expected = "") :
00050             missing_exception(origin, (expected.empty() ?
leatherman::locale::format("Configuration key \"{1}\" is null", path)
00051                             : leatherman::locale::format("Configuration key \"{1}\" is
set to null but expected {2}", path, expected))) { }
00052     };
00053
00059     struct bad_value_exception : public config_exception {
00060         bad_value_exception(config_origin const& origin, std::string const& path, std::string const&
message) :
00061             config_exception(origin, leatherman::locale::format("Invalid value at '{1}': {2}",
path, message)) { }
00062         bad_value_exception(std::string const& path, std::string const& message) :
00063             config_exception(leatherman::locale::format("Invalid value at '{1}': {2}", path,
message)) { }

```

```

00064     };
00065
00071     struct bad_path_exception : public config_exception {
00072         bad_path_exception(config_origin const& origin, std::string const& path, std::string const&
message) :
00073             config_exception(origin, path.empty() ? message : leatherman::locale::format("Invalid
path '{1}': {2}", path, message)) { }
00074         bad_path_exception(std::string const& path, std::string const& message) :
00075             config_exception(path.empty() ? message : leatherman::locale::format("Invalid path
'{1}': {2}", path, message)) { }
00076     };
00077
00085     struct bug_or_broken_exception : public config_exception {
00086         bug_or_broken_exception(std::string const& message) : config_exception(message) { }
00087     };
00088
00093     struct io_exception : public config_exception {
00094         io_exception(config_origin const& origin, std::string const& message) :
config_exception(origin, message) { }
00095     };
00096
00101     struct parse_exception : public config_exception {
00102         parse_exception(config_origin const& origin, std::string const& message) :
config_exception(origin, message) { }
00103     };
00104
00105
00110     struct unresolved_substitution_exception : public parse_exception {
00111         unresolved_substitution_exception(config_origin const& origin, std::string const& detail) :
00112             parse_exception(origin, leatherman::locale::format("Could not resolve substitution to a
value: {1}", detail)) { }
00113     };
00114
00123     struct not_resolved_exception : public bug_or_broken_exception {
00124         not_resolved_exception(std::string const& message) : bug_or_broken_exception(message) { }
00125     };
00126
00127     struct not_possible_to_resolve_exception : public bug_or_broken_exception {
00128         not_possible_to_resolve_exception(std::string const& message) :
bug_or_broken_exception(message) { }
00129     };
00130
00136     struct validation_problem {
00137         validation_problem(std::string path_, shared_origin origin_, std::string problem_) :
00138             path(std::move(path_)), origin(std::move(origin_)), problem(std::move(problem_)) { }
00139
00140         const std::string path;
00141         const shared_origin origin;
00142         const std::string problem;
00143
00144         std::string to_string() {
00145             return leatherman::locale::format("ValidationProblem({1},{2},{3})", path,
origin->description(), problem);
00146         }
00147     };
00148
00155     struct validation_failed_exception : public config_exception {
00156         validation_failed_exception(std::vector<validation_problem> problems_) :
00157             config_exception(make_message(problems_), problems_.std::move(problems_)) { }
00158
00159         const std::vector<validation_problem> problems;
00160
00161     private:
00162         static std::string make_message(std::vector<validation_problem> const& problems_) {
00163             std::string msg;
00164             for (auto &p : problems_) {
00165                 auto sep = std::string(": ");
00166                 for (auto &s : {p.origin->description(), sep, p.path, sep, p.problem, std::string(",
")}) {
00167                     msg.append(s);
00168                 }
00169             }
00170             if (msg.empty()) {
00171                 throw bug_or_broken_exception(leatherman::locale::format("validation_failed_exception
must have a non-empty list of problems"));
00172             }
00173             msg.resize(msg.length() - 2);
00174             return msg;
00175         }
00176     };
00177
00181     struct generic_exception : public config_exception {
00182         generic_exception(std::string const& message) : config_exception(message) { }
00183     };
00184 } // namespace hocon

```


8.3 config_include_context.hpp

```

00001 #pragma once
00002
00003 #include "types.hpp"
00004 #include "config_parse_options.hpp"
00005 #include "export.h"
00006
00007 namespace hocon {
00008
00021     class LIBCPP_HOCON_EXPORT config_include_context {
00022     public:
00023         config_include_context() : _cur_dir(new std::string("")) {}
00040         virtual shared_parseable relative_to(std::string file_name) const = 0;
00041
00049         virtual config_parse_options parse_options() const = 0;
00050
00051         void set_cur_dir(std::string dir) const {
00052             _cur_dir->assign(dir);
00053         }
00054
00055         std::string get_cur_dir() const {
00056             return *_cur_dir;
00057         }
00058
00059     protected:
00060         std::shared_ptr<std::string> _cur_dir;
00061     };
00062
00063 } // namespace hocon

```

8.4 config_includer.hpp

```

00001 #pragma once
00002
00003 #include "types.hpp"
00004 #include <string>
00005 #include <memory>
00006 #include "export.h"
00007
00008 namespace hocon {
00015     class LIBCPP_HOCON_EXPORT config_includer {
00016     public:
00032         virtual shared_includer with_fallback(shared_includer fallback) const = 0;
00033
00051         virtual shared_object include(shared_include_context context, std::string what) const = 0;
00052     };
00053 } // namespace hocon

```

8.5 config_includer_file.hpp

```

00001 #pragma once
00002
00003 #include "export.h"
00004 #include "types.hpp"
00005
00006 namespace hocon {
00014     class LIBCPP_HOCON_EXPORT config_includer_file {
00015     public:
00027         virtual shared_object include_file(shared_include_context context, std::string what) const =
0;
00028     };
00029
00030 } // namespace hocon

```

8.6 config_list.hpp

```

00001 #pragma once
00002
00003 #include "types.hpp"
00004 #include "export.h"
00005 #include <vector>
00006
00007 namespace hocon {

```

```

00008
00037     class LIBCPP_HOCON_EXPORT config_list : public config_value {
00038     public:
00039         config_list(shared_origin origin) : config_value(move(origin)) {}
00040
00041         // list interface
00042         using iterator = std::vector<shared_value>::const_iterator;
00043         virtual bool is_empty() const = 0;
00044         virtual size_t size() const = 0;
00045         virtual shared_value operator[](size_t index) const = 0;
00046         virtual shared_value get(size_t index) const = 0;
00047         virtual iterator begin() const = 0;
00048         virtual iterator end() const = 0;
00049         virtual unwrapped_value unwrapped() const = 0;
00050     };
00051 } // namespace hocon

```

8.7 config_mergeable.hpp

```

00001 #pragma once
00002
00003 #include <memory>
00004 #include "export.h"
00005
00006 namespace hocon {
00007
00008     class LIBCPP_HOCON_EXPORT config_mergeable {
00009     friend class config_value;
00010     public:
00062         virtual std::shared_ptr<const config_mergeable> with_fallback(std::shared_ptr<const
config_mergeable> other) const = 0;
00063
00064     protected:
00069         virtual shared_value to_fallback_value() const = 0;
00070     };
00071
00072 } // namespace hocon

```

8.8 config_object.hpp

```

00001 #pragma once
00002
00003 #include "config_value.hpp"
00004 #include "config_mergeable.hpp"
00005 #include "path.hpp"
00006 #include <unordered_map>
00007 #include "export.h"
00008
00009 namespace hocon {
00010
00011     class LIBCPP_HOCON_EXPORT config_object : public config_value {
00012     friend class config;
00013     friend class config_value;
00014     friend class simple_config_object;
00015     friend class resolve_source;
00016     friend class config_delayed_merge_object;
00017     public:
00025         virtual std::shared_ptr<const config> to_config() const;
00026
00027         config_object(shared_origin origin);
00028
00029         config_value::type value_type() const override;
00030
00031         virtual shared_object with_value(path raw_path, shared_value value) const = 0;
00032         virtual shared_object with_value(std::string key, shared_value value) const = 0;
00033
00046         virtual shared_value attempt_peek_with_partial_resolve(std::string const& key) const = 0;
00047
00052         virtual std::vector<std::string> key_set() const = 0;
00053
00054         // map interface
00055         using iterator = std::unordered_map<std::string, shared_value>::const_iterator;
00056         virtual bool is_empty() const = 0;
00057         virtual size_t size() const = 0;
00058         virtual shared_value operator[](std::string const& key) const = 0;
00059         virtual shared_value get(std::string const& key) const = 0;
00060         virtual iterator begin() const = 0;
00061         virtual iterator end() const = 0;
00062

```

```

00063     protected:
00064         shared_value peek_path(path desired_path) const;
00065         shared_value peek_assuming_resolved(std::string const& key, path original_path) const;
00066
00067         virtual shared_object new_copy(resolve_status const& status, shared_origin origin) const = 0;
00068         shared_value new_copy(shared_origin origin) const override;
00069
00070         shared_value construct_delayed_merge(shared_origin origin, std::vector<shared_value> stack)
00071         const override;
00072
00073         virtual std::unordered_map<std::string, shared_value> const& entry_set() const = 0;
00074         virtual shared_object without_path(path raw_path) const = 0;
00075         virtual shared_object with_only_path(path raw_path) const = 0;
00076         virtual shared_object with_only_path_or_null(path raw_path) const = 0;
00077
00078         static shared_value peek_path(const config_object* self, path desired_path);
00079         static shared_origin merge_origins(std::vector<shared_value> const& stack);
00080     };
00081 } // namespace hocon

```

8.9 config_origin.hpp

```

00001 #pragma once
00002
00003 #include "types.hpp"
00004 #include <memory>
00005 #include <string>
00006 #include <vector>
00007 #include "export.h"
00008
00009 namespace hocon {
00010
00031     class config_origin {
00032     public:
00039         LIBCPP_HOCON_EXPORT virtual std::string const& description() const = 0;
00040
00058         LIBCPP_HOCON_EXPORT virtual shared_origin with_line_number(int line_number) const = 0;
00059
00066         LIBCPP_HOCON_EXPORT virtual int line_number() const = 0;
00067
00078         LIBCPP_HOCON_EXPORT virtual std::vector<std::string> const& comments() const = 0;
00079
00095         LIBCPP_HOCON_EXPORT virtual shared_origin with_comments(std::vector<std::string> comments)
00096         const = 0;
00097     };
00098 } // namespace hocon

```

8.10 config_parse_options.hpp

```

00001 #pragma once
00002
00003 #include "types.hpp"
00004 #include "config_syntax.hpp"
00005 #include "export.h"
00006
00007 namespace hocon {
00025     class LIBCPP_HOCON_EXPORT config_parse_options {
00026     public:
00033         config_parse_options();
00034
00041         static config_parse_options defaults();
00042
00051         config_parse_options set_syntax(config_syntax syntax) const;
00052
00056         config_syntax const& get_syntax() const;
00057
00068         config_parse_options set_origin_description(shared_string origin_description) const;
00069
00074         shared_string const& get_origin_description() const;
00075
00084         config_parse_options set_allow_missing(bool allow_missing) const;
00085
00090         bool get_allow_missing() const;
00091
00099         config_parse_options set_includer(shared_includer includer) const;
00100
00110         config_parse_options prepend_includer(shared_includer includer) const;

```

```

00111
00120         config_parse_options append_includer(shared_includer includer) const;
00121
00126         shared_includer const& get_includer() const;
00127
00128     private:
00129         config_parse_options(shared_string origin_desc,
00130                             bool allow_missing, shared_includer includer,
00131                             config_syntax syntax = config_syntax::UNSPECIFIED);
00132         config_parse_options with_fallback_origin_description(shared_string origin_description) const;
00133
00134         config_syntax _syntax;
00135         shared_string _origin_description;
00136         bool _allow_missing;
00137         shared_includer _includer;
00138     };
00139 } // namespace hocon

```

8.11 config_parseable.hpp

```

00001 #pragma once
00002
00003 #include "types.hpp"
00004 #include "export.h"
00005
00006 namespace hocon {
00007
00019     class LIBCPP_HOCON_EXPORT config_parseable {
00020     public:
00031         virtual shared_object parse(config_parse_options const& options) const = 0;
00032
00036         virtual shared_origin origin() const = 0;
00037
00044         virtual config_parse_options const& options() const = 0;
00045     };
00046
00047 } // namespace hocon

```

8.12 config_render_options.hpp

```

00001 #pragma once
00002
00003 #include "export.h"
00004
00005 namespace hocon {
00006
00020     class LIBCPP_HOCON_EXPORT config_render_options {
00021     public:
00026         config_render_options(bool origin_comments = true, bool comments = true,
00027                             bool formatted = true, bool json = true);
00028
00035         static config_render_options concise();
00036
00046         config_render_options set_comments(bool value);
00047
00054         bool get_comments() const;
00055
00072         config_render_options set_origin_comments(bool value);
00073
00080         bool get_origin_comments() const;
00081
00090         config_render_options set_formatted(bool value);
00091
00098         bool get_formatted() const;
00099
00111         config_render_options set_json(bool value);
00112
00119         bool get_json() const;
00120
00121     private:
00122         bool _origin_comments;
00123         bool _comments;
00124         bool _formatted;
00125         bool _json;
00126     };
00127
00128 } // namespace hocon

```

8.13 config_resolve_options.hpp

```

00001 #pragma once
00002
00003 #include "export.h"
00004
00005 namespace hocon {
00006
00030     class LIBCPP_HOCON_EXPORT config_resolve_options {
00031     public:
00038         config_resolve_options(bool use_system_environment = true, bool allow_unresolved = false);
00039
00046         config_resolve_options set_use_system_environment(bool value) const;
00047
00055         bool get_use_system_environment() const;
00056
00068         config_resolve_options set_allow_unresolved(bool value) const;
00069
00076         bool get_allow_unresolved() const;
00077
00078     private:
00079         bool _use_system_environment;
00080         bool _allow_unresolved;
00081     };
00082
00083 } // namespace hocon

```

8.14 config_syntax.hpp

```

00001 #pragma once
00002
00003 enum class config_syntax {
00010     JSON,
00011
00018     CONF,
00019
00021     UNSPECIFIED
00022
00023     // Original project also supported Java's .properties formats, but we do not
00024 };

```

8.15 config_value.hpp

```

00001
00002 #pragma once
00003
00004 #include "config_origin.hpp"
00005 #include "config_render_options.hpp"
00006 #include "config_mergeable.hpp"
00007 #include "path.hpp"
00008 #include <string>
00009 #include <vector>
00010 #include "export.h"
00011 #include <leatherman/locale/locale.hpp>
00012
00013 namespace hocon {
00014
00015     class unmergeable;
00016     class resolve_source;
00017     class resolve_context;
00018
00019     template<typename T>
00020     struct resolve_result;
00021
00022     enum class resolve_status { RESOLVED, UNRESOLVED };
00023
00039     class LIBCPP_HOCON_EXPORT config_value : public config_mergeable, public
std::enable_shared_from_this<config_value> {
00040     friend class token;
00041     friend class value;
00042     friend class default_transformer;
00043     friend class config;
00044     friend class config_object;
00045     friend class simple_config_object;
00046     friend class simple_config_list;
00047     friend class config_concatenation;
00048     friend class resolve_context;
00049     friend class config_delayed_merge;
00050     friend class config_delayed_merge_object;

```

```

00051     public:
00052     enum class type {
00053         OBJECT, LIST, NUMBER, BOOLEAN, CONFIG_NULL, STRING, UNSPECIFIED
00054     };
00055     static char const* type_name(type t) {
00056         switch (t) {
00057             case type::OBJECT: return "object";
00058             case type::LIST: return "list";
00059             case type::NUMBER: return "number";
00060             case type::BOOLEAN: return "boolean";
00061             case type::CONFIG_NULL: return "null";
00062             case type::STRING: return "string";
00063             case type::UNSPECIFIED: return "unspecified";
00064             default: throw std::logic_error(leatherman::locale::format("Got impossible value for
type enum"));
00065         }
00066     }
00067     virtual shared_origin const& origin() const;
00068     virtual type value_type() const = 0;
00069     char const* value_type_name() const {
00070         return type_name(value_type());
00071     }
00072     virtual unwrapped_value unwrapped() const = 0;
00073     virtual std::string render() const;
00074     virtual std::string render(config_render_options options) const;
00075     shared_config at_key(std::string const& key) const;
00076     shared_config at_path(std::string const& path_expression) const;
00077     virtual shared_value with_origin(shared_origin origin) const;
00078     virtual shared_value relativized(std::string prefix) const { return shared_from_this(); }
00079     virtual resolve_status get_resolve_status() const;
00080     friend resolve_status resolve_status_from_values(std::vector<shared_value> const& v);
00081     std::shared_ptr<const config_mergeable> with_fallback(std::shared_ptr<const config_mergeable>
other) const override;
00082     virtual bool operator==(config_value const& other) const = 0;
00083     virtual std::string transform_to_string() const;
00084     protected:
00085     config_value(shared_origin origin);
00086     virtual void render(std::string& result, int indent, bool at_root, std::string const& at_key,
config_render_options options) const;
00087     virtual void render(std::string& result, int indent, bool at_root, config_render_options
options) const;
00088     static void indent(std::string& result, int indent, config_render_options const& options);
00089     shared_config at_key(shared_origin origin, std::string const& key) const;
00090     shared_config at_path(shared_origin origin, path raw_path) const;
00091     virtual shared_value new_copy(shared_origin origin) const = 0;
00092     virtual resolve_result<shared_value>
    resolve_substitutions(resolve_context const& context, resolve_source const& source) const;
00093     static std::vector<shared_value> replace_child_in_list(std::vector<shared_value> const&
values,
00094     shared_value const& child, shared_value
replacement);
00095     static bool has_descendant_in_list(std::vector<shared_value> const& values, shared_value
const& descendant);
00096     class modifier {
00097     public:
00098         virtual shared_value modify_child_may_throw(std::string const& key_or_null, shared_value
v) = 0;
00099     };
00100     class no_exceptions_modifier : public modifier {
00101     public:
00102         no_exceptions_modifier(std::string prefix);
00103         shared_value modify_child_may_throw(std::string const &key_or_null, shared_value v)
override;

```

```

00222         shared_value modify_child(std::string const& key, shared_value v) const;
00223     private:
00224         std::string _prefix;
00225     };
00226     void require_not_ignoring_fallbacks() const;
00227
00228     /* this is virtualized rather than a field because only some subclasses
00229      * really need to store the boolean, and they may be able to pack it
00230      * with another boolean to save space.
00231      */
00232     virtual bool ignores_fallbacks() const;
00233     virtual shared_value with_fallbacks_ignored() const;
00234
00235     shared_value merged_with_the_unmergeable(std::vector<shared_value> stack,
00236                                              std::shared_ptr<const unmergeable> fallback) const;
00237     shared_value merged_with_the_unmergeable(std::shared_ptr<const unmergeable> fallback) const;
00238
00239     shared_value merged_with_object(std::vector<shared_value> stack, shared_object fallback)
00240     const;
00241     virtual shared_value merged_with_object(shared_object fallback) const;
00242
00243     shared_value merged_with_non_object(std::vector<shared_value> stack, shared_value fallback)
00244     const;
00245     virtual shared_value merged_with_non_object(shared_value fallback) const;
00246
00247     virtual shared_value construct_delayed_merge(shared_origin origin, std::vector<shared_value>
00248     stack) const;
00249
00250     shared_value to_fallback_value() const override;
00251
00252     template<typename T>
00253     static bool equals(config_value const& other, std::function<bool(T const&)> checker)
00254     {
00255         // Use pointer casts to avoid exceptions.
00256         auto other_t = dynamic_cast<T const*>(&other);
00257         if (!other_t) {
00258             return false;
00259         }
00260         // Pass to checker as a reference to clarify the object will exist.
00261         return checker(*other_t);
00262     }
00263
00264     private:
00265         shared_value delay_merge(std::vector<shared_value> stack, shared_value fallback) const;
00266         shared_origin _origin;
00267     };
00268 } // namespace hocon

```

8.16 config_value_factory.hpp

```

00001 # pragma once
00002
00003 #include "types.hpp"
00004 #include "export.h"
00005 #include <string>
00006
00007 namespace hocon {
00008     class LIBCPP_HOCON_EXPORT config_value_factory {
00009     public:
00010         static shared_value from_any_ref(unwrapped_value value, std::string origin_description = "");
00011     };
00012 } // namespace hocon

```

8.17 export.h

```

00001
00002 #ifndef LIBCPP_HOCON_EXPORT_H
00003 #define LIBCPP_HOCON_EXPORT_H
00004
00005 #ifndef LIBCPP_HOCON_STATIC_DEFINE
00006 # define LIBCPP_HOCON_EXPORT
00007 # define LIBCPP_HOCON_NO_EXPORT
00008 #else
00009 # undef LIBCPP_HOCON_EXPORT
00010 # undef LIBCPP_HOCON_NO_EXPORT
00011 # define LIBCPP_HOCON_EXPORT __attribute__((visibility("default")))
00012 # define LIBCPP_HOCON_NO_EXPORT __attribute__((visibility("hidden")))

```

```

00013 #     else
00014 #         /* We are using this library */
00015 #         define LIBCPP_HOCON_EXPORT __attribute__((visibility("default")))
00016 #     endif
00017 # endif
00018
00019 # ifndef LIBCPP_HOCON_NO_EXPORT
00020 #     define LIBCPP_HOCON_NO_EXPORT __attribute__((visibility("hidden")))
00021 # endif
00022 #endif
00023
00024 #ifndef LIBCPP_HOCON_DEPRECATED
00025 #     define LIBCPP_HOCON_DEPRECATED __attribute__((__deprecated__))
00026 #endif
00027
00028 #ifndef LIBCPP_HOCON_DEPRECATED_EXPORT
00029 #     define LIBCPP_HOCON_DEPRECATED_EXPORT LIBCPP_HOCON_EXPORT LIBCPP_HOCON_DEPRECATED
00030 #endif
00031
00032 #ifndef LIBCPP_HOCON_DEPRECATED_NO_EXPORT
00033 #     define LIBCPP_HOCON_DEPRECATED_NO_EXPORT LIBCPP_HOCON_NO_EXPORT LIBCPP_HOCON_DEPRECATED
00034 #endif
00035
00036 #if 0 /* DEFINE_NO_DEPRECATED */
00037 #     ifndef LIBCPP_HOCON_NO_DEPRECATED
00038 #         define LIBCPP_HOCON_NO_DEPRECATED
00039 #     endif
00040 #endif
00041
00042 #endif /* LIBCPP_HOCON_EXPORT_H */

```

8.18 functional_list.hpp

```

00001 #pragma once
00002
00003 #include <cassert>
00004 #include <memory>
00005 #include <functional>
00006 #include <initializer_list>
00007 #include <iterator>
00008 #include <iostream> // print
00009
00010 template<class T> class FwdListIter;
00011
00012 template<class T>
00013 class List
00014 {
00015     struct Item
00016     {
00017         Item(T v, std::shared_ptr<const Item> tail)
00018             : _val(v), _next(std::move(tail))
00019         {}
00020         // singleton
00021         explicit Item(T v) : _val(v) {}
00022         // ~Item() { std::cout << "~" << _val << std::endl; }
00023         T _val;
00024         std::shared_ptr<const Item> _next;
00025     };
00026     friend Item;
00027     explicit List(std::shared_ptr<const Item> items)
00028         : _head(std::move(items)) {}
00029 public:
00030     // Empty list
00031     List() {}
00032     // Cons
00033     List(T v, List const & tail)
00034         : _head(std::make_shared<Item>(v, tail._head)) {}
00035     // Singleton
00036     explicit List(T v) : _head(std::make_shared<Item>(v)) {}
00037
00038     bool isEmpty() const { return !_head; } // conversion to bool
00039     T front() const
00040     {
00041         assert(!isEmpty());
00042         return _head->_val;
00043     }
00044     List popped_front() const
00045     {
00046         assert(!isEmpty());
00047         return List(_head->_next);
00048     }
00049     // Additional utilities
00050     List pushed_front(T v) const

```



```

00051     {
00052         return List(v, *this);
00053     }
00054     List take(int n)
00055     {
00056         if (n <= 0 || isEmpty()) return List();
00057         return popped_front().take(n - 1).pushed_front(front());
00058     }
00059     List insertedAt(int i, T v) const
00060     {
00061         if (i == 0) {
00062             return pushed_front(v);
00063         } else {
00064             assert(!isEmpty());
00065             return List<T>(front(), popped_front().insertedAt(i - 1, v));
00066         }
00067     }
00068     List removed(T v) const
00069     {
00070         if (isEmpty()) return List();
00071         if (v == front())
00072             return popped_front().removed(v);
00073         return List(front(), popped_front().removed(v));
00074     }
00075     List removed1(T v) const
00076     {
00077         if (isEmpty()) return List();
00078         if (v == front())
00079             return popped_front();
00080         return List(front(), popped_front().removed(v));
00081     }
00082     bool member(T v) const
00083     {
00084         if (isEmpty()) return false;
00085         if (v == front()) return true;
00086         return popped_front().member(v);
00087     }
00088     template<class F>
00089     void forEach(F f) const
00090     {
00091         Item const * it = _head.get();
00092         while (it != nullptr)
00093         {
00094             f(it->_val);
00095             it = it->_next.get();
00096         }
00097     }
00098
00099     friend class FwdListIter<T>;
00100     // For debugging
00101     int headCount() const { return _head.use_count(); }
00102 private:
00103     std::shared_ptr<const Item> _head;
00104 };
00105
00106 template<class T, class P>
00107 bool all(List<T> const & lst, P & p)
00108 {
00109     if (lst.isEmpty())
00110         return true;
00111     if (!p(lst.front()))
00112         return false;
00113     return all(lst.popped_front(), p);
00114 }
00115
00116 template<class T>
00117 class FwdListIter : public std::iterator<std::forward_iterator_tag, T>
00118 {
00119 public:
00120     FwdListIter() {} // end
00121     FwdListIter(List<T> const & lst) : _cur(lst._head)
00122     {}
00123     T operator*() const { return _cur->_val; }
00124     FwdListIter & operator++()
00125     {
00126         _cur = _cur->_next;
00127         return *this;
00128     }
00129     bool operator==(FwdListIter<T> const & other)
00130     {
00131         return _cur == other._cur;
00132     }
00133     bool operator!=(FwdListIter<T> const & other)
00134     {
00135         return !(*this == other);
00136     }
00137 private:

```

```

00138     std::shared_ptr<const typename List<T>::Item> _cur;
00139 };
00140
00141 template<class T>
00142 class OutListIter : public std::iterator<std::output_iterator_tag, T>
00143 {
00144 public:
00145     OutListIter() {}
00146     T & operator*() { return _val; }
00147     OutListIter & operator++()
00148     {
00149         _lst = List<T>(_val, _lst);
00150         return *this;
00151     }
00152     List<T> getList() const { return _lst; }
00153 private:
00154     T _val;
00155     List<T> _lst;
00156 };
00157
00158
00159 namespace std
00160 {
00161     template<class T>
00162     FwdListIter<T> begin(List<T> const & lst)
00163     {
00164         return FwdListIter<T>(lst);
00165     }
00166     template<class T>
00167     FwdListIter<T> end(List<T> const & lst)
00168     {
00169         return FwdListIter<T>();
00170     }
00171 }
00172
00173 template<class T>
00174 List<T> concat(List<T> const & a, List<T> const & b)
00175 {
00176     if (a.isEmpty())
00177         return b;
00178     return List<T>(a.front(), concat(a.popped_front(), b));
00179 }
00180
00181 template<class T, class F>
00182 auto fmap(F f, List<T> lst) -> List<decltype(f(lst.front()))>
00183 {
00184     using U = decltype(f(lst.front()));
00185     static_assert(std::is_convertible<F, std::function<U(T)>::value,
00186         "fmap requires a function type U(T)");
00187     if (lst.isEmpty())
00188         return List<U>();
00189     else
00190         return List<U>(f(lst.front()), fmap(f, lst.popped_front()));
00191 }
00192
00193 template<class T, class P>
00194 List<T> filter(P p, List<T> lst)
00195 {
00196     static_assert(std::is_convertible<P, std::function<bool(T)>::value,
00197         "filter requires a function type bool(T)");
00198     if (lst.isEmpty())
00199         return List<T>();
00200     if (p(lst.front()))
00201         return List<T>(lst.front(), filter(p, lst.popped_front()));
00202     else
00203         return filter(p, lst.popped_front());
00204 }
00205
00206 template<class T, class U, class F>
00207 U foldr(F f, U acc, List<T> lst)
00208 {
00209     static_assert(std::is_convertible<F, std::function<U(T, U)>::value,
00210         "foldr requires a function type U(T, U)");
00211     if (lst.isEmpty())
00212         return acc;
00213     else
00214         return f(lst.front(), foldr(f, acc, lst.popped_front()));
00215 }
00216
00217 template<class T, class U, class F>
00218 U foldl(F f, U acc, List<T> lst)
00219 {
00220     static_assert(std::is_convertible<F, std::function<U(U, T)>::value,
00221         "foldl requires a function type U(U, T)");
00222     if (lst.isEmpty())
00223         return acc;
00224     else

```

```

00225         return foldl(f, f(acc, lst.front()), lst.popped_front());
00226     }
00227
00228 // Set difference a \ b
00229 template<class T>
00230 List<T> set_diff(List<T> const & as, List<T> const & bs)
00231 {
00232     return foldl([], (List<T> const & acc, T x) {
00233         return acc.removed(x);
00234     }, as, bs);
00235 }
00236
00237 // Set union of two lists, xs u ys
00238 // Assume no duplicates inside either list
00239 template<class T>
00240 List<T> set_union(List<T> const & xs, List<T> const & ys)
00241 {
00242     // xs u ys = (ys \ xs) ++ xs
00243     // removed all xs from ys
00244     auto trimmed = foldl([], (List<T> const & acc, T x) {
00245         return acc.removed(x);
00246     }, ys, xs);
00247     return concat(trimmed, xs);
00248 }
00249
00250 template<class T>
00251 List<T> concatAll(List<List<T>> const & xss)
00252 {
00253     if (xss.isEmpty()) return List<T>();
00254     return concat(xss.front(), concatAll(xss.popped_front()));
00255 }
00256
00257 // consumes the list when called:
00258 // forEach(std::move(lst), f);
00259
00260 template<class T, class F>
00261 void forEach(List<T> lst, F f)
00262 {
00263     static_assert(std::is_convertible<F, std::function<void(T)>>::value,
00264         "forEach requires a function type void(T)");
00265     while (!lst.isEmpty()) {
00266         f(lst.front());
00267         lst = lst.popped_front();
00268     }
00269 }
00270
00271 template<class Beg, class End>
00272 auto fromIt(Beg it, End end) -> List<typename Beg::value_type>
00273 {
00274     typedef typename Beg::value_type T;
00275     if (it == end)
00276         return List<T>();
00277     T item = *it;
00278     return List<T>(item, fromIt(++it, end));
00279 }
00280
00281 template<class T, class F>
00282 List<T> iterateN(F f, T init, int count)
00283 {
00284     if (count <= 0) return List<T>();
00285     return iterateN(f, f(init), count - 1).pushed_front(init);
00286 }
00287
00288 // Pass lst by value not reference!
00289 template<class T>
00290 void printRaw(List<T> lst)
00291 {
00292     if (lst.isEmpty()) {
00293         std::cout << std::endl;
00294     } else {
00295         std::cout << "(" << lst.front() << ", " << lst.headCount() - 1 << ") ";
00296         printRaw(lst.popped_front());
00297     }
00298 }
00299
00300 template<class T>
00301 std::ostream& operator<<(std::ostream& os, List<T> const & lst)
00302 {
00303     os << "[";
00304     forEach(lst, [&os](T v) {
00305         os << v << " ";
00306     });
00307     os << "]";
00308     return os;
00309 }
00310
00311 template<class T>

```

```

00312 List<T> reversed(List<T> const & lst)
00313 {
00314     return foldl({}(List<T> const & acc, T v)
00315                 {
00316                     return List<T>(v, acc);
00317                 }, List<T>(), lst);
00318 }

```

8.19 config_document.hpp

```

00001 #pragma once
00002
00003 #include <string>
00004 #include <memory>
00005 #include <hocon/config_value.hpp>
00006 #include "../export.h"
00007
00008 namespace hocon {
00026     class LIBCPP_HOCON_EXPORT config_document {
00027     public:
00046         virtual std::unique_ptr<config_document> with_value_text(std::string path, std::string
newValue) const = 0;
00047
00060         virtual std::unique_ptr<config_document> with_value(std::string path,
std::shared_ptr<config_value> new_value)
00061         const = 0;
00062
00072         virtual std::unique_ptr<config_document> without_path(std::string path) const = 0;
00073
00080         virtual bool has_path(std::string const& path) const = 0;
00081
00087         virtual std::string render() const = 0;
00088     };
00089
00093     bool operator==(config_document const& lhs, config_document const& rhs);
00094
00095 } // namespace hocon

```

8.20 config_document_factory.hpp

```

00001 #pragma once
00002
00003 #include "config_document.hpp"
00004 #include <hocon/config_parse_options.hpp>
00005 #include "../export.h"
00006
00010 namespace hocon { namespace config_document_factory {
00011
00021     LIBCPP_HOCON_EXPORT std::shared_ptr<config_document> parse_file(std::string input_file_path,
config_parse_options options);
00022
00023     LIBCPP_HOCON_EXPORT std::shared_ptr<config_document> parse_file(std::string input_file_path);
00026
00034     LIBCPP_HOCON_EXPORT std::shared_ptr<config_document> parse_string(std::string s,
config_parse_options options);
00035
00037     LIBCPP_HOCON_EXPORT std::shared_ptr<config_document> parse_string(std::string s);
00038
00039 }} // namespace hocon::config_document_factory

```

8.21 config_node.hpp

```

00001 #pragma once
00002
00003 #include <string>
00004 #include "../export.h"
00005
00006 namespace hocon {
00021     class LIBCPP_HOCON_EXPORT config_node {
00022     public:
00029         virtual std::string render() const = 0;
00030     };
00031
00032 } // namespace hocon

```

8.22 path.hpp

```

00001 #pragma once
00002
00003 #include <hocon/functional_list.hpp>
00004 #include <hocon/types.hpp>
00005
00006 #include <string>
00007 #include <vector>
00008 #include <memory>
00009 #include "export.h"
00010
00011 namespace hocon {
00012
00013     class LIBCPP_HOCON_EXPORT path {
00014     public:
00015         path();
00016         explicit path(std::string first, path const& remainder);
00017         explicit path(std::vector<std::string> elements);
00018         explicit path(std::vector<path> paths_to_concat);
00019
00020         shared_string first() const;
00021
00022         path remainder() const;
00023
00024         path parent() const;
00025
00026         bool has_remainder() const;
00027         bool empty() const;
00028         shared_string last() const;
00029         path prepend(path prefix);
00030         int length() const;
00031         path sub_path(int remove_from_front);
00032         path sub_path(int start_index, int end_index);
00033         bool starts_with(path other) const;
00034
00035         bool operator==(path const& other) const;
00036         bool operator!=(path const& other) const;
00037
00038         static bool has_funky_chars(std::string const& s);
00039
00040         void append_to_string(std::string& base) const;
00041
00042         std::string to_string() const;
00043
00044         std::string render() const;
00045
00046         static path new_key(std::string key);
00047         static path new_path(std::string path);
00048
00049     private:
00050         path(List<shared_string> other_path);
00051         List<shared_string> _path;
00052     };
00053
00054 } // namespace hocon

```

8.23 program_options.hpp

```

00001 #pragma once
00002
00003 #include "config.hpp"
00004 #include "config_list.hpp"
00005
00006 #include <boost/format.hpp>
00007 #include <boost/program_options.hpp>
00008
00009 namespace hocon { namespace program_options {
00010     namespace po = boost::program_options;
00011
00012     template<class charT>
00013     po::basic_parsed_options<charT>
00014     parse_hocon(shared_config cfg, const po::options_description& desc, bool allow_unregistered) {
00015         po::parsed_options result(&desc);
00016
00017         std::map<std::string, shared_config> to_do;
00018         to_do.emplace("", cfg);
00019
00020         while (!to_do.empty()) {
00021             auto iter = to_do.begin();
00022             auto prefix = iter->first;
00023             auto cfg = iter->second;

```

```

00024         to_do.erase(iter);
00025
00026         for (const auto& entry : cfg->entry_set()) {
00027             po::option opt;
00028             if (prefix.empty()) {
00029                 opt.string_key = entry.first;
00030             } else {
00031                 opt.string_key = prefix + "." + entry.first;
00032             }
00033
00034             // Skips options that are not registered in the description.
00035             // This is different behavior than the built in program_options
00036             // parsers, which pass the options along somehow. But for now
00037             // it stops an exception from being thrown if there is an unknown
00038             // item in the config file.
00039             if (allow_unregistered && !desc.find_nothrow(opt.string_key, false)) {
00040                 continue;
00041             }
00042
00043             if (entry.second->value_type() == config_value::type::LIST) {
00044                 // if this is a list, we want to check if any of the entries are
00045                 // objects. If so, we need to further expand those sub-trees, with
00046                 // list indices being expanded into our key
00047
00048                 auto list = std::dynamic_pointer_cast<const config_list>(entry.second);
00049                 for (size_t i = 0; i < list->size(); ++i) {
00050                     const auto& value = list->get(i);
00051                     if (value->value_type() == config_value::type::LIST ||
00052                         value->value_type() == config_value::type::OBJECT) {
00053                         boost::throw_exception(po::invalid_config_file_syntax(list->transform_to_string(),
00054                             po::invalid_syntax::unrecognized_line));
00055                     } else {
00056                         opt.value.push_back(value->transform_to_string());
00057                     }
00058                 } else {
00059                     opt.value.push_back(entry.second->transform_to_string());
00060                 }
00061                 if (!opt.value.empty()) {
00062                     result.options.emplace_back(std::move(opt));
00063                 }
00064             }
00065         }
00066
00067         // let Boost convert our utf-8 entries to wchars if needed
00068         return po::basic_parsed_options<charT>(result);
00069     }
00070
00071     template<class charT>
00072     po::basic_parsed_options<charT>
00073     parse_file(std::basic_string<charT> file_basename, const po::options_description& desc, bool
00074         allow_unregistered=false) {
00075         shared_config cfg = config::parse_file_any_syntax(file_basename)->resolve();
00076         return parse_hocon<charT>(cfg, desc, allow_unregistered);
00077     }
00078
00079     template<class charT>
00080     po::basic_parsed_options<charT>
00081     parse_string(std::basic_string<charT> s, const po::options_description& desc, bool
00082         allow_unregistered=false) {
00083         shared_config cfg = config::parse_string(s)->resolve();
00084         return parse_hocon<charT>(cfg, desc, allow_unregistered);
00085     }
00086 }

```

8.24 types.hpp

```

00001 #pragma once
00002
00003 #include <memory>
00004 #include <vector>
00005 #include <unordered_map>
00006 #pragma GCC diagnostic push
00007 #pragma GCC diagnostic ignored "-Waddress"
00008 #if defined(__GNUC__) && __GNUC__ > 5
00009 #pragma GCC diagnostic ignored "-Wnonnull-compare"
00010 #endif
00011 #include "boost/variant.hpp"
00012 #pragma GCC diagnostic pop
00013
00014 namespace hocon {
00015

```

```

00021     using duration = std::pair<int64_t, int>;
00022
00023     class config;
00024     using shared_config = std::shared_ptr<const config>;
00025
00026     class config_object;
00027     using shared_object = std::shared_ptr<const config_object>;
00028
00029     class config_origin;
00030     using shared_origin = std::shared_ptr<const config_origin>;
00031
00032     class path;
00033
00034     class config_value;
00035     using shared_value = std::shared_ptr<const config_value>;
00036
00037     class config_list;
00038     using shared_list = std::shared_ptr<const config_list>;
00039
00040     typedef boost::make_recursive_variant<boost::blank, std::string, int64_t, double, int, bool,
00041         std::vector<boost::recursive_variant_>, std::unordered_map<std::string,
00042             boost::recursive_variant_>::type unwrapped_value;
00043
00044     class container;
00045     using shared_container = std::shared_ptr<const container>;
00046
00047     class abstract_config_node;
00048     using shared_node = std::shared_ptr<const abstract_config_node>;
00049     using shared_node_list = std::vector<shared_node>;
00050
00051     using shared_string = std::shared_ptr<const std::string>;
00052
00053     class config_parse_options;
00054
00055     class config_includer;
00056     using shared_includer = std::shared_ptr<const config_includer>;
00057
00058     class config_include_context;
00059     using shared_include_context = std::shared_ptr<const config_include_context>;
00060
00061     class config_parseable;
00062     using shared_parseable = std::shared_ptr<const config_parseable>;
00063 } // namespace hocon

```

8.25 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).

8.25.1 Detailed Description

Declares macros for the cpp-hocon library version.

Definition in file [version.h](#).

8.25.2 Macro Definition Documentation

8.25.2.1 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](#).

8.25.2.2 CPP_HOCON_VERSION_MAJOR

```
#define CPP_HOCON_VERSION_MAJOR 0
```

The cpp-hocon library major version.

Definition at line 10 of file [version.h](#).

8.25.2.3 CPP_HOCON_VERSION_MINOR

```
#define CPP_HOCON_VERSION_MINOR 3
```

The cpp-hocon library minor version.

Definition at line 14 of file [version.h](#).

8.25.2.4 CPP_HOCON_VERSION_PATCH

```
#define CPP_HOCON_VERSION_PATCH 0
```

The cpp-hocon library patch version.

Definition at line 18 of file [version.h](#).

8.25.2.5 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](#).

8.26 version.h

[Go to the documentation of this file.](#)

```
00001
00005 #pragma once
00006
00010 #define CPP_HOCON_VERSION_MAJOR 0
00014 #define CPP_HOCON_VERSION_MINOR 3
00018 #define CPP_HOCON_VERSION_PATCH 0
00019
00023 #define CPP_HOCON_VERSION "0.3.0"
00024
00028 #define CPP_HOCON_VERSION_WITH_COMMIT "0.3.0"
```


8.27 config_document_parser.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_parse_options.hpp>
00004 #include <internal/nodes/config_node_object.hpp>
00005 #include <internal/nodes/config_node_root.hpp>
00006 #include <internal/nodes/config_node_include.hpp>
00007 #include <internal/simple_config_origin.hpp>
00008 #include <internal/tokenizer.hpp>
00009
00010 #include <stack>
00011
00012 namespace hocon { namespace config_document_parser {
00013
00014     std::shared_ptr<config_node_root> parse(token_iterator tokens,
00015                                           shared_origin origin,
00016                                           config_parse_options options);
00017
00018     shared_node_value parse_value(token_iterator tokens,
00019                                   shared_origin origin,
00020                                   config_parse_options options);
00021
00022     class parse_context {
00023     public:
00024         parse_context(config_syntax flavor, shared_origin origin, token_iterator tokens);
00025
00026         std::shared_ptr<config_node_root> parse();
00027
00031         shared_node_value parse_single_value();
00032
00033     private:
00034         parse_exception parse_error(std::string message);
00035
00036         shared_token pop_token();
00037         shared_token next_token();
00038         shared_token next_token_collecting_whitespace(shared_node_list& nodes);
00039         void put_back(shared_token token);
00040
00049         bool check_element_separator(shared_node_list& nodes);
00050
00052         shared_node_value consolidate_values(shared_node_list& nodes);
00053
00054         std::string add_quote_suggestion(std::string bad_token, std::string message,
00055                                         bool inside_equals, path* last_path);
00056
00057         std::string add_quote_suggestion(std::string bad_token, std::string message);
00058
00059         shared_node_value parse_value(shared_token t);
00060         std::shared_ptr<config_node_path> parse_key(shared_token t);
00061         bool is_key_value_separator(shared_token t);
00062         std::shared_ptr<config_node_include> parse_include(shared_node_list& children);
00063         std::shared_ptr<config_node_complex_value> parse_object(bool had_open_curly);
00064         std::shared_ptr<config_node_complex_value> parse_array();
00065
00066         static bool is_include_keyword(shared_token t);
00067         static bool is_unquoted_whitespace(shared_token t);
00068         static bool is_valid_array_element(shared_token t);
00069
00070         int _line_number;
00071         std::stack<shared_token> _buffer;
00072         token_iterator _tokens;
00073         config_syntax _flavor;
00074         shared_origin _base_origin;
00075
00076         // this is the number of "equals" we are inside,
00077         // used to modify the error message to reflect that
00078         // someone may think this is .properties format.
00079         int _equals_count;
00080     };
00081
00082
00083 }} // namespace hocon::config_document_parser

```

8.28 config_parser.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_value.hpp>
00004 #include <hocon/config_syntax.hpp>
00005 #include <hocon/config_include_context.hpp>
00006 #include <hocon/config_parse_options.hpp>
00007 #include <hocon/path.hpp>

```

```

00008 #include <internal/nodes/config_node_concatenation.hpp>
00009 #include <internal/nodes/config_node_root.hpp>
00010 #include <internal/nodes/abstract_config_node_value.hpp>
00011 #include <internal/nodes/config_node_object.hpp>
00012 #include <internal/nodes/config_node_array.hpp>
00013 #include <internal/nodes/config_node_include.hpp>
00014 #include <internal/full_includer.hpp>
00015 #include <memory>
00016 #include <vector>
00017 #include <unordered_map>
00018 #include <string>
00019
00020 namespace hocon { namespace config_parser {
00021
00022     shared_value parse(std::shared_ptr<const config_node_root> document,
00023                       shared_origin origin,
00024                       config_parse_options options,
00025                       shared_include_context include_context);
00026
00027     class parse_context {
00028     public:
00029         int _line_number;
00030         std::shared_ptr<const config_node_root> _document;
00031         std::shared_ptr<const full_includer> _includer;
00032         shared_include_context _include_context;
00033         config_syntax _flavor;
00034         shared_origin _base_origin, _line_origin;
00035         std::vector<path> _path_stack;
00036
00037         parse_context(config_syntax flavor, shared_origin origin, std::shared_ptr<const
00038 config_node_root> document,
00039                       std::shared_ptr<const full_includer> includer, shared_include_context
00040 include_context);
00041
00042         shared_value parse();
00043
00044         int array_count;
00045
00046     private:
00047         static shared_object create_value_under_path(path p, shared_value value);
00048         shared_origin line_origin() const;
00049         path full_current_path() const;
00050         shared_value parse_value(shared_node_value n, std::vector<std::string>& comments);
00051         void parse_include(std::unordered_map<std::string, shared_value> &values,
00052 std::shared_ptr<const config_node_include> n);
00053         shared_object parse_object(shared_node_object n);
00054         shared_value parse_array(shared_node_array n);
00055         shared_value parse_concatenation(shared_node_concatenation n);
00056     };
00057 } } // namespace hocon::config_parser

```

8.29 config_util.hpp

```

00001 #pragma once
00002
00003 #include <string>
00004
00005 namespace hocon {
00006
00007     bool is_whitespace(char codepoint);
00008
00009     bool is_whitespace_not_newline(char codepoint);
00010
00011     bool is_C0_control(char c);
00012
00013     std::string render_json_string(std::string const& s);
00014
00015     std::string render_string_unquoted_if_possible(std::string const& s);
00016
00017 } // namespace hocon

```

8.30 container.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_value.hpp>
00004 namespace hocon {
00005

```

```

00012     class container {
00013     public:
00019         virtual shared_value replace_child(shared_value const& child, shared_value replacement) const
= 0;
00020
00025         virtual bool has_descendant(shared_value const& descendant) const = 0;
00026     };
00027
00028 } // namespace hocon

```

8.31 default_transformer.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_value.hpp>
00004
00005 namespace hocon {
00006
00007     class default_transformer {
00008     public:
00009         static shared_value transform(shared_value value, config_value::type requested);
00010     };
00011
00012 } // namespace hocon

```

8.32 full_includer.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_includer.hpp>
00004 #include <hocon/config_includer_file.hpp>
00005
00006
00007 namespace hocon {
00008
00009     class full_includer : public config_includer, public config_includer_file {
00010         // This is a faithful port of the Java fullIncluder interface
00011     };
00012
00013 } // namespace hocon

```

8.33 abstract_config_node.hpp

```

00001 #pragma once
00002
00003 #include <hocon/parser/config_node.hpp>
00004 #include <internal/token.hpp>
00005 #include <vector>
00006
00007 namespace hocon {
00008
00009     class abstract_config_node : public config_node {
00010     public:
00011         std::string render() const;
00012         bool operator==(const abstract_config_node &other) const;
00013         virtual token_list get_tokens() const = 0;
00014     };
00015
00016 } // namespace hocon

```

8.34 abstract_config_node_value.hpp

```

00001 #pragma once
00002
00003 #include "abstract_config_node.hpp"
00004
00005 namespace hocon {
00006
00008     class abstract_config_node_value : public abstract_config_node { };
00009
00010     using shared_node_value = std::shared_ptr<const abstract_config_node_value>;
00011
00012 } // namespace hocon

```

8.35 config_node_array.hpp

```

00001 #pragma once
00002
00003 #include "config_node_complex_value.hpp"
00004
00005 namespace hocon {
00006
00007     class config_node_array;
00008     using shared_node_array = std::shared_ptr<const config_node_array>;
00009
00010     class config_node_array : public config_node_complex_value {
00011     public:
00012         config_node_array(shared_node_list children);
00013
00014         std::shared_ptr<const config_node_complex_value> new_node(shared_node_list nodes) const
00015         override;
00016     };
00017 } // namespace hocon

```

8.36 config_node_comment.hpp

```

00001 #pragma once
00002
00003 #include "config_node_single_token.hpp"
00004
00005 namespace hocon {
00006
00007     class config_node_comment : public config_node_single_token {
00008     public:
00009         config_node_comment(shared_token comment);
00010
00011         std::string comment_text() const;
00012     };
00013
00014 } // namespace hocon

```

8.37 config_node_complex_value.hpp

```

00001 #pragma once
00002
00003 #include "abstract_config_node_value.hpp"
00004
00005 namespace hocon {
00006
00007     class config_node_complex_value : public abstract_config_node_value {
00008     public:
00009         config_node_complex_value(shared_node_list children);
00010
00011         token_list get_tokens() const override;
00012
00013         shared_node_list const& children() const;
00014
00015         std::shared_ptr<const config_node_complex_value> indent_text(
00016             shared_node indentation) const;
00017
00018         virtual std::shared_ptr<const config_node_complex_value> new_node(
00019             shared_node_list nodes) const = 0;
00020
00021     private:
00022         shared_node_list _children;
00023     };
00024
00025 } // namespace hocon

```

8.38 config_node_concatenation.hpp

```

00001 #pragma once
00002
00003 #include "config_node_complex_value.hpp"
00004
00005 namespace hocon {
00006
00007     class config_node_concatenation : public config_node_complex_value {

```

```

00008     public:
00009         config_node_concatenation(shared_node_list children);
00010
00011         std::shared_ptr<const config_node_complex_value> new_node(shared_node_list nodes) const
00012         override;
00013     };
00014     using shared_node_concatenation = std::shared_ptr<const config_node_concatenation>;
00015
00016 } // namespace hocon

```

8.39 config_node_field.hpp

```

00001 #pragma once
00002
00003 #include "abstract_config_node_value.hpp"
00004 #include "config_node_path.hpp"
00005
00006 namespace hocon {
00007
00008     class config_node_field : public abstract_config_node_value {
00009     public:
00010         config_node_field(shared_node_list children);
00011
00012         token_list get_tokens() const override;
00013
00014         std::shared_ptr<const config_node_field> replace_value(shared_node_value new_value) const;
00015         shared_node_value get_value() const;
00016         shared_token separator() const;
00017         std::vector<std::string> comments() const;
00018         std::shared_ptr<const config_node_path> path() const;
00019
00020     private:
00021         shared_node_list _children;
00022     };
00023
00024 } // namespace hocon

```

8.40 config_node_include.hpp

```

00001 #pragma once
00002
00003 #include "abstract_config_node.hpp"
00004
00005 namespace hocon {
00006
00007     enum class config_include_kind { URL, FILE, CLASSPATH, HEURISTIC };
00008
00009     class config_node_include : public abstract_config_node {
00010     public:
00011         config_node_include(shared_node_list children,
00012                             config_include_kind kind);
00013
00014         token_list get_tokens() const override;
00015
00016         shared_node_list const& children() const;
00017         config_include_kind kind() const;
00018         std::string name() const;
00019
00020     private:
00021         shared_node_list _children;
00022         config_include_kind _kind;
00023     };
00024
00025 } // namespace hocon

```

8.41 config_node_object.hpp

```

00001 #pragma once
00002
00003 #include "config_node_complex_value.hpp"
00004 #include "config_node_path.hpp"
00005 #include <hocon/path.hpp>
00006 #include <hocon/config_syntax.hpp>

```

```

00007
00008 namespace hocon {
00009
00010     class config_node_object;
00011     using shared_node_object = std::shared_ptr<const config_node_object>;
00012
00013     class config_node_object : public config_node_complex_value {
00014     public:
00015         config_node_object(shared_node_list children);
00016
00017         std::shared_ptr<const config_node_complex_value> new_node(
00018             shared_node_list nodes) const override;
00019
00020         bool has_value(path desired_path) const;
00021
00022         shared_node_object change_value_on_path(path desired_path,
00023             shared_node_value value,
00024             config_syntax flavor) const;
00025
00026         shared_node_object set_value_on_path(std::string desired_path,
00027             shared_node_value value,
00028             config_syntax flavor = config_syntax::CONF) const;
00029
00030         shared_node_object set_value_on_path(config_node_path desired_path,
00031             shared_node_value value,
00032             config_syntax flavor = config_syntax::CONF) const;
00033
00034         shared_node_list indentation() const;
00035
00036         shared_node_object add_value_on_path(config_node_path desired_path,
00037             shared_node_value value,
00038             config_syntax flavor) const;
00039
00040         shared_node_object remove_value_on_path(std::string desired_path, config_syntax flavor) const;
00041
00042         static bool contains_token(shared_node node, token_type token);
00043     };
00044
00045 } // namespace hocon

```

8.42 config_node_path.hpp

```

00001 #pragma once
00002
00003 #include <internal/nodes/abstract_config_node.hpp>
00004 #include <hocon/path.hpp>
00005 #include <internal/token.hpp>
00006
00007 namespace hocon {
00008
00009     class config_node_path : public abstract_config_node {
00010     public:
00011         config_node_path(path node_path, token_list tokens);
00012
00013         token_list get_tokens() const override;
00014         path get_path() const;
00015
00016         config_node_path sub_path(int to_remove);
00017         config_node_path first();
00018
00019     private:
00020         path _path;
00021         token_list _tokens;
00022     };
00023
00024
00025 } // namespace hocon

```

8.43 config_node_root.hpp

```

00001 #pragma once
00002
00003 #include "config_node_complex_value.hpp"
00004 #include <hocon/config_syntax.hpp>
00005
00006 namespace hocon {
00007
00008     class config_node_root : public config_node_complex_value {
00009     public:

```

```

00010         config_node_root(shared_node_list children, shared_origin origin);
00011
00012         std::shared_ptr<const config_node_complex_value> new_node(shared_node_list nodes) const
00013         override;
00014
00014         std::shared_ptr<const config_node_complex_value> value() const;
00015         std::shared_ptr<const config_node_root> set_value(std::string desired_path,
00016                                                         shared_node_value,
00017                                                         config_syntax flavor) const;
00018         bool has_value(std::string desired_path) const;
00019
00020     private:
00021         shared_origin _origin;
00022     };
00023
00024 } // namespace hocon

```

8.44 config_node_simple_value.hpp

```

00001 #pragma once
00002
00003 #include "abstract_config_node_value.hpp"
00004 #include <hocon/config_value.hpp>
00005
00006 namespace hocon {
00007
00008     class config_node_simple_value : public abstract_config_node_value {
00009     public:
00010         config_node_simple_value(shared_token value);
00011
00012         shared_token get_token() const;
00013         shared_value get_value() const;
00014
00015         token_list get_tokens() const override;
00016
00017     private:
00018         shared_token _token;
00019     };
00020
00021 } // namespace hocon

```

8.45 config_node_single_token.hpp

```

00001 #pragma once
00002
00003 #include <internal/token.hpp>
00004 #include "abstract_config_node.hpp"
00005
00006 namespace hocon {
00007
00008     class config_node_single_token : public abstract_config_node {
00009     public:
00010         config_node_single_token(shared_token t);
00011
00012         token_list get_tokens() const override;
00013
00014         shared_token get_token() const;
00015
00016     private:
00017         shared_token _token;
00018     };
00019
00020 } // namespace hocon

```

8.46 parseable.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_parseable.hpp>
00004 #include <boost/nowide/fstream.hpp>
00005 #include <internal/simple_config_origin.hpp>
00006 #include <hocon/config_object.hpp>
00007 #include <hocon/config_include_context.hpp>
00008
00009 namespace hocon {

```

```

00010
00011     class config_document;
00012
00013     class parseable : public config_parseable, public std::enable_shared_from_this<parseable> {
00014     public:
00015         static std::shared_ptr<parseable> new_file(std::string input_file_path, config_parse_options
options);
00016         static std::shared_ptr<parseable> new_string(std::string s, config_parse_options options);
00017         static std::shared_ptr<parseable> new_not_found(std::string what_not_found, std::string
message,
00018                                                         config_parse_options options);
00019
00020         static config_syntax syntax_from_extension(std::string name);
00021
00022         void post_construct(config_parse_options const& base_options);
00023
00024         std::shared_ptr<config_document> parse_config_document();
00025         shared_object parse(config_parse_options const& options) const override;
00026         shared_object parse() const;
00027
00028         shared_value parse_value() const;
00029
00030         config_parse_options const& options() const override;
00031         std::shared_ptr<const config_origin> origin() const override;
00032
00033         virtual std::unique_ptr<std::istream> reader(config_parse_options const& options) const;
00034         virtual std::unique_ptr<std::istream> reader() const = 0;
00035         virtual shared_origin create_origin() const = 0;
00036
00037         virtual config_syntax guess_syntax() const;
00038         virtual config_syntax content_type() const;
00039         virtual std::shared_ptr<config_parseable> relative_to(std::string file_name) const;
00040
00041         std::string to_string() const;
00042         std::string get_cur_dir() const;
00043         void set_cur_dir(std::string dir) const;
00044         void separate_filepath(const std::string& path, std::string* file_dir,
                                std::string* file_name) const;
00045
00046         // Disable copy constructors, as include_context assumes it can hold a reference to parseable.
00047         parseable() = default;
00048         parseable(parseable const&) = delete;
00049         parseable& operator=(parseable const&) = delete;
00050
00051     private:
00052         std::shared_ptr<config_document> parse_document(config_parse_options const& base_options)
const;
00053
00054         std::shared_ptr<config_document> parse_document(shared_origin origin,
00055                                                         config_parse_options const& final_options)
const;
00056         std::shared_ptr<config_document> raw_parse_document(std::unique_ptr<std::istream> stream,
00057                                                             shared_origin origin,
00058                                                         config_parse_options const& options)
const;
00059         std::shared_ptr<config_document> raw_parse_document(shared_origin origin,
00060                                                         config_parse_options const& options)
const;
00061
00062         shared_value parse_value(config_parse_options const& base_options) const;
00063         shared_value parse_value(shared_origin origin, config_parse_options const& options) const;
00064         shared_value raw_parse_value(std::unique_ptr<std::istream> stream,
00065                                     shared_origin origin,
00066                                     config_parse_options const& options) const;
00067         shared_value raw_parse_value(shared_origin origin, config_parse_options const& options) const;
00068
00069         config_parse_options fixup_options(config_parse_options const& base_options) const;
00070
00071         std::vector<parseable> _parse_stack;
00072
00073         shared_origin _initial_origin;
00074         config_parse_options _initial_options;
00075         shared_include_context _include_context;
00076
00077         static const int MAX_INCLUDE_DEPTH;
00078     };
00079
00080     class parseable_file : public parseable {
00081     public:
00082         parseable_file(std::string input_file_path, config_parse_options options);
00083         std::unique_ptr<std::istream> reader() const override;
00084         shared_origin create_origin() const override;
00085         config_syntax guess_syntax() const override;
00086
00087     private:
00088         std::string _input;
00089     };

```



```

00090     class parseable_string : public parseable {
00091     public:
00092         parseable_string(std::string s, config_parse_options options);
00093         std::unique_ptr<std::istream> reader() const override;
00094         shared_origin create_origin() const override;
00095
00096     private:
00097         std::string _input;
00098     };
00099
00100     // NOTE: this is not a faithful port of the 'ParseableResources' class from the
00101     // upstream, because at least for now we're not going to try to do anything
00102     // crazy like look for files on the ruby load path. However, there is a decent
00103     // chunk of logic elsewhere in the codebase that is written with the assumption
00104     // that this class will provide the 'last resort' attempt to find a config file
00105     // before giving up, so we're basically port just enough to have it provide
00106     // that last resort behavior
00107     class parseable_resources : public parseable {
00108     public:
00109         parseable_resources(std::string resource, config_parse_options options);
00110
00111         std::unique_ptr<std::istream> reader() const override;
00112         shared_origin create_origin() const override;
00113
00114     private:
00115         std::string _resource;
00116     };
00117
00118     // this is a parseable that doesn't exist and just throws when you try to
00119     // parse it
00120     class parseable_not_found : public parseable {
00121     public:
00122         parseable_not_found(std::string what, std::string message, config_parse_options options);
00123
00124         std::unique_ptr<std::istream> reader() const override;
00125         shared_origin create_origin() const override;
00126
00127     private:
00128         std::string _what;
00129         std::string _message;
00130     };
00131
00132 } // namespace hocon

```

8.47 path_builder.hpp

```

00001 #pragma once
00002
00003 #include <hocon/path.hpp>
00004 #include <stack>
00005
00006 namespace hocon {
00007
00008     class path_builder {
00009     public:
00010         void append_key(std::string key);
00011         void append_path(path path_to_append);
00012
00013         path result();
00014
00015     private:
00016         std::stack<std::string> _keys;
00017     };
00018
00019 } // namespace hocon
00020

```

8.48 path_parser.hpp

```

00001 #pragma once
00002
00003 #include "path_builder.hpp"
00004 #include "simple_config_origin.hpp"
00005 #include "tokenizer.hpp"
00006 #include <hocon/config_syntax.hpp>
00007 #include <internal/nodes/config_node_path.hpp>
00008
00009 #include <string>
00010 #include <vector>
00011

```

```

00012 namespace hocon {
00013
00014     class path_parser {
00015     public:
00016         static config_node_path parse_path_node(std::string const& path_string,
00017                                                 config_syntax flavor = config_syntax::CONF);
00018
00019         static path parse_path(std::string const& path_string);
00020
00021         static path parse_path_expression(iterator& expression, shared_origin origin,
00022                                         std::string const& original_text = "",
00023                                         token_list* path_tokens = nullptr,
00024                                         config_syntax flavor = config_syntax::CONF);
00025
00026         static config_node_path parse_path_node_expression(iterator& expression,
00027                                                           shared_origin origin,
00028                                                           std::string const& original_text = "",
00029                                                           config_syntax flavor =
00030                                                           config_syntax::CONF);
00031
00032     private:
00033         class element {
00034         public:
00035             element(std::string initial, bool can_be_empty);
00036
00037             std::string to_string() const;
00038
00039             std::string _value;
00040             bool _can_be_empty;
00041         };
00042
00043         static token_list split_token_on_period(shared_token t, config_syntax flavor);
00044
00045         static void add_path_text(std::vector<element>& buff, bool was_quoted, std::string new_text);
00046
00047         static bool looks_unsafe_for_fast_parser(std::string s);
00048
00049         static path fast_path_build(path tail, std::string s);
00050
00051         static path speculative_fast_parse_path(std::string const& path);
00052
00053         static const shared_origin api_origin;
00054     };
00055 } // namespace hocon

```

8.49 replaceable_merge_stack.hpp

```

00001 #pragma once
00002
00003 #include <internal/resolve_context.hpp>
00004 #include <internal/container.hpp>
00005
00006 namespace hocon {
00007
00008     // This is a faithful port of the Java ReplaceableMergeStack interface
00009     class replaceable_merge_stack : public container {
00010     public:
00011         virtual shared_value make_replacement(resolve_context const& context, int skipping) const = 0;
00012     };
00013
00014 } // namespace hocon

```

8.50 resolve_context.hpp

```

00001 #pragma once
00002
00003 #include <hocon/types.hpp>
00004 #include <hocon/config_resolve_options.hpp>
00005 #include <hocon/path.hpp>
00006
00007 #include <unordered_map>
00008
00009 namespace hocon {
00010
00011     class resolve_source;
00012
00013     template<typename T>
00014     struct resolve_result;

```

```

00015
00016     class resolve_context {
00017     public:
00018         resolve_context(config_resolve_options options, path restrict_to_child,
std::vector<shared_value> cycle_markers);
00019         resolve_context(config_resolve_options options, path restrict_to_child);
00020         bool is_restricted_to_child() const;
00021         config_resolve_options options() const;
00022
00023         resolve_result<shared_value> resolve(shared_value original, resolve_source const& source)
const;
00024         path restrict_to_child() const;
00025
00026         resolve_context add_cycle_marker(shared_value value) const;
00027         resolve_context remove_cycle_marker(shared_value value);
00028         resolve_context restrict(path restrict_to) const;
00029         resolve_context unrestricted() const;
00030
00031         static shared_value resolve(shared_value value, shared_object root, config_resolve_options
options);
00032
00033     private:
00034         struct memo_key {
00035             shared_value value;
00036             path restrict_to_child;
00037             bool operator==(const memo_key& other) const {
00038                 return value == other.value && restrict_to_child == other.restrict_to_child;
00039             }
00040         };
00041
00042         struct memo_key_hash {
00043             std::size_t operator()(const memo_key&) const;
00044         };
00045         using resolve_memos = std::unordered_map<memo_key, shared_value, memo_key_hash>;
00046         config_resolve_options _options;
00047         path _restrict_to_child;
00048         resolve_memos _memos;
00049         std::vector<shared_value> _cycle_markers;
00050
00051         resolve_context memoize(const memo_key& key, const shared_value& value) const;
00052     };
00053 } // namespace hocon

```

8.51 resolve_result.hpp

```

00001 #pragma once
00002 #include <internal/resolve_context.hpp>
00003 #include <memory>
00004
00005 namespace hocon {
00006
00007     template<typename V>
00008     struct resolve_result {
00009         resolve_result(resolve_context c, V v) :
00010             context(std::move(c)), value(std::move(v)) {}
00011
00012         resolve_context context;
00013         V value;
00014     };
00015
00016     template<typename T>
00017     static resolve_result<shared_value> make_resolve_result(resolve_context context, T value) {
00018         return resolve_result<shared_value>(std::move(context), std::move(value));
00019     }
00020
00021 } // namespace hocon

```

8.52 resolve_source.hpp

```

00001 #pragma once
00002
00003 #include <memory>
00004 #include <list>
00005 #include <hocon/types.hpp>
00006 #include <internal/resolve_result.hpp>
00007 #include <hocon/config_exception.hpp>
00008
00009 namespace hocon {
00010

```

```

00011     class container;
00012     class resolve_context;
00013     class substitution_expression;
00014
00015     class resolve_source {
00016     public:
00017         typedef std::list<std::shared_ptr<const container>» node;
00018
00019         struct result_with_path {
00020             resolve_result<shared_value> result;
00021             node path_from_root;
00022
00023             result_with_path(resolve_result<shared_value> result_value, node path_from_root_value);
00024         };
00025
00026         resolve_source(shared_object root);
00027         resolve_source(shared_object root, node path_from_root);
00028         resolve_source push_parent(std::shared_ptr<const container> parent) const;
00029         result_with_path lookup_subst(resolve_context context,
std::shared_ptr<substitution_expression> subst, int prefix_length) const;
00030
00031         resolve_source replace_current_parent(std::shared_ptr<const container> old,
std::shared_ptr<const container> replacement) const;
00032         resolve_source replace_within_current_parent(shared_value old, shared_value replacement)
const;
00033         resolve_source reset_parents() const;
00034
00035     private:
00036         struct value_with_path {
00037             shared_value value;
00038             node path_from_root;
00039
00040             value_with_path(shared_value v, node path_from_root_value);
00041         };
00042
00043         shared_object _root;
00044         node _path_from_root;
00045
00046         static value_with_path find_in_object(shared_object obj, path the_path);
00047         static result_with_path find_in_object(shared_object obj, resolve_context context, path
the_path);
00048         static value_with_path find_in_object(shared_object obj, path the_path, node parents);
00049         static not_resolved_exception improve_not_resolved(path what, not_resolved_exception const&
original);
00050
00051         shared_object root_must_be_obj(std::shared_ptr<const container> value) const;
00052
00053         static node replace(const node& list, std::shared_ptr<const container> old, shared_value
replacement);
00054     };
00055 } // namespace hocon

```

8.53 simple_config_document.hpp

```

00001 #pragma once
00002
00003 #include <hocon/parser/config_document.hpp>
00004 #include <internal/nodes/config_node_root.hpp>
00005 #include <hocon/config_parse_options.hpp>
00006 #include <memory>
00007
00008 namespace hocon {
00009
00010     class simple_config_document : public config_document {
00011     public:
00012         simple_config_document(std::shared_ptr<const config_node_root> root,
                                config_parse_options opts);
00013
00014         std::unique_ptr<config_document> with_value_text(std::string path, std::string new_value)
const override;
00016         std::unique_ptr<config_document> with_value(std::string path,
std::shared_ptr<config_value> new_value) const
override;
00018         std::unique_ptr<config_document> without_path(std::string path) const override;
00019
00020         bool has_path(std::string const& path) const override;
00021
00022         std::string render() const override;
00023
00024     private:
00025         std::shared_ptr<const config_node_root> _config_node_tree;
00026         config_parse_options _parse_options;
00027     };
00028 }

```

8.54 simple_config_origin.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_origin.hpp>
00004
00005 #include <string>
00006 #include <vector>
00007 #include <memory>
00008
00009 namespace hocon {
00010
00011     enum class origin_type { GENERIC, FILE, RESOURCE };
00012
00013     class simple_config_origin : public config_origin, public
std::enable_shared_from_this<simple_config_origin> {
00014     public:
00015         simple_config_origin(std::string description, int line_number, int end_line_number,
00016             origin_type org_type, std::string resource_or_null, std::vector<std::string>
comments_or_null);
00017
00019         simple_config_origin(std::string description, int line_number = -1, int end_line_number = -1,
00020             origin_type org_type = origin_type::GENERIC);
00021
00022         int line_number() const override;
00023
00024         std::string const& description() const override;
00025         std::vector<std::string> const& comments() const override;
00026
00031         shared_origin with_line_number(int line_number) const override;
00032
00033         shared_origin with_comments(std::vector<std::string> comments) const override;
00034         std::shared_ptr<const simple_config_origin> append_comments(std::vector<std::string> comments)
const;
00035         std::shared_ptr<const simple_config_origin> prepend_comments(std::vector<std::string>
comments) const;
00036
00037         static shared_origin merge_origins(shared_origin a, shared_origin b);
00038         static shared_origin merge_origins(std::vector<shared_value> const& stack);
00039         static shared_origin merge_origins(std::vector<shared_origin> const& stack);
00040
00041         bool operator==(const simple_config_origin &other) const;
00042         bool operator!=(const simple_config_origin &other) const;
00043
00044     private:
00045         static std::shared_ptr<const simple_config_origin> merge_two(std::shared_ptr<const
simple_config_origin> a,
00046             std::shared_ptr<const
simple_config_origin> b);
00047
00048         // this picks the best pair to merge, because the pair has the most in
00049         // common. we want to merge two lines in the same file rather than something
00050         // else with one of the lines; because two lines in the same file can be
00051         // better consolidated.
00052         static std::shared_ptr<const simple_config_origin> merge_three(std::shared_ptr<const
simple_config_origin> a,
00053             std::shared_ptr<const
simple_config_origin> b,
00054             std::shared_ptr<const
simple_config_origin> c);
00055
00056         static int similarity(std::shared_ptr<const simple_config_origin> a,
00057             std::shared_ptr<const simple_config_origin> b);
00058
00059         std::string _description;
00060         int _line_number;
00061         int _end_line_number;
00062         origin_type _origin_type;
00063         std::string _resource_or_null;
00064         std::vector<std::string> _comments_or_null;
00065     };
00066
00067 } // namespace hocon

```

8.55 simple_include_context.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_include_context.hpp>
00004 #include "parseable.hpp"
00005
00006 namespace hocon {
00007

```

```

00008     class simple_include_context : public config_include_context {
00009     public:
00010         // Include context is part of a parseable, so it can always expect a valid parseable
reference.
00011         simple_include_context(parseable const& parseable);
00012
00013         // Unused method
00014         // shared_include_context with_parseable(weak_parseable new_parseable) const;
00015
00016         shared_parseable relative_to(std::string file_name) const override;
00017         config_parse_options parse_options() const override;
00018
00019     private:
00020         parseable const& _parseable;
00021     };
00022 } // namespace hocon

```

8.56 simple_includer.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_includer.hpp>
00004 #include <hocon/config_includer_file.hpp>
00005 #include <hocon/config_parse_options.hpp>
00006 #include <internal/full_includer.hpp>
00007
00008 namespace hocon {
00009
00010     class name_source;
00011
00012     class simple_includer : public config_includer, public config_includer_file, public
std::enable_shared_from_this<simple_includer> {
00013     public:
00014         simple_includer(shared_includer fallback);
00015
00016         shared_includer with_fallback(shared_includer fallback) const override;
00017
00018         shared_object include(shared_include_context context, std::string what) const override;
00019
00020         shared_object include_without_fallback(shared_include_context context, std::string what)
const;
00021
00022         shared_object include_file(shared_include_context context, std::string what) const override;
00023
00024         static shared_object include_file_without_fallback(shared_include_context context, std::string
what);
00025
00026         static config_parse_options clear_for_include(config_parse_options const& options);
00027
00028         static shared_object from_basename(std::shared_ptr<name_source> source,
std::string name,
00029         config_parse_options options);
00030
00031         static std::shared_ptr<const full_includer> make_full(std::shared_ptr<const config_includer>
includer);
00032
00033     private:
00034         shared_includer _fallback;
00035
00036         // the Proxy is a proxy for an application-provided includer that uses our
00037         // default implementations when the application-provided includer doesn't
00038         // have an implementation.
00039         class proxy : public full_includer, public std::enable_shared_from_this<proxy> {
00040         public:
00041             proxy(std::shared_ptr<const config_includer> delegate);
00042
00043             shared_includer with_fallback(shared_includer fallback) const;
00044
00045             shared_object include(shared_include_context context, std::string what) const;
00046
00047             shared_object include_file(shared_include_context context, std::string what) const;
00048
00049             static std::shared_ptr<const full_includer> make_full(shared_includer includer);
00050
00051         private:
00052             std::shared_ptr<const config_includer> _delegate;
00053         };
00054
00055     };
00056
00057     class name_source {
00058     public:
00059         name_source(shared_include_context context) : _context(move(context)) {
00060             if (nullptr != _context) {

```

```

00061         _context_initialized = true;
00062     } else {
00063         _context_initialized = false;
00064     }
00065 }
00066 name_source() : name_source(nullptr) {}
00067 virtual shared_parseable name_to_parseable(std::string name,
00068                                           config_parse_options parse_options) const = 0;
00069
00070 void set_context(shared_include_context context) {
00071     if (!_context_initialized) {
00072         _context_initialized = true;
00073         _context = context;
00074     }
00075 }
00076
00077 shared_include_context get_context() const {
00078     return _context;
00079 }
00080
00081 bool context_initialized() const {
00082     return _context_initialized;
00083 }
00084
00085 private:
00086     shared_include_context _context;
00087     bool _context_initialized;
00088 };
00089
00090 class relative_name_source : public name_source {
00091 public:
00092     relative_name_source(shared_include_context context);
00093
00094     shared_parseable name_to_parseable(std::string name,
00095                                       config_parse_options parse_options) const override;
00096 };
00097
00098 class file_name_source : public name_source {
00099 public:
00100     file_name_source();
00101
00102     file_name_source(shared_include_context context);
00103
00104     shared_parseable name_to_parseable(std::string name,
00105                                       config_parse_options parse_options) const override;
00106 };
00107
00108 } // namespace hocon

```

8.57 substitution_expression.hpp

```

00001 #pragma once
00002
00003 #include <string>
00004 #include <memory>
00005 #include <hocon/path.hpp>
00006
00007 namespace hocon {
00008
00009     class substitution_expression : public std::enable_shared_from_this<substitution_expression> {
00010     public:
00011         substitution_expression(path the_path, bool optional);
00012
00013         path get_path() const;
00014         bool optional() const;
00015
00016         std::shared_ptr<substitution_expression> change_path(path new_path);
00017
00018         std::string to_string() const;
00019
00020         bool operator==(substitution_expression const& other) const;
00021
00022     private:
00023         const path _path;
00024         const bool _optional;
00025     };
00026
00027 } // namespace hocon
00028

```

8.58 token.hpp

```

00001 #pragma once
00002
00003 #include "simple_config_origin.hpp"
00004
00005 #include <string>
00006
00007 namespace hocon {
00008
00009     enum class token_type {
00010         START, END, COMMA, EQUALS, COLON, OPEN_CURLY, CLOSE_CURLY, OPEN_SQUARE, CLOSE_SQUARE,
00011         VALUE, NEWLINE, UNQUOTED_TEXT, IGNORED_WHITESPACE, SUBSTITUTION, PROBLEM, COMMENT, PLUS_EQUALS
00012     };
00013
00014     struct unsupported_exception : std::runtime_error {
00015         explicit unsupported_exception(std::string const& message);
00016     };
00017
00018     class token {
00019     public:
00020         token(token_type type, shared_origin origin = nullptr,
00021             std::string token_text = "", std::string debug_string = "");
00022
00023         virtual token_type get_token_type() const;
00024         virtual std::string token_text() const;
00025         virtual std::string to_string() const;
00026         virtual shared_origin const& origin() const;
00027
00028         int line_number() const;
00029
00030         virtual bool operator==(const token& other) const;
00031
00032     private:
00033         token_type _token_type;
00034
00035         shared_origin _origin;
00036
00037         std::string _token_text;
00038         std::string _debug_string;
00039     };
00040
00041     using shared_token = std::shared_ptr<const token>;
00042     using token_list = std::vector<shared_token>;
00043
00044 } // namespace hocon
00045

```

8.59 tokenizer.hpp

```

00001 #pragma once
00002
00003 #include "tokens.hpp"
00004 #include "hocon/config_exception.hpp"
00005 #include <hocon/config_syntax.hpp>
00006
00007 #include <boost/nowide/fstream.hpp>
00008 #include <vector>
00009 #include <queue>
00010 #include <string>
00011
00012 namespace hocon {
00013
00014     // This exception should not leave this file
00015     class problem_exception : std::runtime_error {
00016     public:
00017         problem_exception(problem prob);
00018         problem const& get_problem() const;
00019
00020     private:
00021         problem _problem;
00022     };
00023
00024     class iterator {
00025     public:
00026         virtual bool has_next() = 0;
00027         virtual shared_token next() = 0;
00028     };
00029
00030     template <typename iter>
00031     class iterator_wrapper : public iterator {
00032     public:
00033         iterator_wrapper(iter begin, iter end)
00034             : _cur(begin), _end(end) { }
00035

```



```

00035
00036     bool has_next() override {
00037         return _cur != _end;
00038     }
00039
00040     shared_token next() override {
00041         return *_cur++;
00042     }
00043
00044 private:
00045     iter _cur;
00046     iter _end;
00047 };
00048
00049 class token_iterator : public iterator {
00050 public:
00051     token_iterator(shared_origin origin, std::unique_ptr<std::istream> input, bool
allow_comments);
00052     token_iterator(shared_origin origin, std::unique_ptr<std::istream> input, config_syntax
flavor);
00053
00054     bool has_next() override;
00055     shared_token next() override;
00056
00057     static std::string render(token_list tokens);
00058
00059 private:
00060     class whitespace_saver {
00061     public:
00062         whitespace_saver();
00063         void add(char c);
00064         shared_token check(token_type type, shared_origin base_origin, int line_number);
00065
00066     private:
00067         shared_token next_is_not_simple_value(shared_origin base_origin, int line_number);
00068         shared_token next_is_simple_value(shared_origin origin, int line_number);
00069         shared_token create_whitespace_token(shared_origin base_origin, int line_number);
00070
00071         std::string _whitespace;
00072         bool _last_token_was_simple_value;
00073     };
00074
00075     bool start_of_comment(char c);
00076     shared_token pull_comment(char first_char);
00077
00078     char next_char_after_whitespace(whitespace_saver& saver);
00079
00080     shared_token pull_unquoted_text();
00081
00082     shared_token pull_number(char first_char);
00083
00084     void pull_escape_sequence(std::string& parsed, std::string& original);
00085
00086     void append_triple_quoted_string(std::string& parsed, std::string& original);
00087
00088     shared_token pull_quoted_string();
00089
00090     shared_token const& pull_plus_equals();
00091     shared_token pull_substitution();
00092     shared_token pull_next_token(whitespace_saver& saver);
00093     void queue_next_token();
00094
00095     static bool is_simple_value(token_type type);
00096     static std::string as_string(char c);
00097     static shared_origin line_origin(shared_origin base_origin, int line_number);
00098
00099     shared_origin _origin;
00100     std::unique_ptr<std::istream> _input;
00101     bool _allow_comments;
00102     int _line_number;
00103     shared_origin _line_origin;
00104     std::queue<shared_token> _tokens;
00105     whitespace_saver _whitespace_saver;
00106 };
00107
00108 class single_token_iterator : public iterator {
00109 public:
00110     single_token_iterator(shared_token token);
00111
00112     bool has_next() override;
00113     shared_token next() override;
00114
00115 private:
00116     shared_token _token;
00117     bool _has_next;
00118 };
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130

```

```

00131     class token_list_iterator : public iterator {
00132     public:
00133         token_list_iterator(token_list tokens);
00134
00135         bool has_next() override;
00136         shared_token next() override;
00137
00138     private:
00139         token_list _tokens;
00140         int _index;
00141     };
00142
00143 } // namespace hocon

```

8.60 tokens.hpp

```

00001 #pragma once
00002
00003 #include "token.hpp"
00004 #include <hocon/config_value.hpp>
00005
00006 namespace hocon {
00007
00008     class value : public token {
00009     public:
00010         value(shared_value value);
00011         value(shared_value value, std::string original_text);
00012
00013         std::string to_string() const override;
00014         shared_origin const& origin() const override;
00015
00016         shared_value get_value() const;
00017
00018         bool operator==(const token& other) const override;
00019
00020     private:
00021         shared_value _value;
00022     };
00023
00024     class line : public token {
00025     public:
00026         line(shared_origin origin);
00027
00028         std::string to_string() const override;
00029
00030         bool operator==(const token& other) const override;
00031     };
00032
00033     class unquoted_text : public token {
00034     public:
00035         unquoted_text(shared_origin origin, std::string text);
00036
00037         std::string to_string() const override;
00038
00039         bool operator==(const token& other) const override;
00040     };
00041
00042     class ignored_whitespace : public token {
00043     public:
00044         ignored_whitespace(shared_origin origin, std::string whitespace);
00045
00046         std::string to_string() const override;
00047
00048         bool operator==(const token& other) const override;
00049     };
00050
00051     class problem : public token {
00052     public:
00053         problem(shared_origin origin, std::string what, std::string message, bool suggest_quotes);
00054
00055         std::string what() const;
00056         std::string message() const;
00057         bool suggest_quotes() const;
00058
00059         std::string to_string() const override;
00060
00061         bool operator==(const token& other) const override;
00062
00063     private:
00064         std::string _what;
00065         std::string _message;
00066         bool _suggest_quotes;
00067     };

```

```

00068
00069     class comment : public token {
00070     public:
00071         comment(shared_origin origin, std::string text);
00072
00073         std::string text() const;
00074
00075         std::string to_string() const override;
00076         bool operator==(const token& other) const override;
00077
00078     private:
00079         std::string _text;
00080     };
00081
00082     class double_slash_comment : public comment {
00083     public:
00084         double_slash_comment(shared_origin origin, std::string text);
00085
00086         std::string token_text() const override;
00087     };
00088
00089     class hash_comment : public comment {
00090     public:
00091         hash_comment(shared_origin origin, std::string text);
00092
00093         std::string token_text() const override;
00094     };
00095
00096     class substitution : public token {
00097     public:
00098         substitution(shared_origin origin, bool optional, token_list expression);
00099
00100         bool optional() const;
00101         token_list const& expression() const;
00102
00103         std::string token_text() const override;
00104         std::string to_string() const override;
00105
00106         bool operator==(const token& other) const override;
00107
00108     private:
00109         bool _optional;
00110         token_list _expression;
00111     };
00112
00113     class tokens {
00114     public:
00116         static shared_token const& start_token();
00117         static shared_token const& end_token();
00118         static shared_token const& comma_token();
00119         static shared_token const& equals_token();
00120         static shared_token const& colon_token();
00121         static shared_token const& open_curly_token();
00122         static shared_token const& close_curly_token();
00123         static shared_token const& open_square_token();
00124         static shared_token const& close_square_token();
00125         static shared_token const& plus_equals_token();
00126
00127         static bool is_newline(shared_token);
00128         static bool is_ignored_whitespace(shared_token);
00129
00130         static bool is_value_with_type(shared_token t, config_value::type type);
00131
00132         static shared_value get_value(shared_token t);
00133     };
00134
00135 } // namespace hocon

```

8.61 unmergeable.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_value.hpp>
00004 #include <vector>
00005
00006 namespace hocon {
00007
00014     class unmergeable {
00015     public:
00016         virtual std::vector<shared_value> unmerged_values() const = 0;
00017     };
00018
00019 } // namespace hocon
00020

```

8.62 config_boolean.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_value.hpp>
00004
00005 namespace hocon {
00006
00007     class config_boolean : public config_value {
00008     public:
00009         config_boolean(shared_origin origin, bool value);
00010
00011         config_value::type value_type() const override;
00012         std::string transform_to_string() const override;
00013
00014         unwrapped_value unwrapped() const override;
00015
00016         bool bool_value() const;
00017         bool operator==(config_value const& other) const override;
00018
00019     protected:
00020         shared_value new_copy(shared_origin) const override;
00021
00022     private:
00023         bool _value;
00024     };
00025 } // namespace hocon;
00026

```

8.63 config_concatenation.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_value.hpp>
00004 #include <internal/unmergeable.hpp>
00005 #include <internal/container.hpp>
00006 #include <string>
00007 #include <vector>
00008
00009 namespace hocon {
00010
00011     struct config_exception;
00012
00013     class config_concatenation : public config_value, public unmergeable, public container {
00014     public:
00015         config_concatenation(shared_origin origin, std::vector<shared_value> pieces);
00016
00017         config_value::type value_type() const override;
00018         std::vector<shared_value> unmerged_values() const override;
00019
00020         resolve_status get_resolve_status() const override;
00021
00022         shared_value replace_child(shared_value const& child, shared_value replacement) const
00023         override;
00024         bool has_descendant(shared_value const& descendant) const override;
00025         resolve_result<shared_value> resolve_substitutions(resolve_context const& context,
00026         resolve_source const& source) const override;
00027
00028         static std::vector<shared_value> consolidate(std::vector<shared_value> pieces);
00029         static shared_value concatenate(std::vector<shared_value> pieces);
00030         static shared_value relativized(std::string prefix) const override;
00031
00032         unwrapped_value unwrapped() const override;
00033
00034         bool operator==(config_value const& other) const override;
00035
00036     protected:
00037         shared_value new_copy(shared_origin origin) const override;
00038         bool ignores_fallbacks() const override;
00039         void render(std::string& result, int indent, bool at_root, config_render_options options)
00040         const override;
00041
00042     private:
00043         std::vector<shared_value> _pieces;
00044
00045         config_exception not_resolved() const;
00046         static bool is_ignored_whitespace(shared_value value);
00047         static void join(std::vector<shared_value> & builder, shared_value right);
00048     };
00049
00050 } // namespace hocon;
00051

```

8.64 config_delayed_merge.hpp

```

00001 #include <hocon/config_value.hpp>
00002 #include <internal/config_util.hpp>
00003 #include <internal/unmergeable.hpp>
00004 #include <internal/replaceable_merge_stack.hpp>
00005 #include <vector>
00006
00007 #pragma once
00008
00009 namespace hocon {
00010
00011     class config_delayed_merge : public config_value, public unmergeable, public
replaceable_merge_stack {
00012     public:
00013         config_delayed_merge(shared_origin origin, std::vector<shared_value> stack);
00014
00015         config_value::type value_type() const override;
00016
00017         shared_value make_replacement(resolve_context const& context, int skipping) const override;
00018
00019         // static method also used by ConfigDelayedMergeObject; end may be null
00020         static shared_value make_replacement(resolve_context const& context,
std::vector<shared_value> stack,
00021                                             int skipping);
00022
00023         std::vector<shared_value> unmerged_values() const override;
00024
00025         unwrapped_value unwrapped() const override;
00026
00027         resolve_result<shared_value> resolve_substitutions(resolve_context const& context,
resolve_source const& source) const override;
00028
00029         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);
00030         resolve_status get_resolve_status() const override { return resolve_status::UNRESOLVED; }
00031
00032         bool operator==(config_value const& other) const override;
00033
00034         shared_value replace_child(shared_value const& child, shared_value replacement) const
override;
00035         bool has_descendant(shared_value const& descendant) const override;
00036
00037         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);
00038
00039     protected:
00040         shared_value new_copy(shared_origin) const override;
00041
00042         bool ignores_fallbacks() const override;
00043
00044         virtual void render(std::string& result, int indent, bool at_root, std::string const& at_key,
config_render_options options) const override;
00045         virtual void render(std::string& result, int indent, bool at_root, config_render_options
options) const override;
00046
00047     private:
00048         std::vector<shared_value> _stack;
00049     };
00050
00051 } // namespace hocon::config_delayed_merge
00052
00053

```

8.65 config_delayed_merge_object.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_object.hpp>
00004 #include <hocon/config_exception.hpp>
00005 #include <internal/replaceable_merge_stack.hpp>
00006 #include <internal/values/config_delayed_merge.hpp>
00007
00008 namespace hocon {
00009
00010     class config_delayed_merge_object : public config_object, public unmergeable, public
replaceable_merge_stack {
00011     public:
00012         config_delayed_merge_object(shared_origin origin, std::vector<shared_value> const& stack);
00013
00014         resolve_result<shared_value> resolve_substitutions(resolve_context const& context,
resolve_source const& source) const override;
00015         std::vector<shared_value> unmerged_values() const override;

```

```

00016     shared_value make_replacement(resolve_context const& context, int skipping) const override;
00017
00018     shared_object with_value(path raw_path, shared_value value) const override;
00019     shared_object with_value(std::string key, shared_value value) const override;
00020
00021     resolve_status get_resolve_status() const override { return resolve_status::UNRESOLVED; }
00022
00023     std::vector<std::string> key_set() const override { throw not_resolved(); }
00024
00025     // map interface
00026     bool is_empty() const override { throw not_resolved(); }
00027     size_t size() const override { throw not_resolved(); }
00028     shared_value operator[](std::string const& key) const override { throw not_resolved(); }
00029     shared_value get(std::string const& key) const override { throw not_resolved(); }
00030     iterator begin() const override { throw not_resolved(); }
00031     iterator end() const override { throw not_resolved(); }
00032     unwrapped_value unwrapped() const override;
00033
00034     bool operator==(config_value const& other) const override;
00035
00036     // container interface
00037     shared_value replace_child(shared_value const& child, shared_value replacement) const
00038     override;
00039     bool has_descendant(shared_value const& descendant) const override;
00040
00041     protected:
00042     shared_value attempt_peek_with_partial_resolve(std::string const& key) const override;
00043     std::unordered_map<std::string, shared_value> const& entry_set() const override;
00044     shared_object without_path(path raw_path) const override;
00045     shared_object with_only_path(path raw_path) const override;
00046     shared_object with_only_path_or_null(path raw_path) const override;
00047     shared_object new_copy(resolve_status const& status, shared_origin origin) const override;
00048     bool ignores_fallbacks() const override;
00049     virtual void render(std::string& result, int indent, bool at_root, std::string const& at_key,
00050     config_render_options options) const override;
00051     virtual void render(std::string& result, int indent, bool at_root, config_render_options
00052     options) const override;
00053
00054     private:
00055     not_resolved_exception not_resolved() const;
00056
00057     const std::vector<shared_value> _stack;
00058 };
00059
00060 } // namespace hocon::config_delayed_merge_object
00061

```

8.66 config_double.hpp

```

00001 #pragma once
00002
00003 #include "config_number.hpp"
00004 #include <internal/simple_config_origin.hpp>
00005
00006 #include <string>
00007
00008 namespace hocon {
00009
00010     class config_double : public config_number {
00011     public:
00012         config_double(shared_origin origin, double value, std::string original_text);
00013
00014         std::string transform_to_string() const override;
00015
00016         unwrapped_value unwrapped() const override;
00017
00018         int64_t long_value() const override;
00019         double double_value() const override;
00020
00021     protected:
00022         shared_value new_copy(shared_origin) const override;
00023
00024     private:
00025         double _value;
00026     };
00027
00028 } // namespace hocon

```

8.67 config_int.hpp

```

00001 #pragma once
00002
00003 #include "config_number.hpp"
00004
00005 namespace hocon {
00006
00007     class config_int : public config_number {
00008     public:
00009         config_int(shared_origin origin, int value, std::string original_text);
00010
00011         std::string transform_to_string() const override;
00012
00013         unwrapped_value unwrapped() const override;
00014
00015         int64_t long_value() const override;
00016         double double_value() const override;
00017
00018     protected:
00019         shared_value new_copy(shared_origin) const override;
00020
00021     private:
00022         int _value;
00023     };
00024
00025 }

```

8.68 config_long.hpp

```

00001 #pragma once
00002
00003 #include "config_number.hpp"
00004 #include <internal/simple_config_origin.hpp>
00005
00006 #include <string>
00007
00008 namespace hocon {
00009
00010     class config_long : public config_number {
00011     public:
00012         config_long(shared_origin origin, int64_t value, std::string original_text);
00013
00014         std::string transform_to_string() const override;
00015
00016         unwrapped_value unwrapped() const override;
00017
00018         int64_t long_value() const override;
00019         double double_value() const override;
00020
00021     protected:
00022         shared_value new_copy(shared_origin) const override;
00023
00024     private:
00025         int64_t _value;
00026     };
00027
00028 } // namespace hocon

```

8.69 config_null.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_value.hpp>
00004 #include <internal/simple_config_origin.hpp>
00005
00006 #include <string>
00007
00008 namespace hocon {
00009
00010     class config_null : public config_value {
00011     public:
00012         config_null(shared_origin origin);
00013
00014         config_value::type value_type() const override;
00015         std::string transform_to_string() const override;
00016
00017         unwrapped_value unwrapped() const override;
00018
00019     };
00020
00021 }

```

```

00026         bool operator==(config_value const& other) const override;
00027
00028     protected:
00029         shared_value new_copy(shared_origin) const override;
00030         void render(std::string& result, int indent, bool at_root, config_render_options options)
00031             const override;
00032     };
00033 } // namespace hocon

```

8.70 config_number.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_value.hpp>
00004 #include <internal/simple_config_origin.hpp>
00005
00006 #include <string>
00007
00008 namespace hocon {
00009
00010     class config_number : public config_value {
00011     public:
00012         config_number(shared_origin origin,
00013             std::string original_text);
00014
00015         std::string transform_to_string() const override;
00016         config_value::type value_type() const override;
00017
00018         virtual int64_t long_value() const = 0;
00019         virtual double double_value() const = 0;
00020         bool is_whole() const;
00021
00022         bool operator==(const config_number &other) const;
00023         bool operator!=(const config_number &other) const;
00024         bool operator==(config_value const& other) const override;
00025
00026         int int_value_range_checked(std::string const& path) const;
00027
00028         static std::shared_ptr<config_number> new_number(
00029             shared_origin origin, int64_t value, std::string original_text);
00030
00031         static std::shared_ptr<config_number> new_number(
00032             shared_origin origin, double value, std::string original_text);
00033
00034     protected:
00035         std::string _original_text;
00036     };
00037
00038 } // namespace hocon

```

8.71 config_reference.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_value.hpp>
00004 #include <internal/unmergeable.hpp>
00005
00006 namespace hocon {
00007
00008     class substitution_expression;
00009
00010     class config_reference : public config_value, public unmergeable {
00011     public:
00012         config_reference(shared_origin origin, std::shared_ptr<substitution_expression> expr, int
00013             prefix_length = 0);
00014
00015         type value_type() const override;
00016         std::vector<shared_value> unmerged_values() const override;
00017         resolve_status get_resolve_status() const override;
00018         unwrapped_value unwrapped() const override;
00019
00020         std::shared_ptr<substitution_expression> expression() const;
00021
00022         bool operator==(config_value const& other) const override;
00023
00024     protected:
00025         shared_value new_copy(shared_origin origin) const override;
00026
00027     };
00028
00029 }

```



```

00030         resolve_result<shared_value> resolve_substitutions(resolve_context const& context,
00031         resolve_source const& source) const override;
00031         bool ignores_fallbacks() const override { return false; }
00032         void render(std::string& s, int indent, bool at_root, config_render_options options) const
00033         override;
00033     private:
00034         std::shared_ptr<substitution_expression> _expr;
00035         int _prefix_length;
00036     };
00037 };
00038 }

```

8.72 config_string.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_value.hpp>
00004 #include <internal/config_util.hpp>
00005
00006 namespace hocon {
00007
00008     enum class config_string_type { QUOTED, UNQUOTED };
00009
00010     class config_string : public config_value {
00011     public:
00012         config_string(shared_origin origin, std::string text, config_string_type quoted);
00013
00014         config_value::type value_type() const override;
00015         std::string transform_to_string() const override;
00016
00017         unwrapped_value unwrapped() const override;
00018
00019         bool was_quoted() const;
00020         bool operator==(config_value const& other) const override;
00021
00022     protected:
00023         shared_value new_copy(shared_origin) const override;
00024
00025         void render(std::string& s, int indent, bool at_root, config_render_options options) const
00026         override;
00027     private:
00028         std::string _text;
00029         config_string_type _quoted;
00030     };
00031
00032 } // namespace hocon

```

8.73 simple_config_list.hpp

```

00001 #pragma once
00002
00003 #include <hocon/config_value.hpp>
00004 #include <hocon/config_list.hpp>
00005 #include <hocon/config_render_options.hpp>
00006 #include <hocon/config_exception.hpp>
00007 #include <internal/container.hpp>
00008 #include <algorithm>
00009 #include <memory>
00010 #include <vector>
00011 #include <boost/optional.hpp>
00012
00013 namespace hocon {
00014
00015     class simple_config_list : public config_list, public container {
00016     public:
00017         simple_config_list(shared_origin origin, std::vector<shared_value> value);
00018         simple_config_list(shared_origin origin, std::vector<shared_value> value, resolve_status
00019         status);
00019
00020         config_value::type value_type() const override { return config_value::type::LIST; }
00021         resolve_status get_resolve_status() const override { return _resolved; }
00022
00023         shared_value replace_child(shared_value const& child, shared_value replacement) const
00024         override;
00024         bool has_descendant(shared_value const& descendant) const override;
00025
00026         shared_value relativized(const std::string prefix) const override;
00027

```

```

00028         bool contains(shared_value v) const { return std::find(_value.begin(), _value.end(), v) !=
_value.end(); }
00029         bool contains_all(std::vector<shared_value>) const;
00030
00031         int index_of(shared_value v) {
00032             auto pos = find(_value.begin(), _value.end(), v);
00033             if (pos == _value.end()) {
00034                 return -1;
00035             } else {
00036                 return pos - _value.begin();
00037             }
00038         }
00039
00040         // list interface
00041         bool is_empty() const override { return _value.empty(); }
00042         size_t size() const override { return _value.size(); }
00043         shared_value operator[](size_t index) const override { return _value.at(index); }
00044         shared_value get(size_t index) const override { return _value.at(index); }
00045         iterator begin() const override { return _value.begin(); }
00046         iterator end() const override { return _value.end(); }
00047
00048         std::shared_ptr<const simple_config_list> concatenate(std::shared_ptr<const
simple_config_list> other) const;
00049
00050         unwrapped_value unwrapped() const override;
00051
00052         bool operator==(config_value const& other) const override;
00053
00054         protected:
00055             resolve_result<shared_value>
00056             resolve_substitutions(resolve_context const& context, resolve_source const& source) const
override;
00057             shared_value new_copy(shared_origin origin) const override;
00058
00059             void render(std::string& result, int indent, bool at_root, config_render_options options)
const override;
00060
00061         private:
00062             static const long _serial_version_UID = 2L;
00063             const std::vector<shared_value> _value;
00064             const resolve_status _resolved;
00065
00066             std::shared_ptr<const simple_config_list>
00067             modify(no_exceptions_modifier& modifier, boost::optional<resolve_status> new_resolve_status)
const;
00068
00069             std::shared_ptr<const simple_config_list>
00070             modify_may_throw(modifier& modifier, boost::optional<resolve_status> new_resolve_status)
const;
00071
00072             struct resolve_modifier;
00073         };
00074
00075 } // namespace hocon

```

8.74 simple_config_object.hpp

```

00001 #pragma once
00002
00003 #include <internal/container.hpp>
00004 #include <hocon/config_object.hpp>
00005 #include <hocon/config_value.hpp>
00006 #include <hocon/config.hpp>
00007 #include <unordered_map>
00008
00009 namespace hocon {
00010
00011     class simple_config_object : public config_object, public container {
00012     public:
00013         simple_config_object(shared_origin origin, std::unordered_map<std::string, shared_value>
value,
00014                             resolve_status status, bool ignores_fallbacks);
00015
00016         simple_config_object(shared_origin origin, std::unordered_map<std::string, shared_value>
value);
00017
00018         shared_value attempt_peek_with_partial_resolve(std::string const& key) const override;
00019
00020         // map interface
00021         bool is_empty() const override { return _value.empty(); }
00022         size_t size() const override { return _value.size(); }
00023         shared_value operator[](std::string const& key) const override { return _value.at(key); }
00024         iterator begin() const override { return _value.begin(); }

```

```

00025     iterator end() const override { return _value.end(); }
00026     unwrapped_value unwrapped() const override;
00027
00028     shared_value get(std::string const& key) const override {
00029         if (_value.find(key) == _value.end()) {
00030             return nullptr;
00031         }
00032         return _value.at(key);
00033     }
00034
00035     std::unordered_map<std::string, shared_value> const& entry_set() const override;
00036
00037     resolve_status get_resolve_status() const override { return _resolved; }
00038     bool ignores_fallbacks() const override { return _ignores_fallbacks; }
00039     shared_value with_fallbacks_ignored() const override;
00040     shared_value merged_with_object(shared_object fallback) const override;
00041
00042     shared_object with_value(path raw_path, shared_value value) const override;
00043     shared_object with_value(std::string key, shared_value value) const override;
00044     shared_object without_path(path raw_path) const override;
00045     shared_object with_only_path(path raw_path) const override;
00046
00054     shared_object with_only_path_or_null(path raw_path) const override;
00055
00061     shared_value replace_child(shared_value const& child, shared_value replacement) const
00062     override;
00067     bool has_descendant(shared_value const& descendant) const override;
00068
00073     std::vector<std::string> key_set() const override;
00074
00079     std::vector<shared_value> value_set(std::unordered_map<std::string, shared_value> m) const;
00080
00081     bool operator==(config_value const& other) const override;
00082
00083     static std::shared_ptr<simple_config_object> empty();
00084     static std::shared_ptr<simple_config_object> empty(shared_origin origin);
00085     static std::shared_ptr<simple_config_object> empty_instance();
00086
00087     protected:
00088         resolve_result<shared_value>
00089         resolve_substitutions(resolve_context const& context, resolve_source const& source) const
00090         override;
00090         shared_value new_copy(shared_origin) const override;
00091         void render(std::string& s, int indent, bool at_root, config_render_options options) const
00092         override;
00093     private:
00094         std::unordered_map<std::string, shared_value> _value;
00095         resolve_status _resolved;
00096         bool _ignores_fallbacks;
00097
00098         shared_object new_copy(resolve_status const& new_status, shared_origin new_origin) const
00099         override;
00099         std::shared_ptr<simple_config_object> modify(no_exceptions_modifier& modifier) const;
00100         std::shared_ptr<simple_config_object> modify_may_throw(modifier& modifier) const;
00101
00102         static resolve_status resolve_status_from_value(const std::unordered_map<std::string,
00103         shared_value>& value);
00103
00104         struct resolve_modifier;
00105     };
00106
00107 } // namespace hocon

```


Index

- `_cur_dir`
 - `hocon::config_include_context`, [101](#)
 - `hocon::simple_include_context`, [284](#)
 - `_original_text`
 - `hocon::config_double`, [96](#)
 - `hocon::config_int`, [111](#)
 - `hocon::config_long`, [126](#)
 - `hocon::config_number`, [158](#)
- `append_includer`
 - `hocon::config_parse_options`, [172](#)
- `array_count`
 - `hocon::config_parser::parse_context`, [234](#)
- `at_key`
 - `hocon::config`, [38](#)
 - `hocon::config_boolean`, [51](#)
 - `hocon::config_concatenation`, [59](#)
 - `hocon::config_delayed_merge`, [67](#)
 - `hocon::config_delayed_merge_object`, [77](#)
 - `hocon::config_double`, [91](#)
 - `hocon::config_int`, [106](#)
 - `hocon::config_list`, [114](#)
 - `hocon::config_long`, [121](#)
 - `hocon::config_null`, [146](#)
 - `hocon::config_number`, [153](#)
 - `hocon::config_object`, [161](#)
 - `hocon::config_reference`, [179](#)
 - `hocon::config_string`, [192](#)
 - `hocon::config_value`, [201](#)
 - `hocon::simple_config_list`, [260](#)
 - `hocon::simple_config_object`, [270](#)
- `at_path`
 - `hocon::config`, [39](#)
 - `hocon::config_boolean`, [51](#)
 - `hocon::config_concatenation`, [59](#)
 - `hocon::config_delayed_merge`, [67](#)
 - `hocon::config_delayed_merge_object`, [77](#)
 - `hocon::config_double`, [91](#)
 - `hocon::config_int`, [106](#)
 - `hocon::config_list`, [114](#)
 - `hocon::config_long`, [121](#)
 - `hocon::config_null`, [146](#)
 - `hocon::config_number`, [153](#)
 - `hocon::config_object`, [161](#)
 - `hocon::config_reference`, [179](#)
 - `hocon::config_string`, [192](#)
 - `hocon::config_value`, [201](#)
 - `hocon::simple_config_list`, [261](#)
 - `hocon::simple_config_object`, [270](#)
- `attempt_peek_with_partial_resolve`
 - `hocon::config_delayed_merge_object`, [77](#)
 - `hocon::config_object`, [162](#)
 - `hocon::simple_config_object`, [271](#)
- `bad_path_exception`
 - `hocon::bad_path_exception`, [31](#)
- `bad_value_exception`
 - `hocon::bad_value_exception`, [32](#)
- `begin`
 - `hocon::config_delayed_merge_object`, [78](#)
 - `hocon::simple_config_list`, [261](#)
 - `hocon::simple_config_object`, [271](#)
 - `std`, [27](#)
- `bug_or_broken_exception`
 - `hocon::bug_or_broken_exception`, [33](#)
- C++ HOCON Parser, [1](#)
- `check_valid`
 - `hocon::config`, [39](#)
- `comments`
 - `hocon::config_origin`, [169](#)
 - `hocon::simple_config_origin`, [281](#)
- `concise`
 - `hocon::config_render_options`, [185](#)
- `config`
 - `hocon::config_object`, [167](#)
 - `hocon::config_value`, [205](#)
- `config_concatenation`
 - `hocon::config_value`, [205](#)
- `config_delayed_merge`
 - `hocon::config_value`, [205](#)
- `config_delayed_merge_object`
 - `hocon::config_object`, [167](#)
 - `hocon::config_value`, [205](#)
- `config_exception`
 - `hocon::config_exception`, [98](#)
 - `hocon::missing_exception`, [225](#)
 - `hocon::null_exception`, [231](#)
 - `hocon::wrong_type_exception`, [300](#)
- `config_include_context`
 - `hocon::config_include_context`, [99](#)
- `config_include_kind`
 - `hocon`, [21](#)
- `config_list`
 - `hocon::config_list`, [114](#)
- `config_object`
 - `hocon::config`, [49](#)
 - `hocon::config_value`, [205](#)
- `config_parse_options`
 - `hocon::config_parse_options`, [172](#)

- config_parseable
 - hocon::config, 49
- config_render_options
 - hocon::config_render_options, 185
- config_resolve_options
 - hocon::config_resolve_options, 189
- config_string_type
 - hocon, 21
- config_value
 - hocon::config, 49
 - hocon::config_mergeable, 129
 - hocon::config_object, 167
- construct_delayed_merge
 - hocon::config_delayed_merge_object, 78
 - hocon::config_object, 162
 - hocon::simple_config_object, 271
- contains
 - hocon::simple_config_list, 261
- context
 - hocon::resolve_result< V >, 253
- context_initialized
 - hocon::file_name_source, 210
 - hocon::name_source, 227
 - hocon::relative_name_source, 250
- CPP_HOCON_VERSION
 - version.h, 320
- CPP_HOCON_VERSION_MAJOR
 - version.h, 320
- CPP_HOCON_VERSION_MINOR
 - version.h, 320
- CPP_HOCON_VERSION_PATCH
 - version.h, 320
- CPP_HOCON_VERSION_WITH_COMMIT
 - version.h, 320
- create_origin
 - hocon::parseable_file, 238
 - hocon::parseable_not_found, 240
 - hocon::parseable_resources, 242
 - hocon::parseable_string, 244
- default_transformer
 - hocon::config_value, 205
- defaults
 - hocon::config_parse_options, 172
- description
 - hocon::config_origin, 169
 - hocon::simple_config_origin, 281
- double_value
 - hocon::config_double, 92
 - hocon::config_int, 107
 - hocon::config_long, 122
- duration
 - hocon, 19
- end
 - hocon::config_delayed_merge_object, 78
 - hocon::simple_config_list, 261
 - hocon::simple_config_object, 271
 - std, 27
- entry_set
 - hocon::config, 40
 - hocon::config_delayed_merge_object, 78
 - hocon::simple_config_object, 272
- equals
 - hocon::config_boolean, 53
 - hocon::config_concatenation, 60
 - hocon::config_delayed_merge, 68
 - hocon::config_delayed_merge_object, 78
 - hocon::config_double, 92
 - hocon::config_int, 107
 - hocon::config_list, 115
 - hocon::config_long, 122
 - hocon::config_null, 146
 - hocon::config_number, 154
 - hocon::config_object, 162
 - hocon::config_reference, 179
 - hocon::config_string, 194
 - hocon::config_value, 201
 - hocon::simple_config_list, 261
 - hocon::simple_config_object, 272
- forEach
 - List< T >, 222
- from_any_ref
 - hocon::config_value_factory, 207
- front
 - List< T >, 222
- FwdListIter
 - FwdListIter< T >, 213
- FwdListIter< T >, 213
 - FwdListIter, 213
 - List< T >, 224
 - operator!=, 214
 - operator++, 214
 - operator==, 214
 - operator*, 214
- generic_exception
 - hocon::generic_exception, 215
- get
 - hocon::config_delayed_merge_object, 78
 - hocon::simple_config_list, 262
 - hocon::simple_config_object, 272
- get_allow_missing
 - hocon::config_parse_options, 172
- get_allow_unresolved
 - hocon::config_resolve_options, 189
- get_comments
 - hocon::config_render_options, 185
- get_context
 - hocon::file_name_source, 210
 - hocon::name_source, 227
 - hocon::relative_name_source, 250
- get_cur_dir
 - hocon::config_include_context, 99
 - hocon::simple_include_context, 283
- get_duration
 - hocon::config, 40

- get_formatted
 - hocon::config_render_options, 186
- get_homogeneous_unwrapped_list
 - hocon::config, 41
- get_includer
 - hocon::config_parse_options, 173
- get_is_null
 - hocon::config, 41
- get_json
 - hocon::config_render_options, 186
- get_origin_comments
 - hocon::config_render_options, 186
- get_origin_description
 - hocon::config_parse_options, 173
- get_resolve_status
 - hocon::config_concatenation, 60
 - hocon::config_delayed_merge, 68
 - hocon::config_delayed_merge_object, 79
 - hocon::config_reference, 180
 - hocon::simple_config_list, 262
 - hocon::simple_config_object, 272
- get_syntax
 - hocon::config_parse_options, 173
- get_tokens
 - hocon::config_node_array, 131
 - hocon::config_node_comment, 132
 - hocon::config_node_complex_value, 133
 - hocon::config_node_concatenation, 134
 - hocon::config_node_field, 135
 - hocon::config_node_include, 137
 - hocon::config_node_object, 138
 - hocon::config_node_path, 139
 - hocon::config_node_root, 141
 - hocon::config_node_simple_value, 142
 - hocon::config_node_single_token, 143
- get_use_system_environment
 - hocon::config_resolve_options, 189
- getList
 - OutListIter< T >, 232
- guess_syntax
 - hocon::parseable_file, 238
- has_descendant
 - hocon::config_concatenation, 60
 - hocon::config_delayed_merge, 68
 - hocon::config_delayed_merge_object, 79
 - hocon::container, 208
 - hocon::replaceable_merge_stack, 251
 - hocon::simple_config_list, 262
 - hocon::simple_config_object, 272
- has_funky_chars
 - hocon::path, 246
- has_next
 - hocon::iterator_wrapper< iter >, 219
 - hocon::single_token_iterator, 287
 - hocon::token_iterator, 290
 - hocon::token_list_iterator, 291
- has_path
 - hocon::config, 41
 - hocon::config_document, 87
 - hocon::simple_config_document, 256
- has_path_or_null
 - hocon::config, 42
- headCount
 - List< T >, 222
- hocon, 15
 - config_include_kind, 21
 - config_string_type, 21
 - duration, 19
 - make_resolve_result, 22
 - operator==, 22
 - origin_type, 22
 - resolve_status, 22
 - shared_config, 19
 - shared_container, 19
 - shared_include_context, 19
 - shared_includer, 19
 - shared_list, 19
 - shared_node, 19
 - shared_node_array, 20
 - shared_node_concatenation, 20
 - shared_node_list, 20
 - shared_node_object, 20
 - shared_node_value, 20
 - shared_object, 20
 - shared_origin, 20
 - shared_parseable, 20
 - shared_string, 21
 - shared_token, 21
 - shared_value, 21
 - time_unit, 22
 - token_list, 21
 - token_type, 22
 - unwrapped_value, 21
- hocon/config.hpp, 301
- hocon/config_exception.hpp, 303
- hocon/config_include_context.hpp, 305
- hocon/config_includer.hpp, 305
- hocon/config_includer_file.hpp, 305
- hocon/config_list.hpp, 305
- hocon/config_mergeable.hpp, 306
- hocon/config_object.hpp, 306
- hocon/config_origin.hpp, 307
- hocon/config_parse_options.hpp, 307
- hocon/config_parseable.hpp, 308
- hocon/config_render_options.hpp, 308
- hocon/config_resolve_options.hpp, 309
- hocon/config_syntax.hpp, 309
- hocon/config_value.hpp, 309
- hocon/config_value_factory.hpp, 311
- hocon/export.h, 311
- hocon/functional_list.hpp, 312
- hocon/parser/config_document.hpp, 316
- hocon/parser/config_document_factory.hpp, 316
- hocon/parser/config_node.hpp, 316
- hocon/path.hpp, 317
- hocon/program_options.hpp, 317

- hocon/types.hpp, 318
- hocon/version.h, 319, 320
- hocon::abstract_config_node, 29
 - render, 29
- hocon::abstract_config_node_value, 30
 - render, 30
- hocon::bad_path_exception, 31
 - bad_path_exception, 31
- hocon::bad_value_exception, 32
 - bad_value_exception, 32
- hocon::bug_or_broken_exception, 33
 - bug_or_broken_exception, 33
- hocon::comment, 34
 - operator==, 34
 - to_string, 34
- hocon::config, 35
 - at_key, 38
 - at_path, 39
 - check_valid, 39
 - config_object, 49
 - config_parseable, 49
 - config_value, 49
 - entry_set, 40
 - get_duration, 40
 - get_homogeneous_unwrapped_list, 41
 - get_is_null, 41
 - has_path, 41
 - has_path_or_null, 42
 - is_empty, 42
 - is_resolved, 42
 - origin, 43
 - parse_file_any_syntax, 43
 - parse_string, 44
 - parseable, 49
 - resolve, 44, 45
 - resolve_with, 45, 46
 - root, 46
 - to_fallback_value, 46
 - with_fallback, 47
 - with_only_path, 47
 - with_value, 48
 - without_path, 48
- hocon::config_boolean, 49
 - at_key, 51
 - at_path, 51
 - equals, 53
 - new_copy, 53
 - operator==, 53
 - origin, 53
 - relativized, 53
 - render, 54
 - to_fallback_value, 54
 - transform_to_string, 55
 - type, 51
 - type_name, 55
 - unwrapped, 55
 - value_type, 55
 - value_type_name, 55
- with_fallback, 55
 - with_origin, 56
- hocon::config_concatenation, 57
 - at_key, 59
 - at_path, 59
 - equals, 60
 - get_resolve_status, 60
 - has_descendant, 60
 - ignores_fallbacks, 60
 - new_copy, 60
 - operator==, 60
 - origin, 61
 - relativized, 61
 - render, 61, 62
 - replace_child, 62
 - resolve_substitutions, 62
 - to_fallback_value, 63
 - type, 59
 - type_name, 63
 - unmerged_values, 63
 - unwrapped, 63
 - value_type, 63
 - value_type_name, 63
 - with_fallback, 64
 - with_origin, 64
- hocon::config_delayed_merge, 65
 - at_key, 67
 - at_path, 67
 - equals, 68
 - get_resolve_status, 68
 - has_descendant, 68
 - ignores_fallbacks, 68
 - make_replacement, 68
 - new_copy, 68
 - operator==, 69
 - origin, 69
 - relativized, 69
 - render, 69, 70
 - replace_child, 71
 - resolve_substitutions, 71
 - to_fallback_value, 71
 - type, 67
 - type_name, 71
 - unmerged_values, 71
 - unwrapped, 72
 - value_type, 72
 - value_type_name, 72
 - with_fallback, 72
 - with_origin, 73
- hocon::config_delayed_merge_object, 74
 - at_key, 77
 - at_path, 77
 - attempt_peek_with_partial_resolve, 77
 - begin, 78
 - construct_delayed_merge, 78
 - end, 78
 - entry_set, 78
 - equals, 78

- get, 78
- get_resolve_status, 79
- has_descendant, 79
- ignores_fallbacks, 79
- is_empty, 79
- iterator, 76
- key_set, 79
- make_replacement, 79
- new_copy, 80
- operator==, 80
- operator[], 80
- origin, 80
- relativized, 80
- render, 81
- replace_child, 82
- resolve_substitutions, 82
- size, 82
- to_config, 82
- to_fallback_value, 83
- type, 76
- type_name, 83
- unmerged_values, 83
- unwrapped, 83
- value_type, 83
- value_type_name, 84
- with_fallback, 84
- with_only_path, 85
- with_only_path_or_null, 85
- with_origin, 85
- with_value, 85
- without_path, 86
- hocon::config_document, 86
 - has_path, 87
 - render, 87
 - with_value, 87
 - with_value_text, 88
 - without_path, 88
- hocon::config_document_parser::parse_context, 233
 - parse_single_value, 233
- hocon::config_double, 89
 - _original_text, 96
 - at_key, 91
 - at_path, 91
 - double_value, 92
 - equals, 92
 - long_value, 92
 - new_copy, 92
 - operator==, 92
 - origin, 92
 - relativized, 93
 - render, 93
 - to_fallback_value, 94
 - transform_to_string, 94
 - type, 91
 - type_name, 94
 - unwrapped, 94
 - value_type, 95
 - value_type_name, 95
 - with_fallback, 95
 - with_origin, 96
- hocon::config_exception, 97
 - config_exception, 98
- hocon::config_include_context, 98
 - _cur_dir, 101
 - config_include_context, 99
 - get_cur_dir, 99
 - parse_options, 99
 - relative_to, 100
 - set_cur_dir, 100
- hocon::config_includer, 101
 - include, 101
 - with_fallback, 102
- hocon::config_includer_file, 102
 - include_file, 103
- hocon::config_int, 104
 - _original_text, 111
 - at_key, 106
 - at_path, 106
 - double_value, 107
 - equals, 107
 - long_value, 107
 - new_copy, 107
 - operator==, 107
 - origin, 107
 - relativized, 108
 - render, 108
 - to_fallback_value, 109
 - transform_to_string, 109
 - type, 106
 - type_name, 109
 - unwrapped, 109
 - value_type, 110
 - value_type_name, 110
 - with_fallback, 110
 - with_origin, 111
- hocon::config_list, 111
 - at_key, 114
 - at_path, 114
 - config_list, 114
 - equals, 115
 - iterator, 114
 - origin, 115
 - relativized, 115
 - render, 115, 116
 - to_fallback_value, 116
 - type, 114
 - type_name, 116
 - unwrapped, 117
 - value_type, 117
 - value_type_name, 117
 - with_fallback, 117
 - with_origin, 118
- hocon::config_long, 119
 - _original_text, 126
 - at_key, 121
 - at_path, 121

- double_value, 122
- equals, 122
- long_value, 122
- new_copy, 122
- operator==, 122
- origin, 122
- relativized, 123
- render, 123
- to_fallback_value, 124
- transform_to_string, 124
- type, 121
- type_name, 124
- unwrapped, 124
- value_type, 125
- value_type_name, 125
- with_fallback, 125
- with_origin, 126
- hocon::config_mergeable, 127
 - config_value, 129
 - to_fallback_value, 128
 - with_fallback, 128
- hocon::config_node, 129
 - render, 130
- hocon::config_node_array, 130
 - get_tokens, 131
 - new_node, 131
 - render, 131
- hocon::config_node_comment, 131
 - get_tokens, 132
 - render, 132
- hocon::config_node_complex_value, 132
 - get_tokens, 133
 - render, 133
- hocon::config_node_concatenation, 133
 - get_tokens, 134
 - new_node, 134
 - render, 134
- hocon::config_node_field, 135
 - get_tokens, 135
 - render, 135
- hocon::config_node_include, 136
 - get_tokens, 137
 - render, 137
- hocon::config_node_object, 137
 - get_tokens, 138
 - new_node, 138
 - render, 138
- hocon::config_node_path, 139
 - get_tokens, 139
 - render, 139
- hocon::config_node_root, 140
 - get_tokens, 141
 - new_node, 141
 - render, 141
- hocon::config_node_simple_value, 141
 - get_tokens, 142
 - render, 142
- hocon::config_node_single_token, 142
 - get_tokens, 143
 - render, 143
- hocon::config_null, 143
 - at_key, 146
 - at_path, 146
 - equals, 146
 - new_copy, 146
 - operator==, 146
 - origin, 147
 - relativized, 147
 - render, 147, 148
 - to_fallback_value, 148
 - transform_to_string, 148
 - type, 145
 - type_name, 149
 - unwrapped, 149
 - value_type, 149
 - value_type_name, 149
 - with_fallback, 149
 - with_origin, 150
- hocon::config_number, 151
 - _original_text, 158
 - at_key, 153
 - at_path, 153
 - equals, 154
 - operator==, 154
 - origin, 154
 - relativized, 154
 - render, 155
 - to_fallback_value, 155
 - transform_to_string, 156
 - type, 153
 - type_name, 156
 - value_type, 156
 - value_type_name, 156
 - with_fallback, 156
 - with_origin, 157
- hocon::config_object, 158
 - at_key, 161
 - at_path, 161
 - attempt_peek_with_partial_resolve, 162
 - config, 167
 - config_delayed_merge_object, 167
 - config_value, 167
 - construct_delayed_merge, 162
 - equals, 162
 - iterator, 161
 - key_set, 162
 - new_copy, 163
 - origin, 163
 - relativized, 163
 - render, 163, 164
 - resolve_source, 167
 - simple_config_object, 167
 - to_config, 164
 - to_fallback_value, 164
 - type, 161
 - type_name, 165

- value_type, 165
- value_type_name, 165
- with_fallback, 165
- with_only_path_or_null, 166
- with_origin, 166
- hocon::config_origin, 168
 - comments, 169
 - description, 169
 - line_number, 169
 - with_comments, 169
 - with_line_number, 170
- hocon::config_parse_options, 170
 - append_includer, 172
 - config_parse_options, 172
 - defaults, 172
 - get_allow_missing, 172
 - get_includer, 173
 - get_origin_description, 173
 - get_syntax, 173
 - prepend_includer, 173
 - set_allow_missing, 174
 - set_includer, 174
 - set_origin_description, 174
 - set_syntax, 175
- hocon::config_parseable, 175
 - options, 176
 - origin, 176
 - parse, 176
- hocon::config_parser::parse_context, 233
 - array_count, 234
- hocon::config_reference, 177
 - at_key, 179
 - at_path, 179
 - equals, 179
 - get_resolve_status, 180
 - ignores_fallbacks, 180
 - new_copy, 180
 - operator==, 180
 - origin, 180
 - relativized, 180
 - render, 181
 - resolve_substitutions, 182
 - to_fallback_value, 182
 - type, 179
 - type_name, 182
 - unmerged_values, 182
 - unwrapped, 182
 - value_type, 182
 - value_type_name, 183
 - with_fallback, 183
 - with_origin, 184
- hocon::config_render_options, 184
 - concise, 185
 - config_render_options, 185
 - get_comments, 185
 - get_formatted, 186
 - get_json, 186
 - get_origin_comments, 186
 - set_comments, 186
 - set_formatted, 187
 - set_json, 187
 - set_origin_comments, 187
- hocon::config_resolve_options, 188
 - config_resolve_options, 189
 - get_allow_unresolved, 189
 - get_use_system_environment, 189
 - set_allow_unresolved, 189
 - set_use_system_environment, 190
- hocon::config_string, 190
 - at_key, 192
 - at_path, 192
 - equals, 194
 - new_copy, 194
 - operator==, 194
 - origin, 194
 - relativized, 194
 - render, 195
 - to_fallback_value, 196
 - transform_to_string, 196
 - type, 192
 - type_name, 196
 - unwrapped, 196
 - value_type, 196
 - value_type_name, 196
 - with_fallback, 197
 - with_origin, 197
- hocon::config_value, 198
 - at_key, 201
 - at_path, 201
 - config, 205
 - config_concatenation, 205
 - config_delayed_merge, 205
 - config_delayed_merge_object, 205
 - config_object, 205
 - default_transformer, 205
 - equals, 201
 - origin, 201
 - relativized, 202
 - render, 202
 - resolve_context, 205
 - simple_config_list, 206
 - simple_config_object, 206
 - to_fallback_value, 203
 - token, 206
 - type, 200
 - type_name, 203
 - value, 206
 - value_type, 203
 - value_type_name, 203
 - with_fallback, 204
 - with_origin, 204
- hocon::config_value::modifier, 226
- hocon::config_value::no_exceptions_modifier, 227
 - modify_child_may_throw, 228
- hocon::config_value_factory, 206
 - from_any_ref, 207

- hocon::container, 207
 - has_descendant, 208
 - replace_child, 208
- hocon::default_transformer, 208
- hocon::double_slash_comment, 209
 - operator==, 209
 - to_string, 209
 - token_text, 210
- hocon::file_name_source, 210
 - context_initialized, 210
 - get_context, 210
 - name_to_parseable, 211
 - set_context, 211
- hocon::full_includer, 211
 - include, 212
 - include_file, 212
 - with_fallback, 212
- hocon::generic_exception, 215
 - generic_exception, 215
- hocon::hash_comment, 215
 - operator==, 216
 - to_string, 216
 - token_text, 216
- hocon::ignored_whitespace, 216
 - operator==, 217
 - to_string, 217
- hocon::io_exception, 217
 - io_exception, 218
- hocon::iterator, 218
- hocon::iterator_wrapper< iter >, 219
 - has_next, 219
 - iterator_wrapper, 219
 - next, 219
- hocon::line, 220
 - operator==, 220
 - to_string, 220
- hocon::missing_exception, 224
 - config_exception, 225
 - missing_exception, 225
- hocon::name_source, 226
 - context_initialized, 227
 - get_context, 227
 - name_source, 227
 - set_context, 227
- hocon::not_possible_to_resolve_exception, 228
 - not_possible_to_resolve_exception, 229
- hocon::not_resolved_exception, 229
 - not_resolved_exception, 230
- hocon::null_exception, 230
 - config_exception, 231
 - null_exception, 231
- hocon::parse_exception, 234
 - parse_exception, 235
- hocon::parseable, 235
 - options, 236
 - origin, 236
 - parse, 236
- hocon::parseable_file, 237
 - create_origin, 238
 - guess_syntax, 238
 - options, 238
 - origin, 238
 - parse, 238
 - reader, 239
- hocon::parseable_not_found, 239
 - create_origin, 240
 - options, 240
 - origin, 240
 - parse, 240
 - reader, 241
- hocon::parseable_resources, 241
 - create_origin, 242
 - options, 242
 - origin, 242
 - parse, 242
 - reader, 243
- hocon::parseable_string, 243
 - create_origin, 244
 - options, 244
 - origin, 244
 - parse, 244
 - reader, 245
- hocon::path, 245
 - has_funky_chars, 246
 - parent, 246
 - remainder, 246
 - render, 246
 - to_string, 246
- hocon::path_builder, 247
 - result, 247
- hocon::path_parser, 247
- hocon::problem, 248
 - operator==, 248
 - to_string, 248
- hocon::problem_exception, 249
- hocon::relative_name_source, 249
 - context_initialized, 250
 - get_context, 250
 - name_to_parseable, 250
 - set_context, 250
- hocon::replaceable_merge_stack, 250
 - has_descendant, 251
 - replace_child, 251
- hocon::resolve_context, 252
- hocon::resolve_result< V >, 252
 - context, 253
 - resolve_result, 253
 - value, 253
- hocon::resolve_source, 253
 - node, 254
- hocon::resolve_source::result_with_path, 254
 - path_from_root, 255
 - result, 255
- hocon::simple_config_document, 255
 - has_path, 256
 - render, 256

- with_value, 256
 - with_value_text, 257
 - without_path, 257
- hocon::simple_config_list, 258
 - at_key, 260
 - at_path, 261
 - begin, 261
 - contains, 261
 - end, 261
 - equals, 261
 - get, 262
 - get_resolve_status, 262
 - has_descendant, 262
 - index_of, 262
 - is_empty, 262
 - iterator, 260
 - new_copy, 262
 - operator==, 263
 - operator[], 263
 - origin, 263
 - relativized, 263
 - render, 263, 264
 - replace_child, 264
 - resolve_substitutions, 265
 - size, 265
 - to_fallback_value, 265
 - type, 260
 - type_name, 265
 - unwrapped, 265
 - value_type, 266
 - value_type_name, 266
 - with_fallback, 266
 - with_origin, 267
- hocon::simple_config_object, 267
 - at_key, 270
 - at_path, 270
 - attempt_peek_with_partial_resolve, 271
 - begin, 271
 - construct_delayed_merge, 271
 - end, 271
 - entry_set, 272
 - equals, 272
 - get, 272
 - get_resolve_status, 272
 - has_descendant, 272
 - ignores_fallbacks, 272
 - is_empty, 273
 - iterator, 270
 - key_set, 273
 - merged_with_object, 273
 - new_copy, 273
 - operator==, 273
 - operator[], 273
 - origin, 274
 - relativized, 274
 - render, 274, 275
 - replace_child, 275
 - resolve_substitutions, 276
 - size, 276
 - to_config, 276
 - to_fallback_value, 276
 - type, 270
 - type_name, 276
 - unwrapped, 277
 - value_set, 277
 - value_type, 277
 - value_type_name, 277
 - with_fallback, 277
 - with_fallbacks_ignored, 278
 - with_only_path, 278
 - with_only_path_or_null, 279
 - with_origin, 279
 - with_value, 279
 - without_path, 279
- hocon::simple_config_origin, 280
 - comments, 281
 - description, 281
 - line_number, 281
 - simple_config_origin, 281
 - with_comments, 282
 - with_line_number, 282
- hocon::simple_include_context, 283
 - _cur_dir, 284
 - get_cur_dir, 283
 - parse_options, 283
 - relative_to, 283
 - set_cur_dir, 284
- hocon::simple_includer, 284
 - include, 285
 - include_file, 285
 - with_fallback, 286
- hocon::single_token_iterator, 287
 - has_next, 287
 - next, 287
- hocon::substitution, 287
 - operator==, 288
 - to_string, 288
 - token_text, 288
- hocon::substitution_expression, 288
- hocon::token, 289
- hocon::token_iterator, 290
 - has_next, 290
 - next, 290
- hocon::token_list_iterator, 291
 - has_next, 291
 - next, 291
- hocon::tokens, 292
 - start_token, 292
- hocon::unmergeable, 292
- hocon::unquoted_text, 293
 - operator==, 293
 - to_string, 293
- hocon::unresolved_substitution_exception, 294
 - unresolved_substitution_exception, 295
- hocon::unsupported_exception, 295
- hocon::validation_failed_exception, 295

- problems, 296
- validation_failed_exception, 296
- hocon::validation_problem, 296
 - origin, 297
 - path, 297
 - problem, 298
 - to_string, 297
 - validation_problem, 297
- hocon::value, 298
 - operator==, 298
 - origin, 298
 - to_string, 299
- hocon::wrong_type_exception, 299
 - config_exception, 300
 - wrong_type_exception, 300
- ignores_fallbacks
 - hocon::config_concatenation, 60
 - hocon::config_delayed_merge, 68
 - hocon::config_delayed_merge_object, 79
 - hocon::config_reference, 180
 - hocon::simple_config_object, 272
- include
 - hocon::config_includer, 101
 - hocon::full_includer, 212
 - hocon::simple_includer, 285
- include_file
 - hocon::config_includer_file, 103
 - hocon::full_includer, 212
 - hocon::simple_includer, 285
- index_of
 - hocon::simple_config_list, 262
- insertedAt
 - List< T >, 222
- internal/config_document_parser.hpp, 321
- internal/config_parser.hpp, 321
- internal/config_util.hpp, 322
- internal/container.hpp, 322
- internal/default_transformer.hpp, 323
- internal/full_includer.hpp, 323
- internal/nodes/abstract_config_node.hpp, 323
- internal/nodes/abstract_config_node_value.hpp, 323
- internal/nodes/config_node_array.hpp, 324
- internal/nodes/config_node_comment.hpp, 324
- internal/nodes/config_node_complex_value.hpp, 324
- internal/nodes/config_node_concatenation.hpp, 324
- internal/nodes/config_node_field.hpp, 325
- internal/nodes/config_node_include.hpp, 325
- internal/nodes/config_node_object.hpp, 325
- internal/nodes/config_node_path.hpp, 326
- internal/nodes/config_node_root.hpp, 326
- internal/nodes/config_node_simple_value.hpp, 327
- internal/nodes/config_node_single_token.hpp, 327
- internal/parseable.hpp, 327
- internal/path_builder.hpp, 329
- internal/path_parser.hpp, 329
- internal/replaceable_merge_stack.hpp, 330
- internal/resolve_context.hpp, 330
- internal/resolve_result.hpp, 331
- internal/resolve_source.hpp, 331
- internal/simple_config_document.hpp, 332
- internal/simple_config_origin.hpp, 333
- internal/simple_include_context.hpp, 333
- internal/simple_includer.hpp, 334
- internal/substitution_expression.hpp, 335
- internal/token.hpp, 336
- internal/tokenizer.hpp, 336
- internal/tokens.hpp, 338
- internal/unmergeable.hpp, 339
- internal/values/config_boolean.hpp, 340
- internal/values/config_concatenation.hpp, 340
- internal/values/config_delayed_merge.hpp, 341
- internal/values/config_delayed_merge_object.hpp, 341
- internal/values/config_double.hpp, 342
- internal/values/config_int.hpp, 343
- internal/values/config_long.hpp, 343
- internal/values/config_null.hpp, 343
- internal/values/config_number.hpp, 344
- internal/values/config_reference.hpp, 344
- internal/values/config_string.hpp, 345
- internal/values/simple_config_list.hpp, 345
- internal/values/simple_config_object.hpp, 346
- io_exception
 - hocon::io_exception, 218
- is_empty
 - hocon::config, 42
 - hocon::config_delayed_merge_object, 79
 - hocon::simple_config_list, 262
 - hocon::simple_config_object, 273
- is_resolved
 - hocon::config, 42
- isEmpty
 - List< T >, 222
- iterator
 - hocon::config_delayed_merge_object, 76
 - hocon::config_list, 114
 - hocon::config_object, 161
 - hocon::simple_config_list, 260
 - hocon::simple_config_object, 270
- iterator_wrapper
 - hocon::iterator_wrapper< iter >, 219
- key_set
 - hocon::config_delayed_merge_object, 79
 - hocon::config_object, 162
 - hocon::simple_config_object, 273
- line_number
 - hocon::config_origin, 169
 - hocon::simple_config_origin, 281
- List
 - List< T >, 221, 222
- List< T >, 221
 - forEach, 222
 - front, 222
 - FwdListIter< T >, 224
 - headCount, 222
 - insertedAt, 221

- isEmpty, 222
- List, 221, 222
- member, 223
- popped_front, 223
- pushed_front, 223
- removed, 223
- removed1, 223
- take, 223
- long_value
 - hocon::config_double, 92
 - hocon::config_int, 107
 - hocon::config_long, 122
- make_replacement
 - hocon::config_delayed_merge, 68
 - hocon::config_delayed_merge_object, 79
- make_resolve_result
 - hocon, 22
- member
 - List< T >, 223
- merged_with_object
 - hocon::simple_config_object, 273
- missing_exception
 - hocon::missing_exception, 225
- modify_child_may_throw
 - hocon::config_value::no_exceptions_modifier, 228
- name_source
 - hocon::name_source, 227
- name_to_parseable
 - hocon::file_name_source, 211
 - hocon::relative_name_source, 250
- new_copy
 - hocon::config_boolean, 53
 - hocon::config_concatenation, 60
 - hocon::config_delayed_merge, 68
 - hocon::config_delayed_merge_object, 80
 - hocon::config_double, 92
 - hocon::config_int, 107
 - hocon::config_long, 122
 - hocon::config_null, 146
 - hocon::config_object, 163
 - hocon::config_reference, 180
 - hocon::config_string, 194
 - hocon::simple_config_list, 262
 - hocon::simple_config_object, 273
- new_node
 - hocon::config_node_array, 131
 - hocon::config_node_concatenation, 134
 - hocon::config_node_object, 138
 - hocon::config_node_root, 141
- next
 - hocon::iterator_wrapper< iter >, 219
 - hocon::single_token_iterator, 287
 - hocon::token_iterator, 290
 - hocon::token_list_iterator, 291
- node
 - hocon::resolve_source, 254
- not_possible_to_resolve_exception
 - hocon::not_possible_to_resolve_exception, 229
- not_resolved_exception
 - hocon::not_resolved_exception, 230
- null_exception
 - hocon::null_exception, 231
- operator!=
 - FwdListIter< T >, 214
- operator++
 - FwdListIter< T >, 214
 - OutListIter< T >, 232
- operator==
 - FwdListIter< T >, 214
 - hocon, 22
 - hocon::comment, 34
 - hocon::config_boolean, 53
 - hocon::config_concatenation, 60
 - hocon::config_delayed_merge, 69
 - hocon::config_delayed_merge_object, 80
 - hocon::config_double, 92
 - hocon::config_int, 107
 - hocon::config_long, 122
 - hocon::config_null, 146
 - hocon::config_number, 154
 - hocon::config_reference, 180
 - hocon::config_string, 194
 - hocon::double_slash_comment, 209
 - hocon::hash_comment, 216
 - hocon::ignored_whitespace, 217
 - hocon::line, 220
 - hocon::problem, 248
 - hocon::simple_config_list, 263
 - hocon::simple_config_object, 273
 - hocon::substitution, 288
 - hocon::unquoted_text, 293
 - hocon::value, 298
- operator[]
 - hocon::config_delayed_merge_object, 80
 - hocon::simple_config_list, 263
 - hocon::simple_config_object, 273
- operator*
 - FwdListIter< T >, 214
 - OutListIter< T >, 232
- options
 - hocon::config_parseable, 176
 - hocon::parseable, 236
 - hocon::parseable_file, 238
 - hocon::parseable_not_found, 240
 - hocon::parseable_resources, 242
 - hocon::parseable_string, 244
- origin
 - hocon::config, 43
 - hocon::config_boolean, 53
 - hocon::config_concatenation, 61
 - hocon::config_delayed_merge, 69
 - hocon::config_delayed_merge_object, 80
 - hocon::config_double, 92
 - hocon::config_int, 107
 - hocon::config_list, 115

- hocon::config_long, 122
- hocon::config_null, 147
- hocon::config_number, 154
- hocon::config_object, 163
- hocon::config_parseable, 176
- hocon::config_reference, 180
- hocon::config_string, 194
- hocon::config_value, 201
- hocon::parseable, 236
- hocon::parseable_file, 238
- hocon::parseable_not_found, 240
- hocon::parseable_resources, 242
- hocon::parseable_string, 244
- hocon::simple_config_list, 263
- hocon::simple_config_object, 274
- hocon::validation_problem, 297
- hocon::value, 298
- origin_type
 - hocon, 22
- OutListIter
 - OutListIter< T >, 232
- OutListIter< T >, 231
 - getList, 232
 - operator++, 232
 - operator*, 232
 - OutListIter, 232
- parent
 - hocon::path, 246
- parse
 - hocon::config_parseable, 176
 - hocon::parseable, 236
 - hocon::parseable_file, 238
 - hocon::parseable_not_found, 240
 - hocon::parseable_resources, 242
 - hocon::parseable_string, 244
- parse_exception
 - hocon::parse_exception, 235
- parse_file_any_syntax
 - hocon::config, 43
- parse_options
 - hocon::config_include_context, 99
 - hocon::simple_include_context, 283
- parse_single_value
 - hocon::config_document_parser::parse_context, 233
- parse_string
 - hocon::config, 44
- parseable
 - hocon::config, 49
- path
 - hocon::validation_problem, 297
- path_from_root
 - hocon::resolve_source::result_with_path, 255
- popped_front
 - List< T >, 223
- prepend_includer
 - hocon::config_parse_options, 173
- problem
 - hocon::validation_problem, 298
- problems
 - hocon::validation_failed_exception, 296
- pushed_front
 - List< T >, 223
- reader
 - hocon::parseable_file, 239
 - hocon::parseable_not_found, 241
 - hocon::parseable_resources, 243
 - hocon::parseable_string, 245
- relative_to
 - hocon::config_include_context, 100
 - hocon::simple_include_context, 283
- relativized
 - hocon::config_boolean, 53
 - hocon::config_concatenation, 61
 - hocon::config_delayed_merge, 69
 - hocon::config_delayed_merge_object, 80
 - hocon::config_double, 93
 - hocon::config_int, 108
 - hocon::config_list, 115
 - hocon::config_long, 123
 - hocon::config_null, 147
 - hocon::config_number, 154
 - hocon::config_object, 163
 - hocon::config_reference, 180
 - hocon::config_string, 194
 - hocon::config_value, 202
 - hocon::simple_config_list, 263
 - hocon::simple_config_object, 274
- remainder
 - hocon::path, 246
- removed
 - List< T >, 223
- removed1
 - List< T >, 223
- render
 - hocon::abstract_config_node, 29
 - hocon::abstract_config_node_value, 30
 - hocon::config_boolean, 54
 - hocon::config_concatenation, 61, 62
 - hocon::config_delayed_merge, 69, 70
 - hocon::config_delayed_merge_object, 81
 - hocon::config_document, 87
 - hocon::config_double, 93
 - hocon::config_int, 108
 - hocon::config_list, 115, 116
 - hocon::config_long, 123
 - hocon::config_node, 130
 - hocon::config_node_array, 131
 - hocon::config_node_comment, 132
 - hocon::config_node_complex_value, 133
 - hocon::config_node_concatenation, 134
 - hocon::config_node_field, 135
 - hocon::config_node_include, 137
 - hocon::config_node_object, 138
 - hocon::config_node_path, 139
 - hocon::config_node_root, 141

- hocon::config_node_simple_value, 142
- hocon::config_node_single_token, 143
- hocon::config_null, 147, 148
- hocon::config_number, 155
- hocon::config_object, 163, 164
- hocon::config_reference, 181
- hocon::config_string, 195
- hocon::config_value, 202
- hocon::path, 246
- hocon::simple_config_document, 256
- hocon::simple_config_list, 263, 264
- hocon::simple_config_object, 274, 275
- replace_child
 - hocon::config_concatenation, 62
 - hocon::config_delayed_merge, 71
 - hocon::config_delayed_merge_object, 82
 - hocon::container, 208
 - hocon::replaceable_merge_stack, 251
 - hocon::simple_config_list, 264
 - hocon::simple_config_object, 275
- resolve
 - hocon::config, 44, 45
- resolve_context
 - hocon::config_value, 205
- resolve_result
 - hocon::resolve_result< V >, 253
- resolve_source
 - hocon::config_object, 167
- resolve_status
 - hocon, 22
- resolve_substitutions
 - hocon::config_concatenation, 62
 - hocon::config_delayed_merge, 71
 - hocon::config_delayed_merge_object, 82
 - hocon::config_reference, 182
 - hocon::simple_config_list, 265
 - hocon::simple_config_object, 276
- resolve_with
 - hocon::config, 45, 46
- result
 - hocon::path_builder, 247
 - hocon::resolve_source::result_with_path, 255
- root
 - hocon::config, 46
- set_allow_missing
 - hocon::config_parse_options, 174
- set_allow_unresolved
 - hocon::config_resolve_options, 189
- set_comments
 - hocon::config_render_options, 186
- set_context
 - hocon::file_name_source, 211
 - hocon::name_source, 227
 - hocon::relative_name_source, 250
- set_cur_dir
 - hocon::config_include_context, 100
 - hocon::simple_include_context, 284
- set_formatted
 - hocon::config_render_options, 187
- set_includer
 - hocon::config_parse_options, 174
- set_json
 - hocon::config_render_options, 187
- set_origin_comments
 - hocon::config_render_options, 187
- set_origin_description
 - hocon::config_parse_options, 174
- set_syntax
 - hocon::config_parse_options, 175
- set_use_system_environment
 - hocon::config_resolve_options, 190
- shared_config
 - hocon, 19
- shared_container
 - hocon, 19
- shared_include_context
 - hocon, 19
- shared_includer
 - hocon, 19
- shared_list
 - hocon, 19
- shared_node
 - hocon, 19
- shared_node_array
 - hocon, 20
- shared_node_concatenation
 - hocon, 20
- shared_node_list
 - hocon, 20
- shared_node_object
 - hocon, 20
- shared_node_value
 - hocon, 20
- shared_object
 - hocon, 20
- shared_origin
 - hocon, 20
- shared_parseable
 - hocon, 20
- shared_string
 - hocon, 21
- shared_token
 - hocon, 21
- shared_value
 - hocon, 21
- simple_config_list
 - hocon::config_value, 206
- simple_config_object
 - hocon::config_object, 167
 - hocon::config_value, 206
- simple_config_origin
 - hocon::simple_config_origin, 281
- size
 - hocon::config_delayed_merge_object, 82
 - hocon::simple_config_list, 265
 - hocon::simple_config_object, 276

- start_token
 - hocon::tokens, 292
- std, 23
 - begin, 27
 - end, 27
- take
 - List< T >, 223
- time_unit
 - hocon, 22
- to_config
 - hocon::config_delayed_merge_object, 82
 - hocon::config_object, 164
 - hocon::simple_config_object, 276
- to_fallback_value
 - hocon::config, 46
 - hocon::config_boolean, 54
 - hocon::config_concatenation, 63
 - hocon::config_delayed_merge, 71
 - hocon::config_delayed_merge_object, 83
 - hocon::config_double, 94
 - hocon::config_int, 109
 - hocon::config_list, 116
 - hocon::config_long, 124
 - hocon::config_mergeable, 128
 - hocon::config_null, 148
 - hocon::config_number, 155
 - hocon::config_object, 164
 - hocon::config_reference, 182
 - hocon::config_string, 196
 - hocon::config_value, 203
 - hocon::simple_config_list, 265
 - hocon::simple_config_object, 276
- to_string
 - hocon::comment, 34
 - hocon::double_slash_comment, 209
 - hocon::hash_comment, 216
 - hocon::ignored_whitespace, 217
 - hocon::line, 220
 - hocon::path, 246
 - hocon::problem, 248
 - hocon::substitution, 288
 - hocon::unquoted_text, 293
 - hocon::validation_problem, 297
 - hocon::value, 299
- token
 - hocon::config_value, 206
- token_list
 - hocon, 21
- token_text
 - hocon::double_slash_comment, 210
 - hocon::hash_comment, 216
 - hocon::substitution, 288
- token_type
 - hocon, 22
- transform_to_string
 - hocon::config_boolean, 55
 - hocon::config_double, 94
 - hocon::config_int, 109
 - hocon::config_long, 124
 - hocon::config_null, 148
 - hocon::config_number, 155
 - hocon::config_object, 161
 - hocon::config_reference, 179
 - hocon::config_string, 192
 - hocon::config_value, 200
 - hocon::simple_config_list, 260
 - hocon::simple_config_object, 270
- type
 - hocon::config_boolean, 51
 - hocon::config_concatenation, 59
 - hocon::config_delayed_merge, 67
 - hocon::config_delayed_merge_object, 76
 - hocon::config_double, 91
 - hocon::config_int, 106
 - hocon::config_list, 114
 - hocon::config_long, 121
 - hocon::config_null, 145
 - hocon::config_number, 153
 - hocon::config_object, 161
 - hocon::config_reference, 179
 - hocon::config_string, 192
 - hocon::config_value, 200
 - hocon::simple_config_list, 260
 - hocon::simple_config_object, 270
- type_name
 - hocon::config_boolean, 55
 - hocon::config_concatenation, 63
 - hocon::config_delayed_merge, 71
 - hocon::config_delayed_merge_object, 83
 - hocon::config_double, 94
 - hocon::config_int, 109
 - hocon::config_list, 116
 - hocon::config_long, 124
 - hocon::config_null, 149
 - hocon::config_number, 156
 - hocon::config_object, 165
 - hocon::config_reference, 182
 - hocon::config_string, 196
 - hocon::config_value, 203
 - hocon::simple_config_list, 265
 - hocon::simple_config_object, 276
- unmerged_values
 - hocon::config_concatenation, 63
 - hocon::config_delayed_merge, 71
 - hocon::config_delayed_merge_object, 83
 - hocon::config_reference, 182
- unresolved_substitution_exception
 - hocon::unresolved_substitution_exception, 295
- unwrapped
 - hocon::config_boolean, 55
 - hocon::config_concatenation, 63
 - hocon::config_delayed_merge, 72
 - hocon::config_delayed_merge_object, 83
 - hocon::config_double, 94
 - hocon::config_int, 109
 - hocon::config_list, 117
 - hocon::config_long, 124
 - hocon::config_null, 149
 - hocon::config_reference, 182
 - hocon::config_string, 196
 - hocon::simple_config_list, 265

- hocon::simple_config_object, 277
- unwrapped_value
 - hocon, 21
- validation_failed_exception
 - hocon::validation_failed_exception, 296
- validation_problem
 - hocon::validation_problem, 297
- value
 - hocon::config_value, 206
 - hocon::resolve_result< V >, 253
- value_set
 - hocon::simple_config_object, 277
- value_type
 - hocon::config_boolean, 55
 - hocon::config_concatenation, 63
 - hocon::config_delayed_merge, 72
 - hocon::config_delayed_merge_object, 83
 - hocon::config_double, 95
 - hocon::config_int, 110
 - hocon::config_list, 117
 - hocon::config_long, 125
 - hocon::config_null, 149
 - hocon::config_number, 156
 - hocon::config_object, 165
 - hocon::config_reference, 182
 - hocon::config_string, 196
 - hocon::config_value, 203
 - hocon::simple_config_list, 266
 - hocon::simple_config_object, 277
- value_type_name
 - hocon::config_boolean, 55
 - hocon::config_concatenation, 63
 - hocon::config_delayed_merge, 72
 - hocon::config_delayed_merge_object, 84
 - hocon::config_double, 95
 - hocon::config_int, 110
 - hocon::config_list, 117
 - hocon::config_long, 125
 - hocon::config_null, 149
 - hocon::config_number, 156
 - hocon::config_object, 165
 - hocon::config_reference, 183
 - hocon::config_string, 196
 - hocon::config_value, 203
 - hocon::simple_config_list, 266
 - hocon::simple_config_object, 277
- version.h
 - CPP_HOCON_VERSION, 320
 - CPP_HOCON_VERSION_MAJOR, 320
 - CPP_HOCON_VERSION_MINOR, 320
 - CPP_HOCON_VERSION_PATCH, 320
 - CPP_HOCON_VERSION_WITH_COMMIT, 320
- with_comments
 - hocon::config_origin, 169
 - hocon::simple_config_origin, 282
- with_fallback
 - hocon::config, 47
- hocon::config_boolean, 55
- hocon::config_concatenation, 64
- hocon::config_delayed_merge, 72
- hocon::config_delayed_merge_object, 84
- hocon::config_double, 95
- hocon::config_includer, 102
- hocon::config_int, 110
- hocon::config_list, 117
- hocon::config_long, 125
- hocon::config_mergeable, 128
- hocon::config_null, 149
- hocon::config_number, 156
- hocon::config_object, 165
- hocon::config_reference, 183
- hocon::config_string, 197
- hocon::config_value, 204
- hocon::full_includer, 212
- hocon::simple_config_list, 266
- hocon::simple_config_object, 277
- hocon::simple_includer, 286
- with_fallbacks_ignored
 - hocon::simple_config_object, 278
- with_line_number
 - hocon::config_origin, 170
 - hocon::simple_config_origin, 282
- with_only_path
 - hocon::config, 47
 - hocon::config_delayed_merge_object, 85
 - hocon::simple_config_object, 278
- with_only_path_or_null
 - hocon::config_delayed_merge_object, 85
 - hocon::config_object, 166
 - hocon::simple_config_object, 279
- with_origin
 - hocon::config_boolean, 56
 - hocon::config_concatenation, 64
 - hocon::config_delayed_merge, 73
 - hocon::config_delayed_merge_object, 85
 - hocon::config_double, 96
 - hocon::config_int, 111
 - hocon::config_list, 118
 - hocon::config_long, 126
 - hocon::config_null, 150
 - hocon::config_number, 157
 - hocon::config_object, 166
 - hocon::config_reference, 184
 - hocon::config_string, 197
 - hocon::config_value, 204
 - hocon::simple_config_list, 267
 - hocon::simple_config_object, 279
- with_value
 - hocon::config, 48
 - hocon::config_delayed_merge_object, 85
 - hocon::config_document, 87
 - hocon::simple_config_document, 256
 - hocon::simple_config_object, 279
- with_value_text
 - hocon::config_document, 88

- hocon::simple_config_document, [257](#)
- without_path
 - hocon::config, [48](#)
 - hocon::config_delayed_merge_object, [86](#)
 - hocon::config_document, [88](#)
 - hocon::simple_config_document, [257](#)
 - hocon::simple_config_object, [279](#)
- wrong_type_exception
 - hocon::wrong_type_exception, [300](#)