cpp-hocon

0.3.0

Generated by Doxygen 1.9.1

Chapter 1

C++ HOCON Parser

This is a port of the TypesafeConfig library to C++.

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

1.1 Caveats

This is a mostly complete implementation of the HOCON format. It currently has some known limitations

- 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.

1.2 Build Requirements

- OSX or Linux
- GCC >= 4.8 or Clang >= 3.4 (with libc++)
- CMake >= 3.2.2
- Boost Libraries >= 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:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

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

Chapter 6

Namespace Documentation

6.1 hocon Namespace Reference

Factory for creating config_document instances.

Classes

- class config

  An immutable map from config paths to config values.
- struct config_exception

  All exceptions thrown by the library are subclasses of `config_exception`.
- struct wrong_type_exception

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

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

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

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

  Exception indicating that a path expression was invalid.
- struct bug_or_broken_exception

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

  Exception indicating that there was an IO error.
- struct parse_exception

  Exception indicating that there was a parse error.
- struct unresolved_substitution_exception

  Exception indicating that a substitution did not resolve to anything.
- struct not_resolved_exception

  Exception indicating that you tried to use a function that requires substitutions to be resolved, but substitutions have not been resolved (that is, config#resolve was not called).
- struct not_possible_to_resolve_exception
- struct validation_problem

Information about a problem that occurred in config#check_valid.

- struct validation_failed_exception

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

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

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

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

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

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

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

  A set of options related to parsing.
- class config_parseable

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

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

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

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

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

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

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

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

  Represents an include statement of the form "include include_kind(include_path)".
- class config_node_object
- class config_node_path
- class config_node_root

- class config_node_simple_value
- class config_node_single_token
- class parseable
- class parseable_file
- class parseable_string
- class parseable_resources
- class parseable_not_found
- class path_builder
- class path_parser
- class replaceable_merge_stack
- class resolve_context
- class resolve_source
- class simple_config_document
- class simple_config_origin
- class simple_include_context
- class simple_includer
- class name_source
- class relative_name_source
- class file_name_source
- class substitution_expression
- struct unsupported_exception
- class token
- class problem_exception
- class iterator
- class iterator_wrapper
- class token_iterator
- class single_token_iterator
- class token_list_iterator
- class value
- class line
- class unquoted_text
- class ignored_whitespace
- class problem
- class comment
- class double_slash_comment
- class hash_comment
- class substitution
- class tokens
- class unmergeable

  Interface that tags a ConfigValue that is not mergeable until after substitutions are resolved.
- class config_boolean
- class config_concatenation

  A ConfigConcatenation represents a list of values to be concatenated (see the spec).
- class config_delayed_merge
- class config_delayed_merge_object
- class config_double
- class config_int
- class config_long
- class config_null

  This exists because sometimes null is not the same as missing.
- class config_number
- class config_reference
- class config_string
- class simple_config_list
- class simple_config_object

Typedefs

- using duration = std::pair< int64_t, int >

  A duration represented as a 64-bit integer of seconds plus a 32-bit number of nanoseconds representing a fraction of a second.
- using **shared_config** = std::shared_ptr< const config >
- using **shared_object** = std::shared_ptr< const config_object >
- using **shared_origin** = std::shared_ptr< const config_origin >
- using **shared_value** = std::shared_ptr< const config_value >
- using **shared_list** = std::shared_ptr< const config_list >
- typedef boost::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 = typedef std::pair<int64_t, int>
```

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

Definition at line 21 of file types.hpp.

6.1.3 Function Documentation

6.1.3.1 operator==()

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

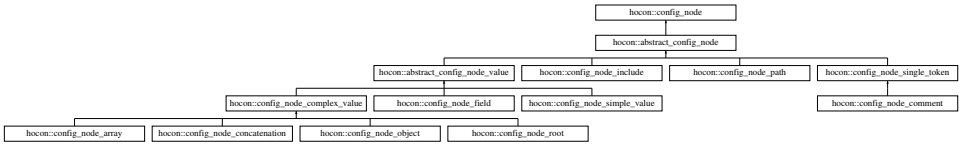
Config documents compare via rendered strings.

Chapter 7

Class Documentation

7.1 hocon::abstract_config_node Class Reference

Inheritance diagram for hocon::abstract_config_node:



Public Member Functions

- std::string render () const

  The original text of the input which was used to form this particular node.
- bool **operator==** (const abstract_config_node &other) const
- virtual token_list **get_tokens** () const =0

7.1.1 Detailed Description

Definition at line 9 of file abstract_config_node.hpp.

7.1.2 Member Function Documentation

7.1.2.1 render()

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

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

Returns

the original text used to form this node as a String

Implements hocon::config_node.

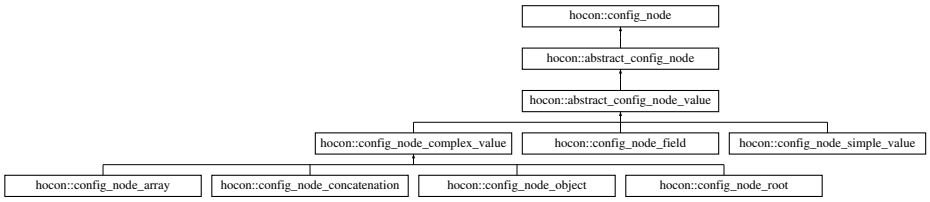
The documentation for this class was generated from the following file:

- internal/nodes/abstract_config_node.hpp

7.2 hocon::abstract_config_node_value Class Reference

This is used to classify certain abstract_config_node subclasses.

Inheritance diagram for hocon::abstract_config_node_value:



Public Member Functions

- std::string render () const

  The original text of the input which was used to form this particular node.
- bool **operator==** (const abstract_config_node &other) const
- virtual token_list **get_tokens** () const =0

7.2.1 Detailed Description

This is used to classify certain abstract_config_node subclasses.

Definition at line 8 of file abstract_config_node_value.hpp.

7.2.2 Member Function Documentation

7.2.2.1 render()

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

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

Returns

the original text used to form this node as a String

Implements hocon::config_node.

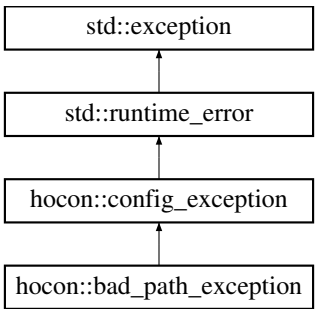
The documentation for this class was generated from the following file:

- internal/nodes/abstract_config_node_value.hpp

7.3 hocon::bad_path_exception Struct Reference

Exception indicating that a path expression was invalid.

Inheritance diagram for hocon::bad_path_exception:



Public Member Functions

- **bad_path_exception** (config_origin const &origin, std::string const &path, std::string const &message)
- **bad_path_exception** (std::string const &path, std::string const &message)

7.3.1 Detailed Description

Exception indicating that a path expression was invalid.

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

Definition at line 71 of file config_exception.hpp.

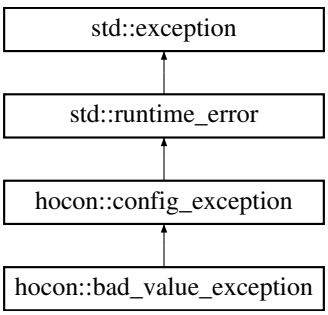
The documentation for this struct was generated from the following file:

- hocon/config_exception.hpp

7.4 hocon::bad_value_exception Struct Reference

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

Inheritance diagram for hocon::bad_value_exception:



Public Member Functions

- **bad_value_exception** (config_origin const &origin, std::string const &path, std::string const &message)
- **bad_value_exception** (std::string const &path, std::string const &message)

7.4.1 Detailed Description

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

Definition at line 59 of file config_exception.hpp.

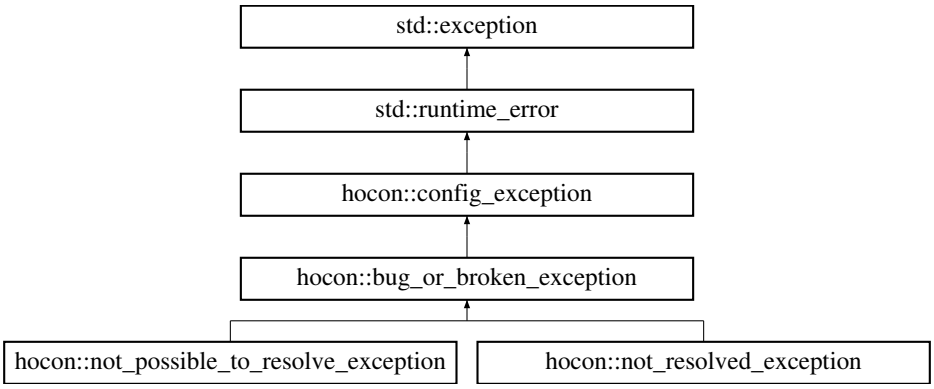
The documentation for this struct was generated from the following file:

- hocon/config_exception.hpp

7.5 hocon::bug_or_broken_exception Struct Reference

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

Inheritance diagram for hocon::bug_or_broken_exception:



Public Member Functions

- **bug_or_broken_exception** (std::string const &message)

7.5.1 Detailed Description

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

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

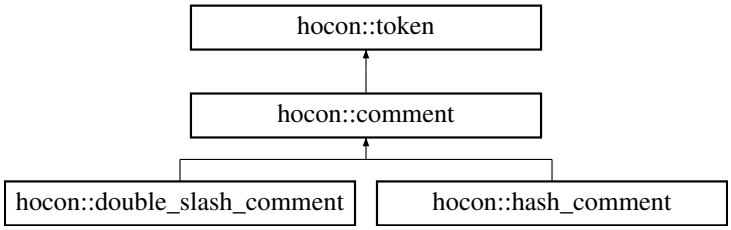
Definition at line 85 of file config_exception.hpp.

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

- hocon/config_exception.hpp

7.6 hocon::comment Class Reference

Inheritance diagram for hocon::comment:



Public Member Functions

- **comment** (shared_origin origin, std::string text)
- std::string **text** () const
- std::string **to_string** () const override
- bool **operator==** (const token &other) const override
- virtual token_type **get_token_type** () const
- virtual std::string **token_text** () const
- virtual shared_origin const & **origin** () const
- int **line_number** () const

7.6.1 Detailed Description

Definition at line 69 of file tokens.hpp.

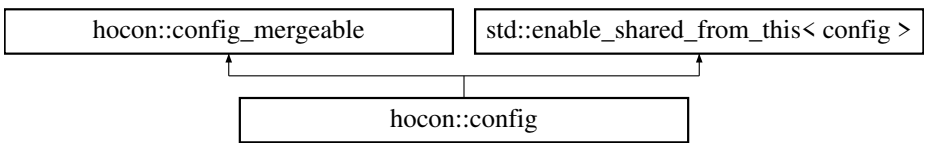
The documentation for this class was generated from the following file:

- internal/tokens.hpp

7.7 hocon::config Class Reference

An immutable map from config paths to config values.

Inheritance diagram for hocon::config:



Public Member Functions

- virtual shared_object root () const

  Gets the.
- virtual shared_origin origin () const

  Gets the origin of the.
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

  Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.
- virtual shared_config resolve () const

  Returns a replacement config with all substitutions (the `${foo.bar}` syntax, see the spec) resolved.
- virtual shared_config resolve (config_resolve_options options) const

  Like config#resolve() but allows you to specify non-default options.
- virtual bool is_resolved () const

  Checks whether the config is completely resolved.
- virtual shared_config resolve_with (shared_config source) const

  Like config#resolve() except that substitution values are looked up in the given source, rather than in this instance.
- virtual shared_config resolve_with (shared_config source, config_resolve_options options) const

  Like config#resolve_with(config) but allows you to specify non-default options.
- virtual void check_valid (shared_config reference, std::vector< std::string > restrict_to_paths) const

  Validates this config against a reference config, throwing an exception if it is invalid.
- virtual bool has_path (std::string const &path) const

  Checks whether a value is present and non-null at the given path.
- virtual bool has_path_or_null (std::string const &path) const

  Checks whether a value is present at the given path, even if the value is null.
- virtual bool is_empty () const

  Returns true if the.
- virtual std::set< std::pair< std::string, std::shared_ptr< const config_value > > > entry_set () const

  Returns the set of path-value pairs, excluding any null values, found by recursing the root object.
- virtual bool get_is_null (std::string const &path) const

  Checks whether a value is set to null at the given path, but throws an exception if the value is entirely unset.
- virtual bool **get_bool** (std::string const &path) const
- virtual int **get_int** (std::string const &path) const
- virtual int64_t **get_long** (std::string const &path) const
- virtual double **get_double** (std::string const &path) const
- virtual std::string **get_string** (std::string const &path) const
- virtual std::shared_ptr< const config_object > **get_object** (std::string const &path) const
- virtual shared_config **get_config** (std::string const &path) const
- virtual unwrapped_value **get_any_ref** (std::string const &path) const
- virtual std::shared_ptr< const config_value > **get_value** (std::string const &path) const
- template<typename T >
  std::vector< T > **get_homogeneous_unwrapped_list** (std::string const &path) const
- virtual shared_list **get_list** (std::string const &path) const
- virtual std::vector< bool > **get_bool_list** (std::string const &path) const
- virtual std::vector< int > **get_int_list** (std::string const &path) const
- virtual std::vector< int64_t > **get_long_list** (std::string const &path) const
- virtual std::vector< double > **get_double_list** (std::string const &path) const
- virtual std::vector< std::string > **get_string_list** (std::string const &path) const
- virtual std::vector< shared_object > **get_object_list** (std::string const &path) const
- virtual std::vector< shared_config > **get_config_list** (std::string const &path) const

- virtual int64_t get_duration (std::string const &path, time_unit unit) const

  Gets a value as an integer number of the specified units.
- virtual shared_config with_only_path (std::string const &path) const

  Clone the config with only the given path (and its children) retained; all sibling paths are removed.
- virtual shared_config without_path (std::string const &path) const

  Clone the config with the given path removed.
- virtual shared_config at_path (std::string const &path) const

  Places the config inside another.
- virtual shared_config at_key (std::string const &key) const

  Places the config inside a.
- virtual shared_config with_value (std::string const &path, std::shared_ptr< const config_value > value) const

  Returns a.
- bool **operator==** (config const &other) const
- **config** (shared_object object)
- template<> std::vector< int64_t > **get_homogeneous_unwrapped_list** (std::string const &path) const

Static Public Member Functions

- static shared_config parse_file_any_syntax (std::string file_basename, config_parse_options options)

  Parses a file with a flexible extension.
- static shared_config parse_file_any_syntax (std::string file_basename)

  Like parseFileAnySyntax(File,ConfigParseOptions) but always uses default parse options.
- static shared_config parse_string (std::string s, config_parse_options options)

  Parses a string (which should be valid HOCON or JSON by default, or the syntax specified in the options otherwise).
- static shared_config parse_string (std::string s)

  Parses a string (which should be valid HOCON or JSON).
- static shared_object **env_variables_as_config_object** ()

Protected Member Functions

- shared_value **find** (std::string const &path_expression, config_value::type expected) const
- shared_value **find** (path path_expression, config_value::type expected, path original_path) const
- shared_value **find** (path path_expression, config_value::type expected) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const

Static Protected Member Functions

- static shared_includer **default_includer** ()

Friends

- class **config_object**
- class **config_value**
- class **config_parseable**
- class **parseable**

7.7.1 Detailed Description

An immutable map from config paths to config values.

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

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

Fundamental operations on a
`config`

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

All operations return a new immutable
`config`

rather than modifying the original instance.

Examples

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

Paths, keys, and config vs. config_object

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

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

The API for a
`config`

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

is in terms of keys. Conceptually,
`config`

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

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

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

Another difference between
`config`

and
`config_object`

is that conceptually,
config_value

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

, while a
config

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

Getting configuration values

The "getters" on a
config

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

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

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

Iteration

If you want to iterate over the contents of a
config

, you can get its
config_object

with root(), and then iterate over the
config_object

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

Resolving substitutions

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

Before using a
config

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

Merging

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

Serialization

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

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

This is an interface but don't implement it yourself

Do not implement
`config`

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

Definition at line 172 of file config.hpp.

7.7.2 Member Function Documentation

7.7.2.1 at_key()

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

Places the config inside a.
`config`

at the given key. See also at_path(). Note that a key is NOT a path expression (see config_util#join_path and config_util#split_path).

Parameters

key	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

path	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

reference	a reference configuration
restrictToPaths	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 +/-2^63 seconds.

Parameters

path	the path to the time value
unit	the units of the number returned

Returns

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

7.7.2.6 get_is_null()

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

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

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

Parameters

path	the path expression

Returns

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

7.7.2.7 has_path()

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

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

This differs in two ways from
`Map.containsKey()`

as implemented by config_object: it looks for a path expression, not a key; and it returns false for null values, while
`contains_key()`

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

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

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

Parameters

path	the path expression

Returns

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

7.7.2.8 has_path_or_null()

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

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

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

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

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

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

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

Parameters

path	the path expression

Returns

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

7.7.2.9 is_empty()

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

Returns true if the.
Config

's root object contains no key-value pairs.

Returns

true if the configuration is empty

7.7.2.10 is_resolved()

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

Checks whether the config is completely resolved.

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

Returns

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

7.7.2.11 origin()

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

Gets the origin of the.
Config

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

Returns

the origin of the
Config

for use in error messages

7.7.2.12 parse_file_any_syntax() [1/2]

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

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

Parameters

fileBasename	a filename with or without extension

Returns

the parsed configuration

7.7.2.13 parse_file_any_syntax() [2/2]

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

Parses a file with a flexible extension.

If the `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

fileBasename	a filename with or without extension
options	parse options

Returns

the parsed configuration

7.7.2.14 parse_string() [1/2]

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

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

Parameters

s	string to parse

Returns

the parsed configuration

7.7.2.15 parse_string() [2/2]

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

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

Parameters

s	string to parse
options	parse options

Returns

the parsed configuration

7.7.2.16 resolve() [1/2]

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

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

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

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

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

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

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

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

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

Returns

an immutable object with substitutions resolved

7.7.2.17 resolve() [2/2]

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

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

Parameters

options	resolve options

Returns

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

7.7.2.18 resolve_with() [1/2]

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

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

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

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

Parameters

source	configuration to pull values from

Returns

an immutable object with substitutions resolved

7.7.2.19 resolve_with() [2/2]

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

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

Parameters

source	source configuration to pull values from
options	resolve options

Returns

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

7.7.2.20 root()

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

Gets the.
`Config`

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

).

Returns

the root object in the configuration

7.7.2.21 to_fallback_value()

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

Converts a config to its root object and a config_value to itself.

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

Implements hocon::config_mergeable.

7.7.2.22 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implements hocon::config_mergeable.

7.7.2.23 with_only_path()

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

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

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

Parameters

path	path to keep

Returns

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

7.7.2.24 with_value()

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

Returns a.
config

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

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

Parameters

path	path expression for the value's new location
value	value at the new path

Returns

the new instance with the new map entry

7.7.2.25 without_path()

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

Clone the config with the given path removed.

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

Parameters

path	path expression to remove

Returns

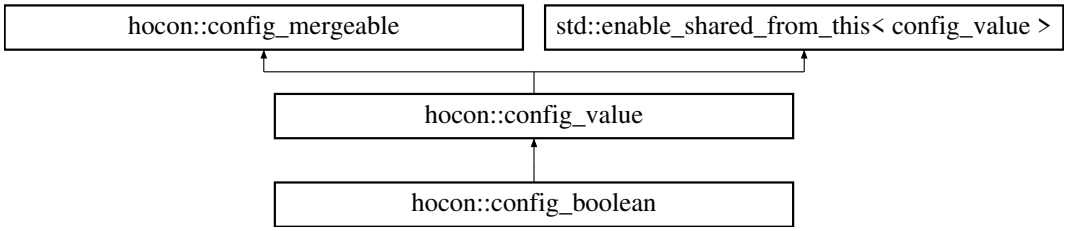
a copy of the config minus the specified path

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

- hocon/config.hpp

7.8 hocon::config_boolean Class Reference

Inheritance diagram for hocon::config_boolean:



Public Types

- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

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

Public Member Functions

- **config_boolean** (shared_origin origin, bool value)
- config_value::type value_type () const override

  The type of the value; matches the JSON type schema.
- std::string **transform_to_string** () const override
- unwrapped_value **unwrapped** () const override
- bool **bool_value** () const
- bool **operator==** (config_value const &other) const override
- virtual shared_origin const & origin () const

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

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

  Renders the config value as a HOCON string.
- virtual std::string render (config_render_options options) const

  Renders the config value to a string, using the provided options.
- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.
- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.
- virtual shared_value with_origin (shared_origin origin) const

  Returns a.
- virtual shared_value relativized (std::string prefix) const

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- virtual resolve_status **get_resolve_status** () const
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

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

Static Public Member Functions

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

Protected Member Functions

- shared_value **new_copy** (shared_origin) const override
- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const
- virtual void **render** (std::string &result, int indent, bool at_root, config_render_options options) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- virtual resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const
- void **require_not_ignoring_fallbacks** () const
- virtual bool **ignores_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const

- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- virtual shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

7.8.1 Detailed Description

Definition at line 7 of file config_boolean.hpp.

7.8.2 Member Enumeration Documentation

7.8.2.1 type

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

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

Definition at line 56 of file config_value.hpp.

7.8.3 Member Function Documentation

7.8.3.1 at_key()

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

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	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

path	path to store this value at.

Returns

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

7.8.3.3 origin()

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

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

Returns

where the value came from

7.8.3.4 relativized()

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

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

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

Parameters

prefix	

Returns

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

Reimplemented in hocon::config_concatenation, and hocon::simple_config_list.

Definition at line 181 of file config_value.hpp.

7.8.3.5 render() [1/2]

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

Renders the config value as a HOCON string.

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

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

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

.

Returns

the rendered value

7.8.3.6 render() [2/2]

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

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

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

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

Parameters

options	the rendering options

Returns

the rendered value

7.8.3.7 to_fallback_value()

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

Converts a config to its root object and a config_value to itself.

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

Implements hocon::config_mergeable.

7.8.3.8 value_type()

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

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

Returns

value's type

Implements hocon::config_value.

7.8.3.9 value_type_name()

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

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.8.3.10 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implements hocon::config_mergeable.

7.8.3.11 with_origin()

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

Returns a.
`config_value`

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

Parameters

origin	the origin set on the returned value

Returns

the new config_value with the given origin

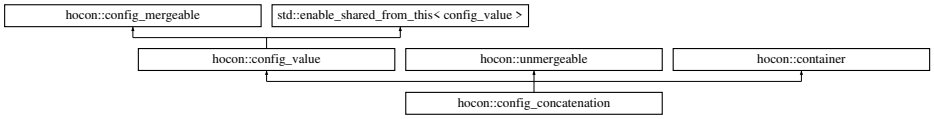
The documentation for this class was generated from the following file:

- internal/values/config_boolean.hpp

7.9 hocon::config_concatenation Class Reference

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

Inheritance diagram for hocon::config_concatenation:



Public Types

- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

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

Public Member Functions

- **config_concatenation** (shared_origin origin, std::vector< shared_value > pieces)
- config_value::type value_type () const override

  The type of the value; matches the JSON type schema.
- std::vector< shared_value > **unmerged_values** () const override
- resolve_status **get_resolve_status** () const override
- shared_value replace_child (shared_value const &child, shared_value replacement) const override

  Replace a child of this value.
- bool has_descendant (shared_value const &descendant) const override

  Super-expensive full traversal to see if descendant is anywhere underneath this container.
- resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const override

- shared_value relativized (std::string prefix) const override

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- unwrapped_value **unwrapped** () const override
- bool **operator==** (config_value const &other) const override
- virtual shared_origin const & origin () const

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

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

  Renders the config value as a HOCON string.
- virtual std::string render (config_render_options options) const

  Renders the config value to a string, using the provided options.
- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.
- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.
- virtual shared_value with_origin (shared_origin origin) const

  Returns a.
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

  Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.
- virtual std::string **transform_to_string** () const

Static Public Member Functions

- static std::vector< shared_value > **consolidate** (std::vector< shared_value > pieces)
- static shared_value **concatenate** (std::vector< shared_value > pieces)
- static char const * **type_name** (type t)

Protected Member Functions

- shared_value **new_copy** (shared_origin origin) const override
- bool **ignores_fallbacks** () const override
- void **render** (std::string &result, int indent, bool at_root, config_render_options options) const override
- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- void **require_not_ignoring_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- virtual shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

7.9.1 Detailed Description

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

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

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

Definition at line 24 of file config_concatenation.hpp.

7.9.2 Member Enumeration Documentation

7.9.2.1 type

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

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

Definition at line 56 of file config_value.hpp.

7.9.3 Member Function Documentation

7.9.3.1 at_key()

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

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	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

path	path to store this value at.

Returns

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

7.9.3.3 has_descendant()

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

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

Implements hocon::container.

7.9.3.4 origin()

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

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

Returns

where the value came from

7.9.3.5 relativized()

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

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

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

Parameters

prefix	

Returns

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

Reimplemented from hocon::config_value.

7.9.3.6 render() [1/2]

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

Renders the config value as a HOCON string.

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

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

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

.

Returns

the rendered value

7.9.3.7 render() [2/2]

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

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

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

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

Parameters

options	the rendering options

Returns

the rendered value

7.9.3.8 replace_child()

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

Replace a child of this value.

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

Implements hocon::container.

7.9.3.9 to_fallback_value()

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

Converts a config to its root object and a config_value to itself.

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

Implements hocon::config_mergeable.

7.9.3.10 value_type()

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

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

Returns

value's type

Implements hocon::config_value.

7.9.3.11 value_type_name()

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

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.9.3.12 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implements hocon::config_mergeable.

7.9.3.13 with_origin()

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

Returns a.
config_value

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

Parameters

origin	the origin set on the returned value

Returns

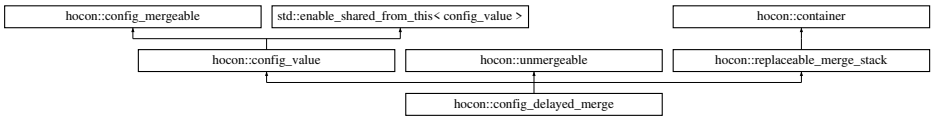
the new config_value with the given origin

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

- internal/values/config_concatenation.hpp

7.10 hocon::config_delayed_merge Class Reference

Inheritance diagram for hocon::config_delayed_merge:



Public Types

- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

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

Public Member Functions

- **config_delayed_merge** (shared_origin origin, std::vector< shared_value > stack)
- config_value::type value_type () const override

  The type of the value; matches the JSON type schema.
- shared_value **make_replacement** (resolve_context const &context, int skipping) const override
- std::vector< shared_value > **unmerged_values** () const override
- unwrapped_value **unwrapped** () const override
- resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const override
- resolve_status **get_resolve_status** () const override
- bool **operator==** (config_value const &other) const override
- shared_value replace_child (shared_value const &child, shared_value replacement) const override

  Replace a child of this value.
- bool has_descendant (shared_value const &descendant) const override

  Super-expensive full traversal to see if descendant is anywhere underneath this container.
- virtual shared_origin const & origin () const

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

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

  Renders the config value as a HOCON string.
- virtual std::string render (config_render_options options) const

  Renders the config value to a string, using the provided options.
- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.
- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.
- virtual shared_value with_origin (shared_origin origin) const

  Returns a.
- virtual shared_value relativized (std::string prefix) const

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

  Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.
- virtual std::string **transform_to_string** () const

Static Public Member Functions

- static shared_value **make_replacement** (resolve_context const &context, std::vector< shared_value > stack, int skipping)
- static resolve_result< shared_value > **resolve_substitutions** (std::shared_ptr< const replaceable_merge_stack > replaceable, const std::vector< shared_value > &_stack, resolve_context const &context, resolve_source const &source)
- static void **render** (std::vector< shared_value > const &stack, std::string &s, int indent_value, bool at_root, std::string const &at_key, config_render_options options)
- static char const * **type_name** (type t)

Protected Member Functions

- shared_value **new_copy** (shared_origin) const override
- bool **ignores_fallbacks** () const override
- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const override
- virtual void **render** (std::string &result, int indent, bool at_root, config_render_options options) const override
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- void **require_not_ignoring_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- virtual shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

7.10.1 Detailed Description

Definition at line 11 of file config_delayed_merge.hpp.

7.10.2 Member Enumeration Documentation

7.10.2.1 type

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

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

Definition at line 56 of file config_value.hpp.

7.10.3 Member Function Documentation

7.10.3.1 at_key()

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

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	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

path	path to store this value at.

Returns

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

7.10.3.3 has_descendant()

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

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

Implements hocon::container.

7.10.3.4 origin()

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

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

Returns

where the value came from

7.10.3.5 relativized()

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

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

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

Parameters

prefix	

Returns

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

Reimplemented in hocon::config_concatenation, and hocon::simple_config_list.

Definition at line 181 of file config_value.hpp.

7.10.3.6 render() [1/2]

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

Renders the config value as a HOCON string.

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

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

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

.

Returns

the rendered value

7.10.3.7 render() [2/2]

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

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

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

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

Parameters

options	the rendering options

Returns

the rendered value

7.10.3.8 replace_child()

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

Replace a child of this value.

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

Implements hocon::container.

7.10.3.9 to_fallback_value()

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

Converts a config to its root object and a config_value to itself.

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

Implements hocon::config_mergeable.

7.10.3.10 value_type()

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

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

Returns

value's type

Implements hocon::config_value.

7.10.3.11 value_type_name()

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

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.10.3.12 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implements hocon::config_mergeable.

7.10.3.13 with_origin()

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

Returns a.
`config_value`

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

Parameters

origin	the origin set on the returned value

Returns

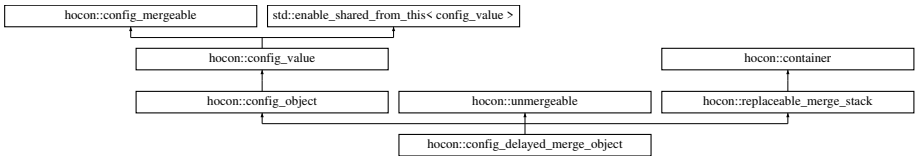
the new config_value with the given origin

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

- internal/values/config_delayed_merge.hpp

7.11 hocon::config_delayed_merge_object Class Reference

Inheritance diagram for hocon::config_delayed_merge_object:



Public Types

- using **iterator** = std::unordered_map< std::string, shared_value >::const_iterator
- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

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

Public Member Functions

- **config_delayed_merge_object** (shared_origin origin, std::vector< shared_value > const &stack)
- resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const override
- std::vector< shared_value > **unmerged_values** () const override
- shared_value **make_replacement** (resolve_context const &context, int skipping) const override
- shared_object **with_value** (path raw_path, shared_value value) const override
- shared_object **with_value** (std::string key, shared_value value) const override
- resolve_status **get_resolve_status** () const override
- std::vector< std::string > key_set () const override

  Construct a list of keys in the _value map.
- bool **is_empty** () const override

- size_t **size** () const override
- shared_value **operator[ ]** (std::string const &key) const override
- shared_value **get** (std::string const &key) const override
- iterator **begin** () const override
- iterator **end** () const override
- unwrapped_value **unwrapped** () const override
- bool **operator==** (config_value const &other) const override
- shared_value replace_child (shared_value const &child, shared_value replacement) const override

  Replace a child of this value.
- bool has_descendant (shared_value const &descendant) const override

  Super-expensive full traversal to see if descendant is anywhere underneath this container.
- virtual std::shared_ptr< const config > to_config () const

  Converts this object to a Config instance, enabling you to use path expressions to find values in the object.
- config_value::type value_type () const override

  The type of the value; matches the JSON type schema.
- virtual shared_origin const & origin () const

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

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

  Renders the config value as a HOCON string.
- virtual std::string render (config_render_options options) const

  Renders the config value to a string, using the provided options.
- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.
- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.
- virtual shared_value with_origin (shared_origin origin) const

  Returns a.
- virtual shared_value relativized (std::string prefix) const

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

  Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.
- virtual std::string **transform_to_string** () const

Static Public Member Functions

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

Protected Member Functions

- shared_value attempt_peek_with_partial_resolve (std::string const &key) const override

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

- shared_object **new_copy** (resolve_status const &status, shared_origin origin) const override
- bool **ignores_fallbacks** () const override
- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const override
- virtual void **render** (std::string &result, int indent, bool at_root, config_render_options options) const override
- shared_value **peek_path** (path desired_path) const
- shared_value **peek_assuming_resolved** (std::string const &key, path original_path) const
- shared_value **new_copy** (shared_origin origin) const override
- shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const override
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- void **require_not_ignoring_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static shared_value **peek_path** (const config_object *self, path desired_path)
- static shared_origin **merge_origins** (std::vector< shared_value > const &stack)
- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

7.11.1 Detailed Description

Definition at line 10 of file config_delayed_merge_object.hpp.

7.11.2 Member Enumeration Documentation

7.11.2.1 type

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

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

Definition at line 56 of file config_value.hpp.

7.11.3 Member Function Documentation

7.11.3.1 at_key()

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

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	key to store this value at.

Returns

a
`config`
instance containing this value at the given key.

7.11.3.2 at_path()

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

Places the value inside a config at the given path.

See also config_value#at_key(String).

Parameters

path	path to store this value at.

Returns

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

7.11.3.3 attempt_peek_with_partial_resolve()

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

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

Parameters

key	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.

7.11.3.4 has_descendant()

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

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

Implements hocon::container.

7.11.3.5 key_set()

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

Construct a list of keys in the _value map.

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

Implements hocon::config_object.

Definition at line 23 of file config_delayed_merge_object.hpp.

7.11.3.6 origin()

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

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

Returns

where the value came from

7.11.3.7 relativized()

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

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

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

Parameters

prefix	

Returns

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

Reimplemented in hocon::config_concatenation, and hocon::simple_config_list.

Definition at line 181 of file config_value.hpp.

7.11.3.8 render() [1/2]

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

Renders the config value as a HOCON string.

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

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

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

.

Returns

the rendered value

7.11.3.9 render() [2/2]

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

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

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

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

Parameters

options	the rendering options

Returns

the rendered value

7.11.3.10 replace_child()

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

Replace a child of this value.

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

Implements hocon::container.

7.11.3.11 to_config()

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

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

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

Returns

a Config with this object as its root

7.11.3.12 to_fallback_value()

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

Converts a config to its root object and a config_value to itself.

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

Implements hocon::config_mergeable.

7.11.3.13 value_type()

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

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

Returns

value's type

Implements hocon::config_value.

7.11.3.14 value_type_name()

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

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.11.3.15 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implements hocon::config_mergeable.

7.11.3.16 with_origin()

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

Returns a.
`config_value`

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

Parameters

origin	the origin set on the returned value

Returns

the new config_value with the given origin

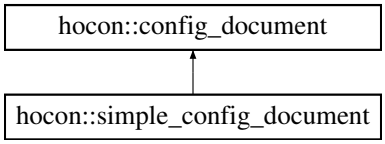
The documentation for this class was generated from the following file:

- internal/values/config_delayed_merge_object.hpp

7.12 hocon::config_document Class Reference

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

Inheritance diagram for hocon::config_document:



Public Member Functions

- virtual std::unique_ptr< config_document > with_value_text (std::string path, std::string newValue) const =0

  Returns a new config_document that is a copy of the current config_document, but with the desired value set at the desired path.
- virtual std::unique_ptr< config_document > with_value (std::string path, std::shared_ptr< config_value > new_value) const =0

  Returns a new config_document that is a copy of the current config_document, but with the desired value set at the desired path.
- virtual std::unique_ptr< config_document > without_path (std::string path) const =0

  Returns a new config_document that is a copy of the current config_document, but with all values at the desired path removed.
- virtual bool has_path (std::string const &path) const =0

  Returns a boolean indicating whether or not a config_document has a value at the desired path.
- virtual std::string render () const =0

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

7.12.1 Detailed Description

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

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

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

Do not implement interface
`config_document`

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

Definition at line 26 of file config_document.hpp.

7.12.2 Member Function Documentation

7.12.2.1 has_path()

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

Returns a boolean indicating whether or not a config_document has a value at the desired path.

null counts as a value for purposes of this check.

Parameters

path	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

path	the path at which to set the desired value
new_value	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

path	the path at which to set the desired value
new_value	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

path	the path to remove from the document

Returns

a copy of the config_document with the desired value removed from the document.

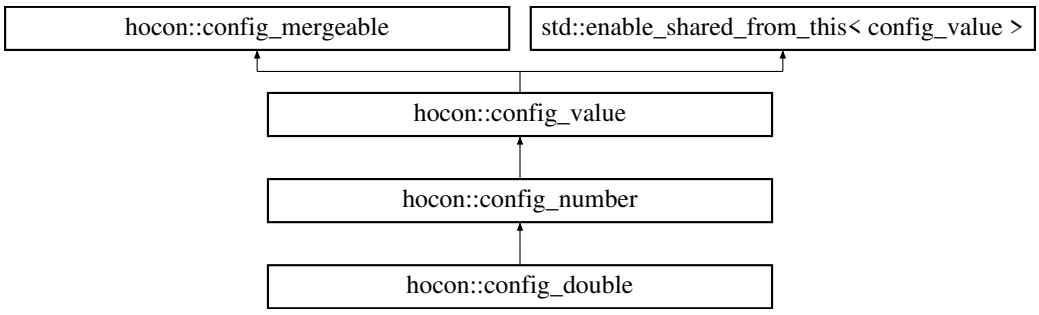
Implemented in hocon::simple_config_document.

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

- hocon/parser/config_document.hpp

7.13 hocon::config_double Class Reference

Inheritance diagram for hocon::config_double:



Public Types

- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

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

Public Member Functions

- **config_double** (shared_origin origin, double value, std::string original_text)
- std::string **transform_to_string** () const override
- unwrapped_value **unwrapped** () const override
- int64_t **long_value** () const override
- double **double_value** () const override
- config_value::type value_type () const override

  The type of the value; matches the JSON type schema.
- bool **is_whole** () const
- bool **operator==** (const config_number &other) const
- bool **operator==** (config_value const &other) const override
- bool **operator!=** (const config_number &other) const
- int **int_value_range_checked** (std::string const &path) const
- virtual shared_origin const & origin () const

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

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

  Renders the config value as a HOCON string.
- virtual std::string render (config_render_options options) const

  Renders the config value to a string, using the provided options.
- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.
- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.
- virtual shared_value with_origin (shared_origin origin) const

  Returns a.
- virtual shared_value relativized (std::string prefix) const

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- virtual resolve_status **get_resolve_status** () const
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

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

Static Public Member Functions

- static std::shared_ptr< config_number > **new_number** (shared_origin origin, int64_t value, std::string original_text)
- static std::shared_ptr< config_number > **new_number** (shared_origin origin, double value, std::string original_text)
- static char const * **type_name** (type t)

Protected Member Functions

- shared_value **new_copy** (shared_origin) const override
- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const
- virtual void **render** (std::string &result, int indent, bool at_root, config_render_options options) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- virtual resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const
- void **require_not_ignoring_fallbacks** () const
- virtual bool **ignores_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- virtual shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

Protected Attributes

- std::string **_original_text**

7.13.1 Detailed Description

Definition at line 10 of file config_double.hpp.

7.13.2 Member Enumeration Documentation

7.13.2.1 type

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

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

Definition at line 56 of file config_value.hpp.

7.13.3 Member Function Documentation

7.13.3.1 at_key()

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

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	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

path	path to store this value at.

Returns

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

7.13.3.3 origin()

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

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

Returns

where the value came from

7.13.3.4 relativized()

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

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

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

Parameters

prefix	

Returns

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

Reimplemented in hocon::config_concatenation, and hocon::simple_config_list.

Definition at line 181 of file config_value.hpp.

7.13.3.5 render() [1/2]

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

Renders the config value as a HOCON string.

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

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

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

.

Returns

the rendered value

7.13.3.6 render() [2/2]

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

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

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

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

Parameters

options	the rendering options

Returns

the rendered value

7.13.3.7 to_fallback_value()

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

Converts a config to its root object and a config_value to itself.

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

Implements hocon::config_mergeable.

7.13.3.8 value_type()

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

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

Returns

value's type

Implements hocon::config_value.

7.13.3.9 value_type_name()

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

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.13.3.10 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implements hocon::config_mergeable.

7.13.3.11 with_origin()

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

Returns a.
config_value

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

Parameters

origin	the origin set on the returned value

Returns

the new config_value with the given origin

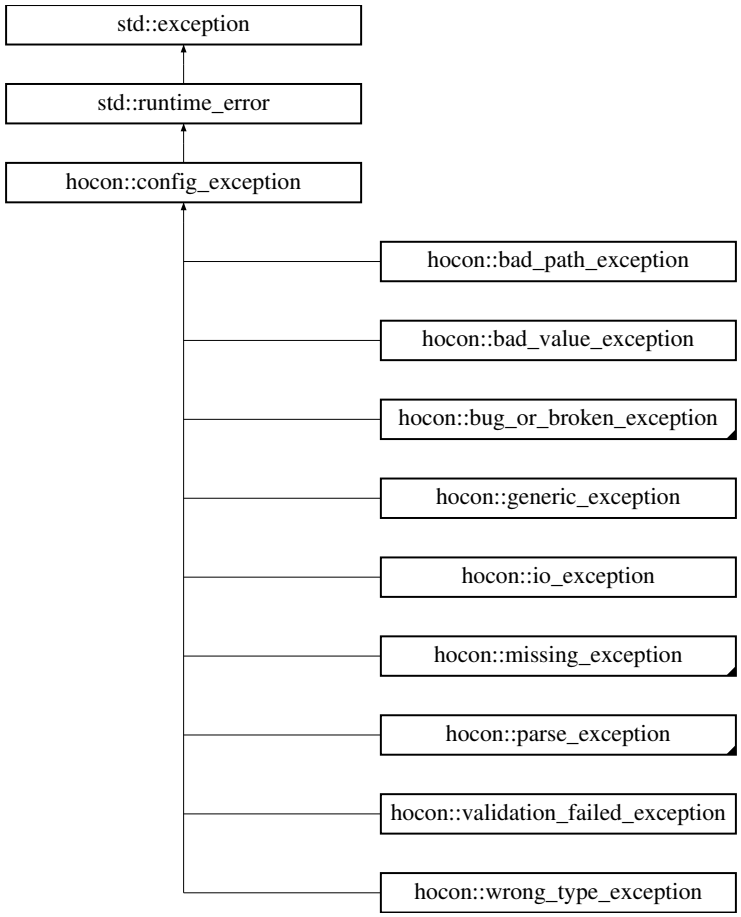
The documentation for this class was generated from the following file:

- internal/values/config_double.hpp

7.14 hocon::config_exception Struct Reference

All exceptions thrown by the library are subclasses of `config_exception`.

Inheritance diagram for hocon::config_exception:



Public Member Functions

- **config_exception** (config_origin const &origin, std::string const &message)
- **config_exception** (std::string const &message)
- **config_exception** (std::string const &message, std::exception const &e)

7.14.1 Detailed Description

All exceptions thrown by the library are subclasses of `config_exception`.

Definition at line 14 of file config_exception.hpp.

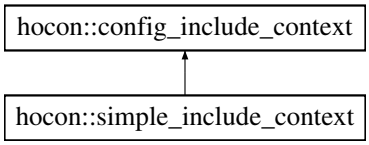
The documentation for this struct was generated from the following file:

- hocon/config_exception.hpp

7.15 hocon::config_include_context Class Reference

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

Inheritance diagram for hocon::config_include_context:



Public Member Functions

- virtual shared_parseable relative_to (std::string file_name) const =0

  Tries to find a name relative to whatever is doing the including, for example in the same directory as the file doing the including.
- virtual config_parse_options parse_options () const =0

  Parse options to use (if you use another method to get a config_parseable then use config_parseable#options() instead though).
- void **set_cur_dir** (std::string dir) const
- std::string **get_cur_dir** () const

Protected Attributes

- std::shared_ptr< std::string > **_cur_dir**

7.15.1 Detailed Description

Context provided to a config_includer; this interface is only useful inside a.
`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 Member Function Documentation

7.15.2.1 parse_options()

```
virtual config_parse_options hocon::config_include_context::parse_options ( ) const  [pure
virtual]
```

Parse options to use (if you use another method to get a config_parseable then use config_parseable#options() instead though).

Returns

the parse options

Implemented in hocon::simple_include_context.

7.15.2.2 relative_to()

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

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

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

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

Parameters

filename	the name to make relative to the resource doing the including

Returns

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

Implemented in hocon::simple_include_context.

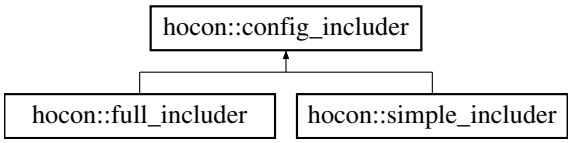
The documentation for this class was generated from the following file:

- hocon/config_include_context.hpp

7.16 hocon::config_includer Class Reference

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

Inheritance diagram for hocon::config_includer:



Public Member Functions

- virtual shared_includer with_fallback (shared_includer fallback) const =0

  Returns a new includer that falls back to the given includer.
- virtual shared_object include (shared_include_context context, std::string what) const =0

  Parses another item to be included.

7.16.1 Detailed Description

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

statements in config files. You may also want to implement config_includer_file and config_includer_URL, or not.

Definition at line 15 of file config_includer.hpp.

7.16.2 Member Function Documentation

7.16.2.1 include()

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

Parses another item to be included.

The returned object typically would not have substitutions resolved. You can throw a config_exception here to abort parsing, or return an empty object, but may not return null.

This method is used for a "heuristic" include statement that does not specify file, or URL resource. If the include statement does specify, then the same class implementing config_includer must also implement config_includer_file or config_includer_URL as needed, or a default includer will be used.

Parameters

context	some info about the include context
what	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

fallback	the previous includer for chaining

Returns

a new includer

Implemented in hocon::simple_includer.

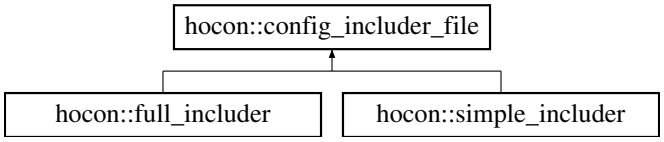
The documentation for this class was generated from the following file:

- hocon/config_includer.hpp

7.17 hocon::config_includer_file Class Reference

Implement this *in addition to* config_includer if you want to support inclusion of files with the.

Inheritance diagram for hocon::config_includer_file:



Public Member Functions

- virtual shared_object include_file (shared_include_context context, std::string what) const =0

  Parses another item to be included.

7.17.1 Detailed Description

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

syntax. If you do not implement this but do implement config_includer, attempts to load files will use the default includer.

Definition at line 14 of file config_includer_file.hpp.

7.17.2 Member Function Documentation

7.17.2.1 include_file()

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

Parses another item to be included.

The returned object typically would not have substitutions resolved. You can throw a config_exception here to abort parsing, or return an empty object, but may not return null.

Parameters

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

Returns

a non-null config_object

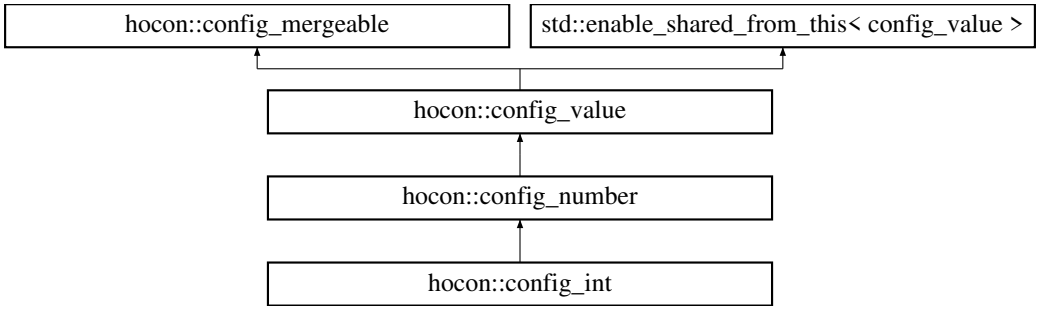
Implemented in hocon::simple_includer.

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

- hocon/config_includer_file.hpp

7.18 hocon::config_int Class Reference

Inheritance diagram for hocon::config_int:



Public Types

- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

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

Public Member Functions

- **config_int** (shared_origin origin, int value, std::string original_text)
- std::string **transform_to_string** () const override
- unwrapped_value **unwrapped** () const override
- int64_t **long_value** () const override
- double **double_value** () const override
- config_value::type value_type () const override

  The type of the value; matches the JSON type schema.
- bool **is_whole** () const
- bool **operator==** (const config_number &other) const
- bool **operator==** (config_value const &other) const override
- bool **operator!=** (const config_number &other) const
- int **int_value_range_checked** (std::string const &path) const
- virtual shared_origin const & origin () const

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

  The printable name of the value type.

- virtual std::string render () const

  Renders the config value as a HOCON string.
- virtual std::string render (config_render_options options) const

  Renders the config value to a string, using the provided options.
- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.
- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.
- virtual shared_value with_origin (shared_origin origin) const

  Returns a.
- virtual shared_value relativized (std::string prefix) const

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- virtual resolve_status **get_resolve_status** () const
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

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

Static Public Member Functions

- static std::shared_ptr< config_number > **new_number** (shared_origin origin, int64_t value, std::string original_text)
- static std::shared_ptr< config_number > **new_number** (shared_origin origin, double value, std::string original_text)
- static char const * **type_name** (type t)

Protected Member Functions

- shared_value **new_copy** (shared_origin) const override
- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const
- virtual void **render** (std::string &result, int indent, bool at_root, config_render_options options) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- virtual resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const
- void **require_not_ignoring_fallbacks** () const
- virtual bool **ignores_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- virtual shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

Protected Attributes

- std::string **_original_text**

7.18.1 Detailed Description

Definition at line 7 of file config_int.hpp.

7.18.2 Member Enumeration Documentation

7.18.2.1 type

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

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

Definition at line 56 of file config_value.hpp.

7.18.3 Member Function Documentation

7.18.3.1 at_key()

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

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	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

path	path to store this value at.

Returns

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

7.18.3.3 origin()

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

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

Returns

where the value came from

7.18.3.4 relativized()

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

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

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

Parameters

prefix	

Returns

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

Reimplemented in hocon::config_concatenation, and hocon::simple_config_list.

Definition at line 181 of file config_value.hpp.

7.18.3.5 render() [1/2]

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

Renders the config value as a HOCON string.

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

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

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

.

Returns

the rendered value

7.18.3.6 render() [2/2]

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

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

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

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

Parameters

options	the rendering options

Returns

the rendered value

7.18.3.7 to_fallback_value()

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

Converts a config to its root object and a config_value to itself.

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

Implements hocon::config_mergeable.

7.18.3.8 value_type()

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

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

Returns

value's type

Implements hocon::config_value.

7.18.3.9 value_type_name()

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

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.18.3.10 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implements hocon::config_mergeable.

7.18.3.11 with_origin()

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

Returns a.
config_value

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

Parameters

origin	the origin set on the returned value

Returns

the new config_value with the given origin

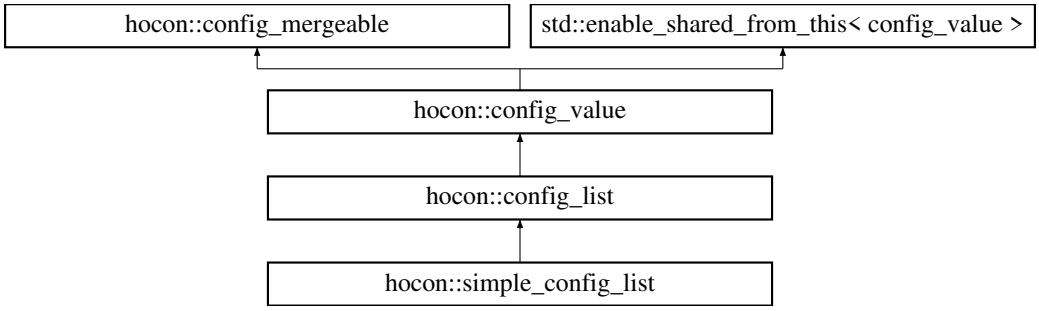
The documentation for this class was generated from the following file:

- internal/values/config_int.hpp

7.19 hocon::config_list Class Reference

Subtype of ConfigValue representing a list value, as in JSON's.

Inheritance diagram for hocon::config_list:



Public Types

- using **iterator** = std::vector< shared_value >::const_iterator
- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

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

Public Member Functions

- **config_list** (shared_origin origin)
- virtual bool **is_empty** () const =0
- virtual size_t **size** () const =0
- virtual shared_value **operator[ ]** (size_t index) const =0
- virtual shared_value **get** (size_t index) const =0
- virtual iterator **begin** () const =0
- virtual iterator **end** () const =0
- virtual unwrapped_value **unwrapped** () const =0
- virtual shared_origin const & origin () const

  The origin of the value (file, line number, etc.), for debugging and error messages.
- virtual type value_type () const =0

  The type of the value; matches the JSON type schema.
- char const * value_type_name () const

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

  Renders the config value as a HOCON string.
- virtual std::string render (config_render_options options) const

  Renders the config value to a string, using the provided options.
- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.
- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.
- virtual shared_value with_origin (shared_origin origin) const

  Returns a.
- virtual shared_value relativized (std::string prefix) const

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- virtual resolve_status **get_resolve_status** () const
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

  Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.
- virtual bool **operator==** (config_value const &other) const =0
- virtual std::string **transform_to_string** () const

Static Public Member Functions

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

Protected Member Functions

- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const
- virtual void **render** (std::string &result, int indent, bool at_root, config_render_options options) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- virtual shared_value **new_copy** (shared_origin origin) const =0
- virtual resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const
- void **require_not_ignoring_fallbacks** () const

- virtual bool **ignores_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- virtual shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

7.19.1 Detailed Description

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

syntax.
`ConfigList`

implements
`java.util.List<ConfigValue>`

so you can use it like a regular Java list. Or call unwrapped() to unwrap the list elements into plain Java values.

Like all ConfigValue subtypes,
`ConfigList`

is immutable. This makes it threadsafe and you never have to create "defensive copies." The mutator methods from java.util.List all throw java.lang.UnsupportedOperationException.

The ConfigValue#valueType method on a list returns ConfigValueType#LIST.

Do not implement
`ConfigList`

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

Definition at line 37 of file config_list.hpp.

7.19.2 Member Enumeration Documentation

7.19.2.1 type

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

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

Definition at line 56 of file config_value.hpp.

7.19.3 Member Function Documentation

7.19.3.1 at_key()

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

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	key to store this value at.

Returns

a
config
instance containing this value at the given key.

7.19.3.2 at_path()

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

Places the value inside a config at the given path.

See also config_value#at_key(String).

Parameters

path	path to store this value at.

Returns

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

7.19.3.3 origin()

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

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

Returns

where the value came from

7.19.3.4 relativized()

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

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

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

Parameters

prefix	

Returns

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

Reimplemented in hocon::config_concatenation, and hocon::simple_config_list.

Definition at line 181 of file config_value.hpp.

7.19.3.5 render() [1/2]

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

Renders the config value as a HOCON string.

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

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

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

.

Returns

the rendered value

7.19.3.6 render() [2/2]

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

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

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

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

Parameters

options	the rendering options

Returns

the rendered value

7.19.3.7 to_fallback_value()

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

Converts a config to its root object and a config_value to itself.

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

Implements hocon::config_mergeable.

7.19.3.8 value_type()

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

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

Returns

value's type

Implemented in hocon::simple_config_list, hocon::config_string, hocon::config_reference, hocon::config_number, hocon::config_null, hocon::config_delayed_merge, hocon::config_concatenation, hocon::config_boolean, and hocon::config_object.

7.19.3.9 value_type_name()

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

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.19.3.10 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implements hocon::config_mergeable.

7.19.3.11 with_origin()

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

Returns a.
config_value

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

Parameters

origin	the origin set on the returned value

Returns

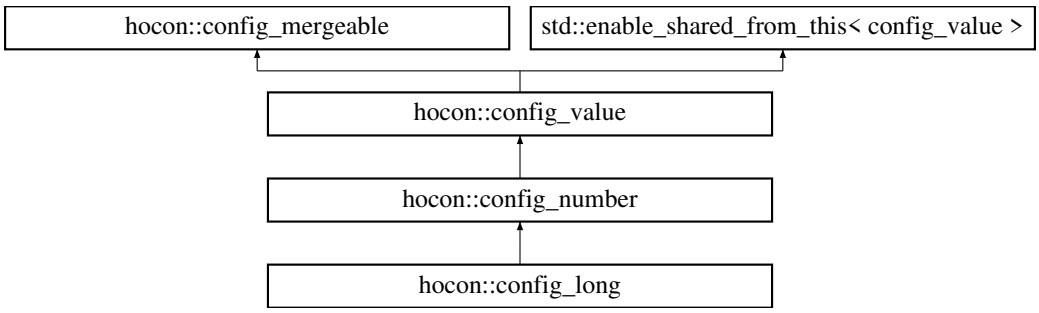
the new config_value with the given origin

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

- hocon/config_list.hpp

7.20 hocon::config_long Class Reference

Inheritance diagram for hocon::config_long:



Public Types

- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

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

Public Member Functions

- **config_long** (shared_origin origin, int64_t value, std::string original_text)
- std::string **transform_to_string** () const override
- unwrapped_value **unwrapped** () const override
- int64_t **long_value** () const override
- double **double_value** () const override
- config_value::type value_type () const override

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

- bool **is_whole** () const
- bool **operator==** (const config_number &other) const
- bool **operator==** (config_value const &other) const override
- bool **operator!=** (const config_number &other) const
- int **int_value_range_checked** (std::string const &path) const
- virtual shared_origin const & origin () const

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

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

  Renders the config value as a HOCON string.
- virtual std::string render (config_render_options options) const

  Renders the config value to a string, using the provided options.
- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.
- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.
- virtual shared_value with_origin (shared_origin origin) const

  Returns a.
- virtual shared_value relativized (std::string prefix) const

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- virtual resolve_status **get_resolve_status** () const
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

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

Static Public Member Functions

- static std::shared_ptr< config_number > **new_number** (shared_origin origin, int64_t value, std::string original_text)
- static std::shared_ptr< config_number > **new_number** (shared_origin origin, double value, std::string original_text)
- static char const * **type_name** (type t)

Protected Member Functions

- shared_value **new_copy** (shared_origin) const override
- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const
- virtual void **render** (std::string &result, int indent, bool at_root, config_render_options options) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- virtual resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const
- void **require_not_ignoring_fallbacks** () const
- virtual bool **ignores_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const

- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- virtual shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

Protected Attributes

- std::string **_original_text**

7.20.1 Detailed Description

Definition at line 10 of file config_long.hpp.

7.20.2 Member Enumeration Documentation

7.20.2.1 type

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

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

Definition at line 56 of file config_value.hpp.

7.20.3 Member Function Documentation

7.20.3.1 at_key()

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

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	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

path	path to store this value at.

Returns

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

7.20.3.3 origin()

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

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

Returns

where the value came from

7.20.3.4 relativized()

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

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

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

Parameters

prefix	

Returns

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

Reimplemented in hocon::config_concatenation, and hocon::simple_config_list.

Definition at line 181 of file config_value.hpp.

7.20.3.5 render() [1/2]

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

Renders the config value as a HOCON string.

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

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

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

.

Returns

the rendered value

7.20.3.6 render() [2/2]

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

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

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

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

Parameters

options	the rendering options

Returns

the rendered value

7.20.3.7 to_fallback_value()

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

Converts a config to its root object and a config_value to itself.

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

Implements hocon::config_mergeable.

7.20.3.8 value_type()

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

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

Returns

value's type

Implements hocon::config_value.

7.20.3.9 value_type_name()

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

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.20.3.10 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implements hocon::config_mergeable.

7.20.3.11 with_origin()

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

Returns a.
`config_value`

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

Parameters

origin	the origin set on the returned value

Returns

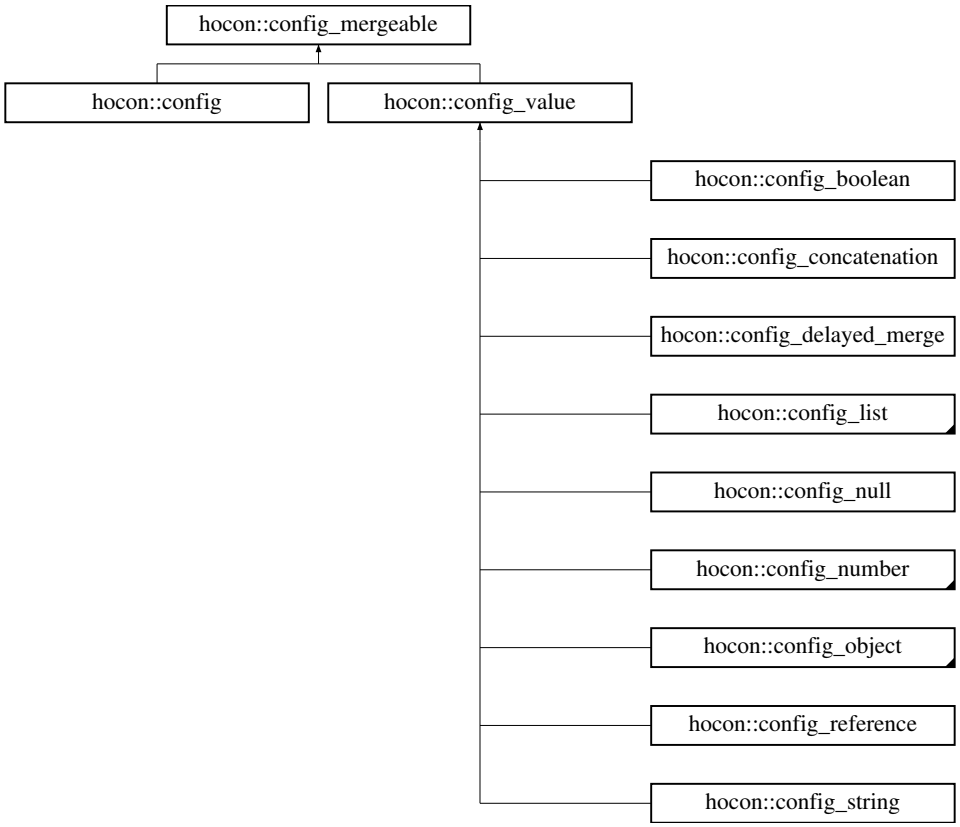
the new config_value with the given origin

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

- internal/values/config_long.hpp

7.21 hocon::config_mergeable Class Reference

Inheritance diagram for hocon::config_mergeable:



Public Member Functions

- virtual std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const =0

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

Protected Member Functions

- virtual shared_value to_fallback_value () const =0

  Converts a config to its root object and a config_value to itself.

Friends

- class **config_value**

7.21.1 Detailed Description

Definition at line 8 of file config_mergeable.hpp.

7.21.2 Member Function Documentation

7.21.2.1 to_fallback_value()

```
virtual shared_value hocon::config_mergeable::to_fallback_value ( ) const  [protected], [pure
virtual]
```

Converts a config to its root object and a config_value to itself.

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

Implemented in hocon::config_value, and hocon::config.

7.21.2.2 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implemented in hocon::config_value, and hocon::config.

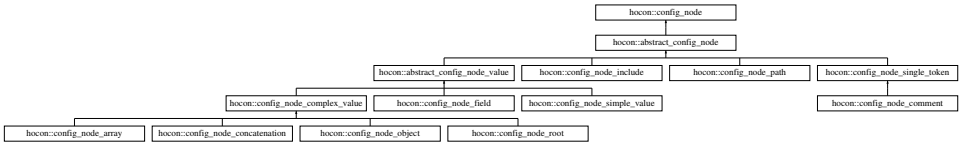
The documentation for this class was generated from the following file:

- hocon/config_mergeable.hpp

7.22 hocon::config_node Class Reference

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

Inheritance diagram for hocon::config_node:



Public Member Functions

- virtual std::string render () const =0

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

7.22.1 Detailed Description

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

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

Do not implement interface
`ConfigNode`

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

Definition at line 21 of file config_node.hpp.

7.22.2 Member Function Documentation

7.22.2.1 render()

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

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

Returns

the original text used to form this node as a String

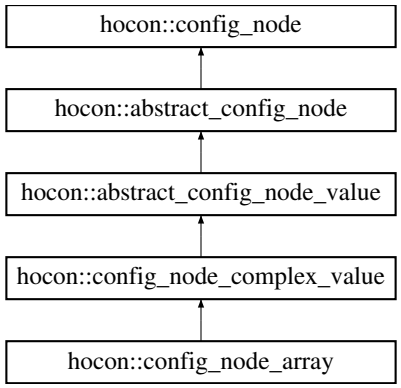
Implemented in hocon::abstract_config_node.

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

- hocon/parser/config_node.hpp

7.23 hocon::config_node_array Class Reference

Inheritance diagram for hocon::config_node_array:



Public Member Functions

- **config_node_array** (shared_node_list children)
- std::shared_ptr< const config_node_complex_value > **new_node** (shared_node_list nodes) const override
- token_list **get_tokens** () const override
- shared_node_list const & **children** () const
- std::shared_ptr< const config_node_complex_value > **indent_text** (shared_node indentation) const
- std::string render () const

  The original text of the input which was used to form this particular node.
- bool **operator==** (const abstract_config_node &other) const

7.23.1 Detailed Description

Definition at line 10 of file config_node_array.hpp.

7.23.2 Member Function Documentation

7.23.2.1 render()

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

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

Returns

the original text used to form this node as a String

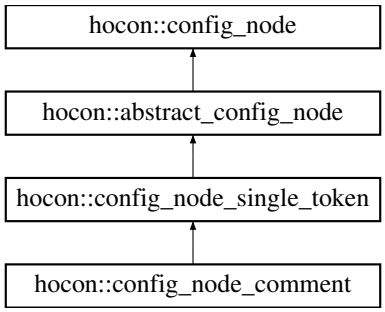
Implements hocon::config_node.

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

- internal/nodes/config_node_array.hpp

7.24 hocon::config_node_comment Class Reference

Inheritance diagram for hocon::config_node_comment:



Public Member Functions

- **config_node_comment** (shared_token comment)
- std::string **comment_text** () const
- token_list **get_tokens** () const override
- shared_token **get_token** () const
- std::string render () const

  The original text of the input which was used to form this particular node.
- bool **operator==** (const abstract_config_node &other) const

7.24.1 Detailed Description

Definition at line 7 of file config_node_comment.hpp.

7.24.2 Member Function Documentation

7.24.2.1 render()

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

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

Returns

the original text used to form this node as a String

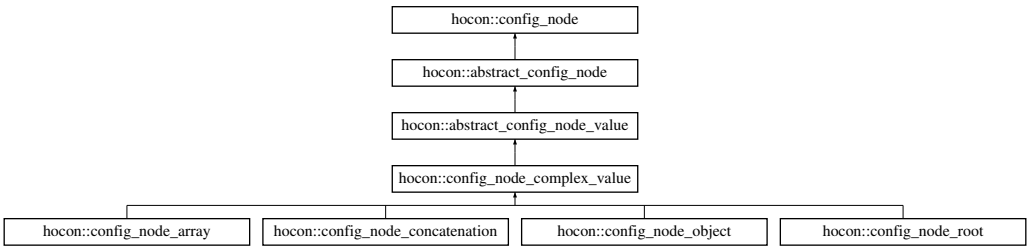
Implements hocon::config_node.

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

- internal/nodes/config_node_comment.hpp

7.25 hocon::config_node_complex_value Class Reference

Inheritance diagram for hocon::config_node_complex_value:



Public Member Functions

- **config_node_complex_value** (shared_node_list children)
- token_list **get_tokens** () const override
- shared_node_list const & **children** () const
- std::shared_ptr< const config_node_complex_value > **indent_text** (shared_node indentation) const
- virtual std::shared_ptr< const config_node_complex_value > **new_node** (shared_node_list nodes) const =0
- std::string render () const

  The original text of the input which was used to form this particular node.
- bool **operator==** (const abstract_config_node &other) const

7.25.1 Detailed Description

Definition at line 7 of file config_node_complex_value.hpp.

7.25.2 Member Function Documentation

7.25.2.1 render()

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

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

Returns

the original text used to form this node as a String

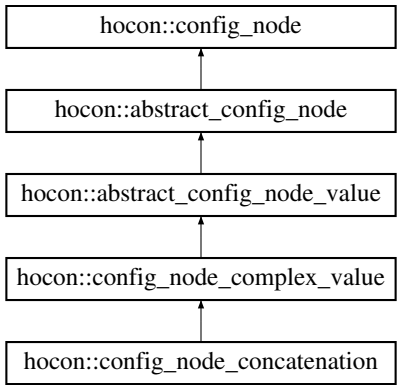
Implements hocon::config_node.

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

- internal/nodes/config_node_complex_value.hpp

7.26 hocon::config_node_concatenation Class Reference

Inheritance diagram for hocon::config_node_concatenation:



Public Member Functions

- **config_node_concatenation** (shared_node_list children)
- std::shared_ptr< const config_node_complex_value > **new_node** (shared_node_list nodes) const override
- token_list **get_tokens** () const override
- shared_node_list const & **children** () const
- std::shared_ptr< const config_node_complex_value > **indent_text** (shared_node indentation) const
- std::string render () const

  The original text of the input which was used to form this particular node.
- bool **operator==** (const abstract_config_node &other) const

7.26.1 Detailed Description

Definition at line 7 of file config_node_concatenation.hpp.

7.26.2 Member Function Documentation

7.26.2.1 render()

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

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

Returns

the original text used to form this node as a String

Implements hocon::config_node.

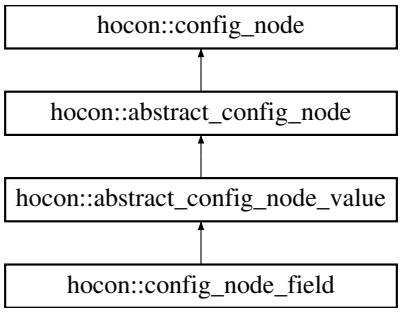
The documentation for this class was generated from the following file:

- internal/nodes/config_node_concatenation.hpp

7.27 hocon::config_node_field Class Reference

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

Inheritance diagram for hocon::config_node_field:



Public Member Functions

- **config_node_field** (shared_node_list children)
- token_list **get_tokens** () const override
- std::shared_ptr< const config_node_field > **replace_value** (shared_node_value new_value) const
- shared_node_value **get_value** () const
- shared_token **separator** () const
- std::vector< std::string > **comments** () const
- std::shared_ptr< const config_node_path > **path** () const
- std::string render () const

  The original text of the input which was used to form this particular node.
- bool **operator==** (const abstract_config_node &other) const

7.27.1 Detailed Description

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

Definition at line 13 of file config_node_field.hpp.

7.27.2 Member Function Documentation

7.27.2.1 render()

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

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

Returns

the original text used to form this node as a String

Implements hocon::config_node.

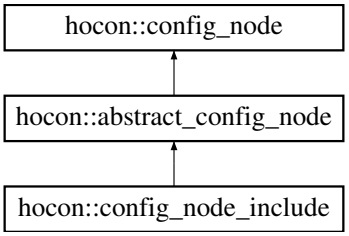
The documentation for this class was generated from the following file:

- internal/nodes/config_node_field.hpp

7.28 hocon::config_node_include Class Reference

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

Inheritance diagram for hocon::config_node_include:



Public Member Functions

- **config_node_include** (shared_node_list children, config_include_kind kind)
- token_list **get_tokens** () const override
- shared_node_list const & **children** () const
- config_include_kind **kind** () const
- std::string **name** () const
- std::string render () const

  The original text of the input which was used to form this particular node.
- bool **operator==** (const abstract_config_node &other) const

7.28.1 Detailed Description

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

Definition at line 10 of file config_node_include.hpp.

7.28.2 Member Function Documentation

7.28.2.1 render()

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

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

Returns

the original text used to form this node as a String

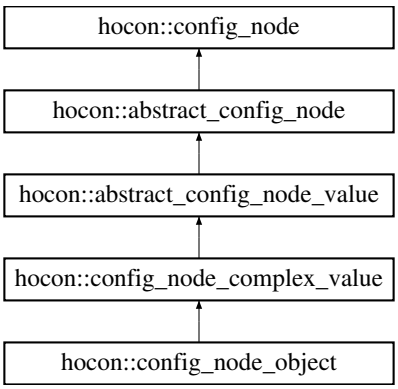
Implements hocon::config_node.

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

- internal/nodes/config_node_include.hpp

7.29 hocon::config_node_object Class Reference

Inheritance diagram for hocon::config_node_object:



Public Member Functions

- **config_node_object** (shared_node_list children)
- std::shared_ptr< const config_node_complex_value > **new_node** (shared_node_list nodes) const override
- bool **has_value** (path desired_path) const
- shared_node_object **change_value_on_path** (path desired_path, shared_node_value value, config_syntax flavor) const
- shared_node_object **set_value_on_path** (std::string desired_path, shared_node_value value, config_syntax flavor=config_syntax::CONF) const
- shared_node_object **set_value_on_path** (config_node_path desired_path, shared_node_value value, config_syntax flavor=config_syntax::CONF) const
- shared_node_list **indentation** () const
- shared_node_object **add_value_on_path** (config_node_path desired_path, shared_node_value value, config_syntax flavor) const
- shared_node_object **remove_value_on_path** (std::string desired_path, config_syntax flavor) const
- token_list **get_tokens** () const override
- shared_node_list const & **children** () const
- std::shared_ptr< const config_node_complex_value > **indent_text** (shared_node indentation) const
- std::string render () const

  The original text of the input which was used to form this particular node.
- bool **operator==** (const abstract_config_node &other) const

Static Public Member Functions

- static bool **contains_token** (shared_node node, token_type token)

7.29.1 Detailed Description

Definition at line 13 of file config_node_object.hpp.

7.29.2 Member Function Documentation

7.29.2.1 render()

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

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

Returns

the original text used to form this node as a String

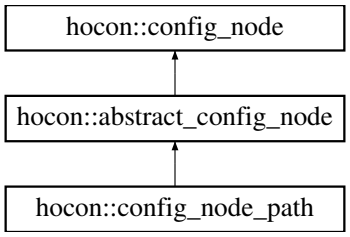
Implements hocon::config_node.

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

- internal/nodes/config_node_object.hpp

7.30 hocon::config_node_path Class Reference

Inheritance diagram for hocon::config_node_path:



Public Member Functions

- **config_node_path** (path node_path, token_list tokens)
- token_list **get_tokens** () const override
- path **get_path** () const
- config_node_path **sub_path** (int to_remove)
- config_node_path **first** ()
- std::string render () const

  The original text of the input which was used to form this particular node.
- bool **operator==** (const abstract_config_node &other) const

7.30.1 Detailed Description

Definition at line 9 of file config_node_path.hpp.

7.30.2 Member Function Documentation

7.30.2.1 render()

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

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

Returns

the original text used to form this node as a String

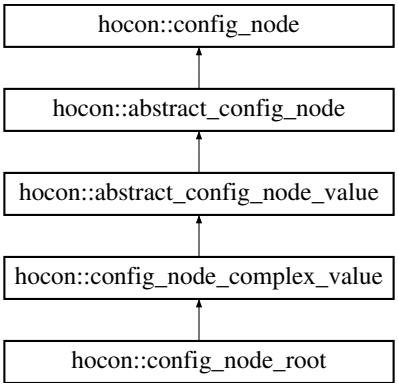
Implements hocon::config_node.

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

- internal/nodes/config_node_path.hpp

7.31 hocon::config_node_root Class Reference

Inheritance diagram for hocon::config_node_root:



Public Member Functions

- **config_node_root** (shared_node_list children, shared_origin origin)
- std::shared_ptr< const config_node_complex_value > **new_node** (shared_node_list nodes) const override
- std::shared_ptr< const config_node_complex_value > **value** () const
- std::shared_ptr< const config_node_root > **set_value** (std::string desired_path, shared_node_value, config_syntax flavor) const
- bool **has_value** (std::string desired_path) const
- token_list **get_tokens** () const override
- shared_node_list const & **children** () const
- std::shared_ptr< const config_node_complex_value > **indent_text** (shared_node indentation) const
- std::string render () const

  The original text of the input which was used to form this particular node.
- bool **operator==** (const abstract_config_node &other) const

7.31.1 Detailed Description

Definition at line 8 of file config_node_root.hpp.

7.31.2 Member Function Documentation

7.31.2.1 render()

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

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

Returns

the original text used to form this node as a String

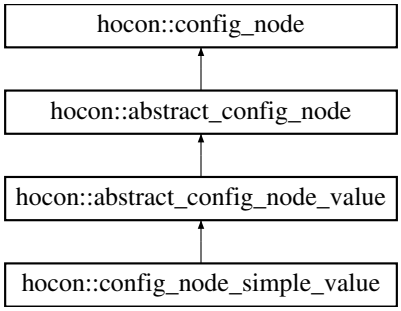
Implements hocon::config_node.

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

- internal/nodes/config_node_root.hpp

7.32 hocon::config_node_simple_value Class Reference

Inheritance diagram for hocon::config_node_simple_value:



Public Member Functions

- **config_node_simple_value** (shared_token value)
- shared_token **get_token** () const
- shared_value **get_value** () const
- token_list **get_tokens** () const override
- std::string render () const

  The original text of the input which was used to form this particular node.
- bool **operator==** (const abstract_config_node &other) const

7.32.1 Detailed Description

Definition at line 8 of file config_node_simple_value.hpp.

7.32.2 Member Function Documentation

7.32.2.1 render()

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

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

Returns

the original text used to form this node as a String

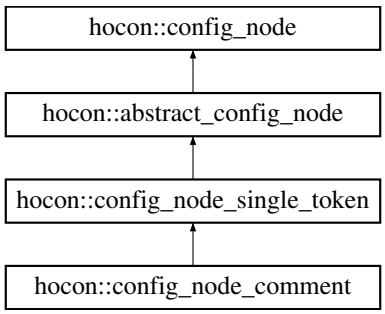
Implements hocon::config_node.

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

- internal/nodes/config_node_simple_value.hpp

7.33 hocon::config_node_single_token Class Reference

Inheritance diagram for hocon::config_node_single_token:



Public Member Functions

- **config_node_single_token** (shared_token t)
- token_list **get_tokens** () const override
- shared_token **get_token** () const
- std::string render () const

  The original text of the input which was used to form this particular node.
- bool **operator==** (const abstract_config_node &other) const

7.33.1 Detailed Description

Definition at line 8 of file config_node_single_token.hpp.

7.33.2 Member Function Documentation

7.33.2.1 render()

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

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

Returns

the original text used to form this node as a String

Implements hocon::config_node.

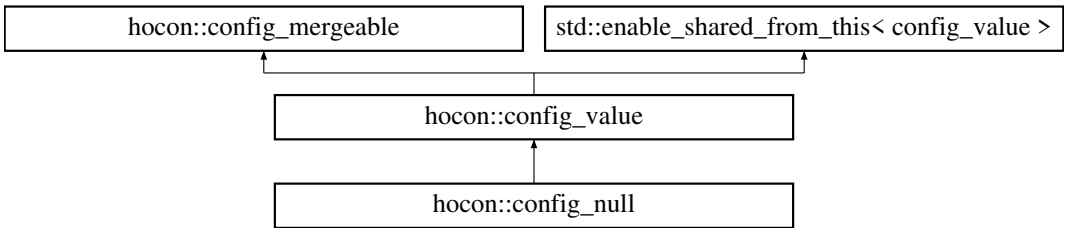
The documentation for this class was generated from the following file:

- internal/nodes/config_node_single_token.hpp

7.34 hocon::config_null Class Reference

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

Inheritance diagram for hocon::config_null:



Public Types

- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

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

Public Member Functions

- **config_null** (shared_origin origin)
- config_value::type value_type () const override

  The type of the value; matches the JSON type schema.
- std::string **transform_to_string** () const override
- unwrapped_value **unwrapped** () const override
- bool **operator==** (config_value const &other) const override
- virtual shared_origin const & origin () const

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

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

  Renders the config value as a HOCON string.
- virtual std::string render (config_render_options options) const

  Renders the config value to a string, using the provided options.
- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.
- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.
- virtual shared_value with_origin (shared_origin origin) const

  Returns a.
- virtual shared_value relativized (std::string prefix) const

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- virtual resolve_status **get_resolve_status** () const
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

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

Static Public Member Functions

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

Protected Member Functions

- shared_value **new_copy** (shared_origin) const override
- void **render** (std::string &result, int indent, bool at_root, config_render_options options) const override
- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- virtual resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const
- void **require_not_ignoring_fallbacks** () const
- virtual bool **ignores_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- virtual shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

7.34.1 Detailed Description

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

Specifically, if a value is set to null we can give a better error message (indicating where it was set to null) in case someone asks for the value. Also, null overrides values set "earlier" in the search path, while missing values do not.

Definition at line 17 of file config_null.hpp.

7.34.2 Member Enumeration Documentation

7.34.2.1 type

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

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

Definition at line 56 of file config_value.hpp.

7.34.3 Member Function Documentation

7.34.3.1 at_key()

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

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	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

path	path to store this value at.

Returns

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

7.34.3.3 origin()

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

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

Returns

where the value came from

7.34.3.4 relativized()

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

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

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

Parameters

prefix	

Returns

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

Reimplemented in hocon::config_concatenation, and hocon::simple_config_list.

Definition at line 181 of file config_value.hpp.

7.34.3.5 render() [1/2]

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

Renders the config value as a HOCON string.

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

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

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

.

Returns

the rendered value

7.34.3.6 render() [2/2]

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

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

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

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

Parameters

options	the rendering options

Returns

the rendered value

7.34.3.7 to_fallback_value()

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

Converts a config to its root object and a config_value to itself.

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

Implements hocon::config_mergeable.

7.34.3.8 value_type()

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

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

Returns

value's type

Implements hocon::config_value.

7.34.3.9 value_type_name()

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

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.34.3.10 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implements hocon::config_mergeable.

7.34.3.11 with_origin()

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

Returns a.
config_value

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

Parameters

origin	the origin set on the returned value

Returns

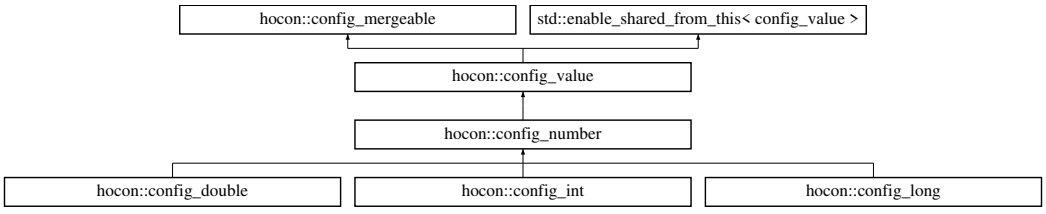
the new config_value with the given origin

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.
- virtual shared_value relativized (std::string prefix) const

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- virtual resolve_status **get_resolve_status** () const
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

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

Static Public Member Functions

- static std::shared_ptr< config_number > **new_number** (shared_origin origin, int64_t value, std::string original_text)
- static std::shared_ptr< config_number > **new_number** (shared_origin origin, double value, std::string original_text)
- static char const * **type_name** (type t)

Protected Member Functions

- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const
- virtual void **render** (std::string &result, int indent, bool at_root, config_render_options options) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- virtual shared_value **new_copy** (shared_origin origin) const =0
- virtual resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const
- void **require_not_ignoring_fallbacks** () const
- virtual bool **ignores_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- virtual shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

Protected Attributes

- std::string **_original_text**

7.35.1 Detailed Description

Definition at line 10 of file config_number.hpp.

7.35.2 Member Enumeration Documentation

7.35.2.1 type

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

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

Definition at line 56 of file config_value.hpp.

7.35.3 Member Function Documentation

7.35.3.1 at_key()

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

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	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

path	path to store this value at.

Returns

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

7.35.3.3 origin()

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

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

Returns

where the value came from

7.35.3.4 relativized()

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

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

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

Parameters

prefix	

Returns

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

Reimplemented in hocon::config_concatenation, and hocon::simple_config_list.

Definition at line 181 of file config_value.hpp.

7.35.3.5 render() [1/2]

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

Renders the config value as a HOCON string.

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

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

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

.

Returns

the rendered value

7.35.3.6 render() [2/2]

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

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

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

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

Parameters

options	the rendering options

Returns

the rendered value

7.35.3.7 to_fallback_value()

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

Converts a config to its root object and a config_value to itself.

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

Implements hocon::config_mergeable.

7.35.3.8 value_type()

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

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

Returns

value's type

Implements hocon::config_value.

7.35.3.9 value_type_name()

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

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.35.3.10 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implements hocon::config_mergeable.

7.35.3.11 with_origin()

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

Returns a.
config_value

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

Parameters

origin	the origin set on the returned value

Returns

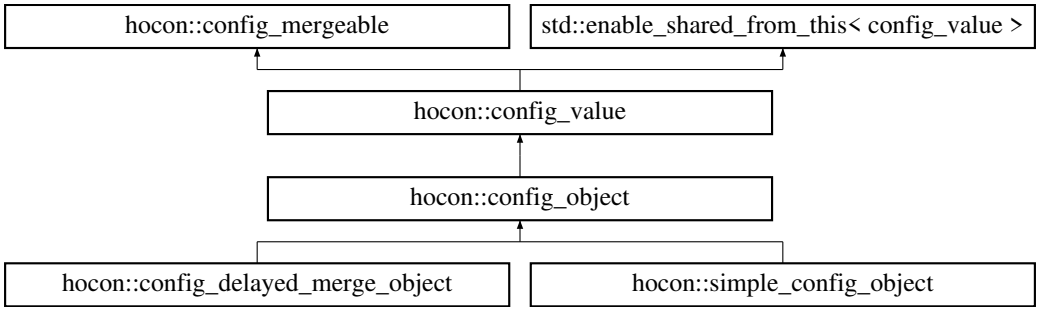
the new config_value with the given origin

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

- internal/values/config_number.hpp

7.36 hocon::config_object Class Reference

Inheritance diagram for hocon::config_object:



Public Types

- using **iterator** = std::unordered_map< std::string, shared_value >::const_iterator
- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

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

Public Member Functions

- virtual std::shared_ptr< const config > to_config () const

  Converts this object to a Config instance, enabling you to use path expressions to find values in the object.
- **config_object** (shared_origin origin)
- config_value::type value_type () const override

  The type of the value; matches the JSON type schema.
- virtual shared_object **with_value** (path raw_path, shared_value value) const =0
- virtual shared_object **with_value** (std::string key, shared_value value) const =0
- virtual shared_value attempt_peek_with_partial_resolve (std::string const &key) const =0

  Look up the key on an only-partially-resolved object, with no transformation or type conversion of any kind; if 'this' is not resolved then try to look up the key anyway if possible.
- virtual std::vector< std::string > key_set () const =0

  Construct a list of keys in the _value map.
- virtual bool **is_empty** () const =0
- virtual size_t **size** () const =0
- virtual shared_value **operator[ ]** (std::string const &key) const =0
- virtual shared_value **get** (std::string const &key) const =0
- virtual iterator **begin** () const =0
- virtual iterator **end** () const =0
- virtual shared_origin const & origin () const

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

  The printable name of the value type.
- virtual unwrapped_value **unwrapped** () const =0
- virtual std::string render () const

  Renders the config value as a HOCON string.
- virtual std::string render (config_render_options options) const

  Renders the config value to a string, using the provided options.
- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.
- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.
- virtual shared_value with_origin (shared_origin origin) const

  Returns a.
- virtual shared_value relativized (std::string prefix) const

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- virtual resolve_status **get_resolve_status** () const
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

  Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.
- virtual bool **operator==** (config_value const &other) const =0
- virtual std::string **transform_to_string** () const

Static Public Member Functions

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

Protected Member Functions

- shared_value **peek_path** (path desired_path) const
- shared_value **peek_assuming_resolved** (std::string const &key, path original_path) const
- virtual shared_object **new_copy** (resolve_status const &status, shared_origin origin) const =0
- shared_value **new_copy** (shared_origin origin) const override
- shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const override
- virtual std::unordered_map< std::string, shared_value > const & **entry_set** () const =0
- virtual shared_object **without_path** (path raw_path) const =0
- virtual shared_object **with_only_path** (path raw_path) const =0
- virtual shared_object **with_only_path_or_null** (path raw_path) const =0
- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const
- virtual void **render** (std::string &result, int indent, bool at_root, config_render_options options) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- virtual resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const
- void **require_not_ignoring_fallbacks** () const
- virtual bool **ignores_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static shared_value **peek_path** (const config_object *self, path desired_path)
- static shared_origin **merge_origins** (std::vector< shared_value > const &stack)
- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

Friends

- class **config**
- class **config_value**
- class **simple_config_object**
- class **resolve_source**
- class **config_delayed_merge_object**

7.36.1 Detailed Description

Definition at line 11 of file config_object.hpp.

7.36.2 Member Enumeration Documentation

7.36.2.1 type

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

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

Definition at line 56 of file config_value.hpp.

7.36.3 Member Function Documentation

7.36.3.1 at_key()

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

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	key to store this value at.

Returns

a
config
instance containing this value at the given key.

7.36.3.2 at_path()

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

Places the value inside a config at the given path.

See also config_value#at_key(String).

Parameters

path	path to store this value at.

Returns

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

7.36.3.3 attempt_peek_with_partial_resolve()

```
virtual shared_value hocon::config_object::attempt_peek_with_partial_resolve (
            std::string const & key ) const  [pure virtual]
```

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

Parameters

key	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

Implemented in hocon::simple_config_object, and hocon::config_delayed_merge_object.

7.36.3.4 key_set()

```
virtual std::vector<std::string> hocon::config_object::key_set ( ) const  [pure virtual]
```

Construct a list of keys in the _value map.

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

Implemented in hocon::simple_config_object, and hocon::config_delayed_merge_object.

7.36.3.5 origin()

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

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

Returns

where the value came from

7.36.3.6 relativized()

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

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

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

Parameters

prefix	

Returns

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

Reimplemented in hocon::config_concatenation, and hocon::simple_config_list.

Definition at line 181 of file config_value.hpp.

7.36.3.7 render() [1/2]

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

Renders the config value as a HOCON string.

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

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

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

.

Returns

the rendered value

7.36.3.8 render() [2/2]

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

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

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

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

Parameters

options	the rendering options

Returns

the rendered value

7.36.3.9 to_config()

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

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

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

Returns

a Config with this object as its root

7.36.3.10 to_fallback_value()

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

Converts a config to its root object and a config_value to itself.

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

Implements hocon::config_mergeable.

7.36.3.11 value_type()

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

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

Returns

value's type

Implements hocon::config_value.

7.36.3.12 value_type_name()

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

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.36.3.13 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implements hocon::config_mergeable.

7.36.3.14 with_origin()

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

Returns a.
config_value

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

Parameters

origin	the origin set on the returned value

Returns

the new config_value with the given origin

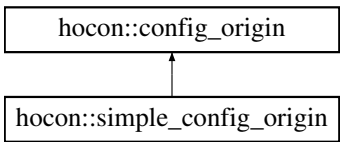
The documentation for this class was generated from the following file:

- hocon/config_object.hpp

7.37 hocon::config_origin Class Reference

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

Inheritance diagram for hocon::config_origin:



Public Member Functions

- virtual LIBCPP_HOCON_EXPORT std::string const & description () const =0

  Returns a string describing the origin of a value or exception.
- virtual LIBCPP_HOCON_EXPORT shared_origin with_line_number (int line_number) const =0

  Returns a.
- virtual LIBCPP_HOCON_EXPORT int line_number () const =0

  Returns a line number where the value or exception originated.
- virtual LIBCPP_HOCON_EXPORT std::vector< std::string > const & comments () const =0

  Returns any comments that appeared to "go with" this place in the file.
- virtual LIBCPP_HOCON_EXPORT shared_origin with_comments (std::vector< std::string > comments) const =0

  Returns a.

7.37.1 Detailed Description

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

Obtain the origin of a value with config_value#origin. Exceptions may have an origin, see config_exception#origin, but be careful because `config_exception.origin()` may return null.

It's best to use this interface only for debugging; its accuracy is "best effort" rather than guaranteed, and a potentially-noticeable amount of memory could probably be saved if origins were not kept around, so in the future there might be some option to discard origins.

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

Definition at line 31 of file config_origin.hpp.

7.37.2 Member Function Documentation

7.37.2.1 comments()

```
virtual LIBCPP_HOCON_EXPORT std::vector<std::string> const& hocon::config_origin::comments ( )
const  [pure virtual]
```

Returns any comments that appeared to "go with" this place in the file.

Often an empty list, but never null. The details of this are subject to change, but at the moment comments that are immediately before an array element or object field, with no blank line after the comment, "go with" that element or field.

Returns

any comments that seemed to "go with" this origin, empty list if none

Implemented in hocon::simple_config_origin.

7.37.2.2 description()

```
virtual LIBCPP_HOCON_EXPORT std::string const& hocon::config_origin::description ( ) const
[pure virtual]
```

Returns a string describing the origin of a value or exception.

This will never return null.

Returns

string describing the origin

Implemented in hocon::simple_config_origin.

7.37.2.3 line_number()

```
virtual LIBCPP_HOCON_EXPORT int hocon::config_origin::line_number ( ) const  [pure virtual]
```

Returns a line number where the value or exception originated.

This will return -1 if there's no meaningful line number.

Returns

line number or -1 if none is available

Implemented in hocon::simple_config_origin.

7.37.2.4 with_comments()

```
virtual LIBCPP_HOCON_EXPORT shared_origin hocon::config_origin::with_comments (
            std::vector< std::string > comments ) const  [pure virtual]
```

Returns a.
config_origin

based on this one, but with the given comments. Does not modify this instance or any
config_value

s with this origin (since they are immutable). To set the returned origin to a
config_value

, use config_value#with_origin.

Note that when the given comments are equal to the comments on this object, a new instance may not be created and
this

is returned directly.

Since

1.3.0

Parameters

comments	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

lineNumber	the new line number

Returns

the created ConfigOrigin

Implemented in hocon::simple_config_origin.

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

- hocon/config_origin.hpp

7.38 hocon::config_parse_options Class Reference

A set of options related to parsing.

Public Member Functions

- config_parse_options ()
  Gets an instance of `config_parse_options` with all fields set to the default values.
- config_parse_options set_syntax (config_syntax syntax) const
  Set the file format.
- config_syntax const & get_syntax () const
  Gets the current syntax option.
- config_parse_options set_origin_description (shared_string origin_description) const
  Set a description for the thing being parsed.
- shared_string const & get_origin_description () const
  Gets the current origin description, which may be null for "automatic".
- config_parse_options set_allow_missing (bool allow_missing) const
  Set to false to throw an exception if the item being parsed (for example a file) is missing.
- bool get_allow_missing () const
  Gets the current "allow missing" flag.
- config_parse_options set_includer (shared_includer includer) const
  Set a config_includer which customizes how includes are handled.
- config_parse_options prepend_includer (shared_includer includer) const
  Prepends a config_includer which customizes how includes are handled.
- config_parse_options append_includer (shared_includer includer) const
  Appends a config_includer which customizes how includes are handled.
- shared_includer const & get_includer () const
  Gets the current includer (will be null for the default includer).

Static Public Member Functions

- static config_parse_options defaults ()
  Gets an instance of `ConfigParseOptions` with all fields set to the default values.

7.38.1 Detailed Description

A set of options related to parsing.

This object is immutable, so the "setters" return a new object.

Here is an example of creating a custom
`config_parse_options`

:

```
config_parse_options options = config_parse_options()
    .set_syntax(config_syntax.JSON)
    .set_allow_missing(false)
```

ClassLoader is Java-specific, so it was not ported to C++.

Definition at line 25 of file config_parse_options.hpp.

7.38.2 Constructor & Destructor Documentation

7.38.2.1 config_parse_options()

```
hocon::config_parse_options::config_parse_options ( )
```

Gets an instance of `config_parse_options` with all fields set to the default values.

Start with this instance and make any changes you need.

Returns

the default parse options

7.38.3 Member Function Documentation

7.38.3.1 append_includer()

```
config_parse_options hocon::config_parse_options::append_includer (
            shared_includer includer ) const
```

Appends a config_includer which customizes how includes are handled.

To append, the library calls {} on the existing includer. includer the includer to append (may not be null) new version of the parse optio

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

includer	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

allow_missing	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

includer	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

origin_description	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

syntax	a syntax or `nullptr` for best guess

Returns

options with the syntax set

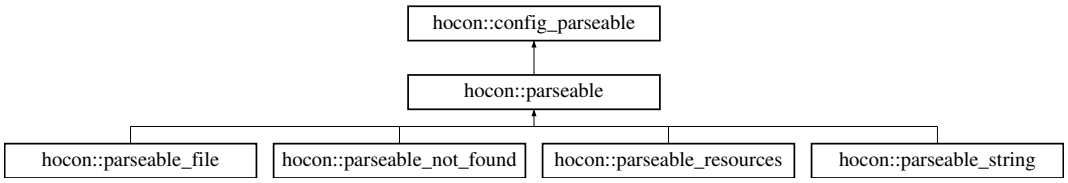
The documentation for this class was generated from the following file:

- hocon/config_parse_options.hpp

7.39 hocon::config_parseable Class Reference

An opaque handle to something that can be parsed, obtained from config_include_context.

Inheritance diagram for hocon::config_parseable:



Public Member Functions

- virtual shared_object parse (config_parse_options const &options) const =0

  Parse whatever it is.
- virtual shared_origin origin () const =0

  Returns a config_origin describing the origin of the 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

options	parse options, should be based on the ones from config_parseable#options()

Returns

the parsed object

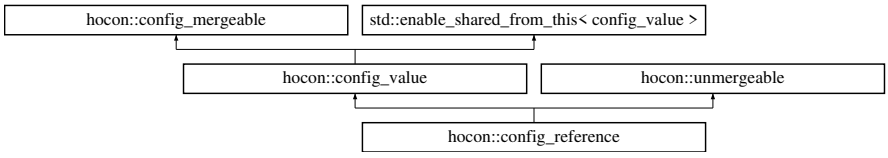
Implemented in hocon::parseable.

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

- hocon/config_parseable.hpp

7.40 hocon::config_reference Class Reference

Inheritance diagram for hocon::config_reference:



Public Types

- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

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

Public Member Functions

- **config_reference** (shared_origin origin, std::shared_ptr< substitution_expression > expr, int prefix_length=0)
- type value_type () const override

  The type of the value; matches the JSON type schema.
- std::vector< shared_value > **unmerged_values** () const override
- resolve_status **get_resolve_status** () const override
- unwrapped_value **unwrapped** () const override
- std::shared_ptr< substitution_expression > **expression** () const
- bool **operator==** (config_value const &other) const override
- virtual shared_origin const & origin () const

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

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

  Renders the config value as a HOCON string.
- virtual std::string render (config_render_options options) const

  Renders the config value to a string, using the provided options.
- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.
- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.
- virtual shared_value with_origin (shared_origin origin) const

  Returns a.
- virtual shared_value relativized (std::string prefix) const

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

  Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.
- virtual std::string **transform_to_string** () const

Static Public Member Functions

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

Protected Member Functions

- shared_value **new_copy** (shared_origin origin) const override
- resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const override
- bool **ignores_fallbacks** () const override
- void **render** (std::string &s, int indent, bool at_root, config_render_options options) const override
- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- void **require_not_ignoring_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const

- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- virtual shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

7.40.1 Detailed Description

Definition at line 10 of file config_reference.hpp.

7.40.2 Member Enumeration Documentation

7.40.2.1 type

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

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

Definition at line 56 of file config_value.hpp.

7.40.3 Member Function Documentation

7.40.3.1 at_key()

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

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	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

path	path to store this value at.

Returns

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

7.40.3.3 origin()

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

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

Returns

where the value came from

7.40.3.4 relativized()

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

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

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

Parameters

prefix	

Returns

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

Reimplemented in hocon::config_concatenation, and hocon::simple_config_list.

Definition at line 181 of file config_value.hpp.

7.40.3.5 render() [1/2]

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

Renders the config value as a HOCON string.

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

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

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

.

Returns

the rendered value

7.40.3.6 render() [2/2]

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

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

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

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

Parameters

options	the rendering options

Returns

the rendered value

7.40.3.7 to_fallback_value()

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

Converts a config to its root object and a config_value to itself.

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

Implements hocon::config_mergeable.

7.40.3.8 value_type()

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

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

Returns

value's type

Implements hocon::config_value.

7.40.3.9 value_type_name()

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

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.40.3.10 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implements hocon::config_mergeable.

7.40.3.11 with_origin()

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

Returns a.
`config_value`

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

Parameters

origin	the origin set on the returned value

Returns

the new config_value with the given origin

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

- internal/values/config_reference.hpp

7.41 hocon::config_render_options Class Reference

Public Member Functions

- config_render_options (bool origin_comments=true, bool comments=true, bool formatted=true, bool json=true)

  Leaving the default arguments will result in a verbose rendering, which contains comments and therefore is not valid JSON.
- config_render_options set_comments (bool value)

  Returns options with comments toggled.
- bool get_comments () const

  Returns whether the options enable comments.
- config_render_options set_origin_comments (bool value)

  Returns options with origin comments toggled.
- bool get_origin_comments () const

  Returns whether the options enable automated origin comments.
- config_render_options set_formatted (bool value)

  Returns options with formatting toggled.
- bool get_formatted () const

  Returns whether the options enable formatting.
- config_render_options set_json (bool value)

  Returns options with JSON toggled.
- bool get_json () const

  Returns whether the options enable JSON.

Static Public Member Functions

- static config_render_options concise ()

  Returns concise render options (no whitespace or comments).

7.41.1 Detailed Description

A set of options related to rendering a config_value. Passed to config_value#render(config_render_options).

Here is an example of creating a
config_render_options

:

```
config_render_options options =
    config_render_options().set_comments(false)
```

Definition at line 20 of file config_render_options.hpp.

7.41.2 Constructor & Destructor Documentation

7.41.2.1 config_render_options()

```
hocon::config_render_options::config_render_options (
            bool origin_comments = true,
            bool comments = true,
            bool formatted = true,
            bool json = true )
```

Leaving the default arguments will result in a verbose rendering, which contains comments and therefore is not valid JSON.

See config_render_options#concise for stripped-down options.

7.41.3 Member Function Documentation

7.41.3.1 concise()

```
static config_render_options hocon::config_render_options::concise ( )  [static]
```

Returns concise render options (no whitespace or comments).

For a resolved config, the concise rendering will be valid JSON.

Returns

the concise render options

7.41.3.2 get_comments()

```
bool hocon::config_render_options::get_comments ( ) const
```

Returns whether the options enable comments.

This method is mostly used by the config lib internally, not by applications.

Returns

true if comments should be rendered

7.41.3.3 get_formatted()

```
bool hocon::config_render_options::get_formatted ( ) const
```

Returns whether the options enable formatting.

This method is mostly used by the config lib internally, not by applications.

Returns

true if the options enable formatting

7.41.3.4 get_json()

```
bool hocon::config_render_options::get_json ( ) const
```

Returns whether the options enable JSON.

This method is mostly used by the config lib internally, not by applications.

Returns

true if only JSON should be rendered

7.41.3.5 get_origin_comments()

```
bool hocon::config_render_options::get_origin_comments ( ) const
```

Returns whether the options enable automated origin comments.

This method is mostly used by the config lib internally, not by applications.

Returns

true if origin comments should be rendered

7.41.3.6 set_comments()

```
config_render_options hocon::config_render_options::set_comments (
            bool value )
```

Returns options with comments toggled.

This controls human-written comments but not the autogenerated "origin of this setting" comments, which are controlled by config_render_options#set_origin_comments.

Parameters

value	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

value	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

value	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

value	true to include autogenerated setting-origin comments in the render

Returns

options with origin comments toggled

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

- hocon/config_render_options.hpp

7.42 hocon::config_resolve_options Class Reference

A set of options related to resolving substitutions.

Public Member Functions

- config_resolve_options (bool use_system_environment=true, bool allow_unresolved=false)
  Returns the default resolve options.
- config_resolve_options set_use_system_environment (bool value) const
  Returns resolve options that disable any reference to "system" data (currently, this means environment variables).
- bool get_use_system_environment () const
  Returns whether the options enable use of system environment variables.
- config_resolve_options set_allow_unresolved (bool value) const
  Returns options with "allow unresolved" set to the given value.
- bool get_allow_unresolved () const
  Returns whether the options allow unresolved substitutions.

7.42.1 Detailed Description

A set of options related to resolving substitutions.

Substitutions use the `${foo.bar}` syntax and are documented in the HOCON spec.

Typically this class would be used with the method config#resolve(config_resolve_options).

This object is immutable, so the "setters" return a new object.

Here is an example of creating a custom
config_resolve_options

:

```
config_resolve_options options = config_resolve_options()
    .set_use_system_environment(false)
```

<p<blockquote>

In addition to config_resolve_options, there's a prebuilt config_resolve_options#no_system which avoids looking at any system environment variables or other external system information. (Right now, environment variables are the only example.)

Definition at line 30 of file config_resolve_options.hpp.

7.42.2 Constructor & Destructor Documentation

7.42.2.1 config_resolve_options()

```
hocon::config_resolve_options::config_resolve_options (
            bool use_system_environment = true,
            bool allow_unresolved = false )
```

Returns the default resolve options.

By default the system environment will be used and unresolved substitutions are not allowed.

Returns

the default resolve options

7.42.3 Member Function Documentation

7.42.3.1 get_allow_unresolved()

```
bool hocon::config_resolve_options::get_allow_unresolved ( ) const
```

Returns whether the options allow unresolved substitutions.

This method is mostly used by the config lib internally, not by applications.

Returns

true if unresolved substitutions are allowed

7.42.3.2 get_use_system_environment()

```
bool hocon::config_resolve_options::get_use_system_environment ( ) const
```

Returns whether the options enable use of system environment variables.

This method is mostly used by the config lib internally, not by applications.

Returns

true if environment variables should be used

7.42.3.3 set_allow_unresolved()

```
config_resolve_options hocon::config_resolve_options::set_allow_unresolved (
            bool value ) const
```

Returns options with "allow unresolved" set to the given value.

By default, unresolved substitutions are an error. If unresolved substitutions are allowed, then a future attempt to use the unresolved value may fail, but config#resolve(config_resolve_options) itself will not throw.

Parameters

value	true to silently ignore unresolved substitutions.

Returns

options with requested setting for whether to allow substitutions

7.42.3.4 set_use_system_environment()

```
config_resolve_options hocon::config_resolve_options::set_use_system_environment (
            bool value ) const
```

Returns resolve options that disable any reference to "system" data (currently, this means environment variables).

Returns

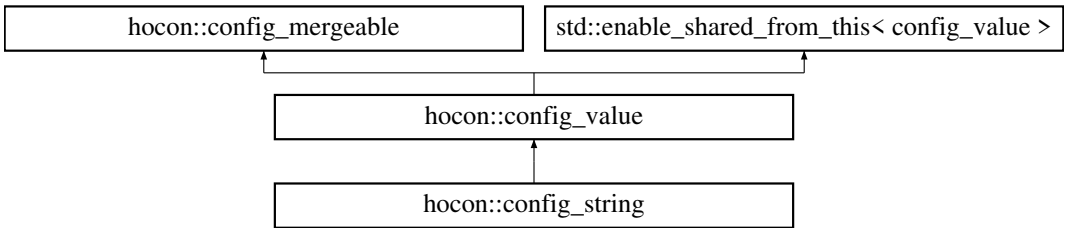
the resolve options with env variables disabled

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

- hocon/config_resolve_options.hpp

7.43 hocon::config_string Class Reference

Inheritance diagram for hocon::config_string:



Public Types

- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

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

Public Member Functions

- **config_string** (shared_origin origin, std::string text, config_string_type quoted)
- config_value::type value_type () const override

  The type of the value; matches the JSON type schema.
- std::string **transform_to_string** () const override
- unwrapped_value **unwrapped** () const override
- bool **was_quoted** () const
- bool **operator==** (config_value const &other) const override
- virtual shared_origin const & origin () const

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

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

Renders the config value as a HOCON string.

- virtual std::string render (config_render_options options) const

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

- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.

- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.

- virtual shared_value with_origin (shared_origin origin) const

  Returns a.

- virtual shared_value relativized (std::string prefix) const

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

- virtual resolve_status **get_resolve_status** () const
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

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

Static Public Member Functions

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

Protected Member Functions

- shared_value **new_copy** (shared_origin) const override
- void **render** (std::string &s, int indent, bool at_root, config_render_options options) const override
- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- virtual resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const
- void **require_not_ignoring_fallbacks** () const
- virtual bool **ignores_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- virtual shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

7.43.1 Detailed Description

Definition at line 10 of file config_string.hpp.

7.43.2 Member Enumeration Documentation

7.43.2.1 type

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

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

Definition at line 56 of file config_value.hpp.

7.43.3 Member Function Documentation

7.43.3.1 at_key()

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

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	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

path	path to store this value at.

Returns

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

7.43.3.3 origin()

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

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

Returns

where the value came from

7.43.3.4 relativized()

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

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

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

Parameters

prefix	

Returns

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

Reimplemented in hocon::config_concatenation, and hocon::simple_config_list.

Definition at line 181 of file config_value.hpp.

7.43.3.5 render() [1/2]

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

Renders the config value as a HOCON string.

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

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

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

.

Returns

the rendered value

7.43.3.6 render() [2/2]

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

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

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

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

Parameters

options	the rendering options

Returns

the rendered value

7.43.3.7 to_fallback_value()

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

Converts a config to its root object and a config_value to itself.

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

Implements hocon::config_mergeable.

7.43.3.8 value_type()

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

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

Returns

value's type

Implements hocon::config_value.

7.43.3.9 value_type_name()

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

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.43.3.10 with_fallback()

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

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

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

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

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

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

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

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

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

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

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

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

Implements hocon::config_mergeable.

7.43.3.11 with_origin()

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

Returns a.
`config_value`

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

Parameters

origin	the origin set on the returned value

Returns

the new config_value with the given origin

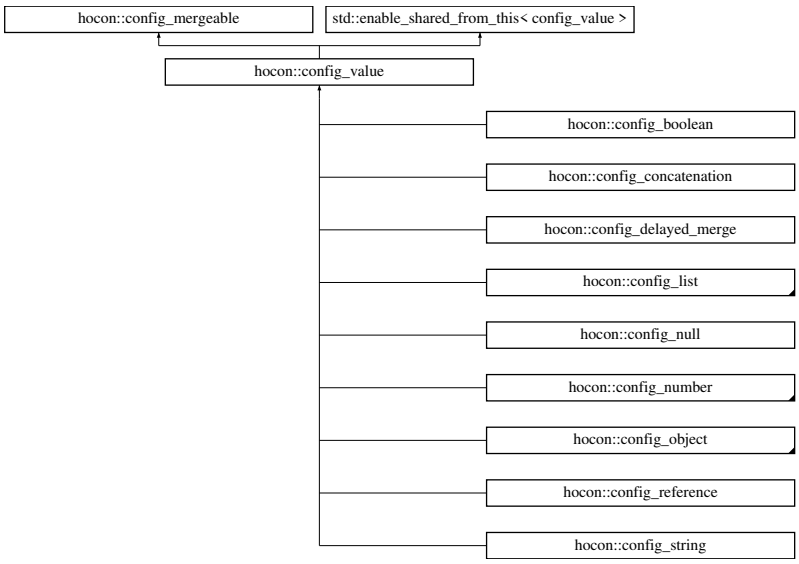
The documentation for this class was generated from the following file:

- internal/values/config_string.hpp

7.44 hocon::config_value Class Reference

An immutable value, following the JSON type schema.

Inheritance diagram for hocon::config_value:



Classes

- class modifier
- class no_exceptions_modifier

Public Types

- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

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

Public Member Functions

- virtual shared_origin const & origin () const

  The origin of the value (file, line number, etc.), for debugging and error messages.
- virtual type value_type () const =0

  The type of the value; matches the JSON type schema.
- char const * value_type_name () const

  The printable name of the value type.
- virtual unwrapped_value **unwrapped** () const =0
- virtual std::string render () const

  Renders the config value as a HOCON string.
- virtual std::string render (config_render_options options) const

  Renders the config value to a string, using the provided options.
- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.
- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.
- virtual shared_value with_origin (shared_origin origin) const

  Returns a.
- virtual shared_value relativized (std::string prefix) const

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- virtual resolve_status **get_resolve_status** () const
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

  Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.
- virtual bool **operator==** (config_value const &other) const =0
- virtual std::string **transform_to_string** () const

Static Public Member Functions

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

Protected Member Functions

- **config_value** (shared_origin origin)
- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const
- virtual void **render** (std::string &result, int indent, bool at_root, config_render_options options) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- virtual shared_value **new_copy** (shared_origin origin) const =0

- virtual resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const
- void **require_not_ignoring_fallbacks** () const
- virtual bool **ignores_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- virtual shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

Friends

- class **token**
- class **value**
- class **default_transformer**
- class **config**
- class **config_object**
- class **simple_config_object**
- class **simple_config_list**
- class **config_concatenation**
- class **resolve_context**
- class **config_delayed_merge**
- class **config_delayed_merge_object**
- resolve_status **resolve_status_from_values** (std::vector< shared_value > const &v)

7.44.1 Detailed Description

An immutable value, following the JSON type schema.

Because this object is immutable, it is safe to use from multiple threads and there's no need for "defensive copies."

Do not implement interface
`ConfigValue`

; it should only be implemented by the config library. Arbitrary implementations will not work because the library internals assume a specific concrete implementation. Also, this interface is likely to grow new methods over time, so third-party implementations will break.

Definition at line 39 of file config_value.hpp.

7.44.2 Member Enumeration Documentation

7.44.2.1 type

```
enum hocon::config_value::type  [strong]
```

The type of a configuration value (following the JSON type schema).

Definition at line 56 of file config_value.hpp.

7.44.3 Member Function Documentation

7.44.3.1 at_key()

```
shared_config hocon::config_value::at_key (
            std::string const & key ) const
```

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	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

path	path to store this value at.

Returns

a
`config`
instance containing this value at the given path.

7.44.3.3 origin()

```
virtual shared_origin const& hocon::config_value::origin ( ) const  [virtual]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

Returns

where the value came from

7.44.3.4 relativized()

```
virtual shared_value hocon::config_value::relativized (
            std::string prefix ) const  [inline], [virtual]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at foo.bar in the parent, and the included file as a substitution ${a.b.c}, the included substitution now needs to be ${foo.bar.a.b.c} because we resolve substitutions globally only after parsing everything.

Parameters

prefix	

Returns

value relativized to the given path or the same value if nothing to do

Reimplemented in hocon::config_concatenation, and hocon::simple_config_list.

Definition at line 181 of file config_value.hpp.

7.44.3.5 render() [1/2]

```
virtual std::string hocon::config_value::render ( ) const  [virtual]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see config#resolve), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to
`render(config_render_options())`

.

Returns

the rendered value

7.44.3.6 render() [2/2]

```
virtual std::string hocon::config_value::render (
            config_render_options options ) const  [virtual]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see config#resolve), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

Parameters

options	the rendering options

Returns

the rendered value

7.44.3.7 to_fallback_value()

```
shared_value hocon::config_value::to_fallback_value ( ) const  [override], [protected], [virtual]
```

Converts a config to its root object and a config_value to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements hocon::config_mergeable.

7.44.3.8 value_type()

```
virtual type hocon::config_value::value_type ( ) const  [pure virtual]
```

The type of the value; matches the JSON type schema.

Returns

value's type

Implemented in hocon::simple_config_list, hocon::config_string, hocon::config_reference, hocon::config_number, hocon::config_null, hocon::config_delayed_merge, hocon::config_concatenation, hocon::config_boolean, and hocon::config_object.

7.44.3.9 value_type_name()

```
char const* hocon::config_value::value_type_name ( ) const  [inline]
```

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.44.3.10 with_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
            std::shared_ptr< const config_mergeable > other ) const  [override], [virtual]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the spec for HOCON. Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

a new object (or the original one, if the fallback doesn't get used)

Implements hocon::config_mergeable.

7.44.3.11 with_origin()

```
virtual shared_value hocon::config_value::with_origin (
            shared_origin origin ) const  [virtual]
```

Returns a.
`config_value`

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single config_value.

Parameters

origin	the origin set on the returned value

Returns

the new config_value with the given origin

The documentation for this class was generated from the following file:

- hocon/config_value.hpp

7.45 hocon::config_value_factory Class Reference

Static Public Member Functions

- static shared_value from_any_ref (unwrapped_value value, std::string origin_description="")

  Creates a ConfigValue from a plain value, which may be a `bool`, `long`, `string`, `unordered_map`, `vector` or `nullptr`.

7.45.1 Detailed Description

Definition at line 8 of file config_value_factory.hpp.

7.45.2 Member Function Documentation

7.45.2.1 from_any_ref()

```
static shared_value hocon::config_value_factory::from_any_ref (
            unwrapped_value value,
            std::string origin_description = "" )  [static]
```

Creates a ConfigValue from a plain value, which may be a `bool`, `long`, `string`, `unordered_map`, `vector` or `nullptr`.

An `unordered_map` must be a `unordered_map` from string to more values that can be supplied to `from_any_ref()`. An `unordered_map` will become a ConfigObject and a `vector` will become a ConfigList.

In a `unordered_map` passed to `from_any_ref()`, the map's keys are plain keys, not path expressions. So if your `unordered_map` has a key "foo.bar" then you will get one object with a key called "foo.bar", rather than an object with a key "foo" containing another object with a key "bar".

The origin_description will be used to set the origin() field on the ConfigValue. It should normally be the name of the file the values came from, or something short describing the value such as "default settings". The origin_description is prefixed to error messages so users can tell where problematic values are coming from.

Supplying the result of ConfigValue.unwrapped() to this function is guaranteed to work and should give you back a ConfigValue that matches the one you unwrapped. The re-wrapped ConfigValue will lose some information that was present in the original such as its origin, but it will have matching values.

Parameters

unwrapped_value	object object to convert to ConfigValue
string	origin_description name of origin file or brief description of what the value is

Returns

shared_value a new value

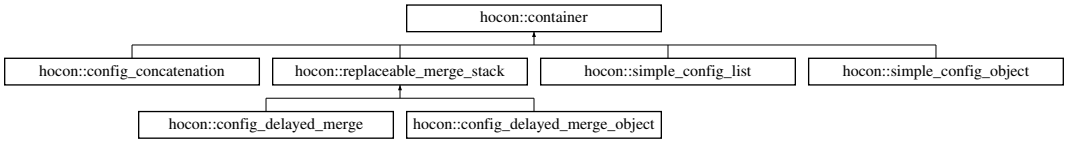
The documentation for this class was generated from the following file:

- hocon/config_value_factory.hpp

7.46 hocon::container Class Reference

An AbstractConfigValue which contains other values.

Inheritance diagram for hocon::container:



Public Member Functions

- virtual shared_value replace_child (shared_value const &child, shared_value replacement) const =0

  Replace a child of this value.
- virtual bool has_descendant (shared_value const &descendant) const =0

  Super-expensive full traversal to see if descendant is anywhere underneath this container.

7.46.1 Detailed Description

An AbstractConfigValue which contains other values.

Java has no way to express "this has to be an AbstractConfigValue also" other than making AbstractConfigValue an interface which would be aggravating. But we can say we are a ConfigValue.

Definition at line 12 of file container.hpp.

7.46.2 Member Function Documentation

7.46.2.1 has_descendant()

```
virtual bool hocon::container::has_descendant (
            shared_value const & descendant ) const  [pure virtual]
```

Super-expensive full traversal to see if descendant is anywhere underneath this container.

Implemented in hocon::simple_config_object, hocon::simple_config_list, hocon::config_delayed_merge_object, hocon::config_delayed_merge, and hocon::config_concatenation.

7.46.2.2 replace_child()

```
virtual shared_value hocon::container::replace_child (
            shared_value const & child,
            shared_value replacement ) const  [pure virtual]
```

Replace a child of this value.

CAUTION if replacement is null, delete the child, which may also delete the parent, or make the parent into a non-container.

Implemented in hocon::simple_config_object, hocon::simple_config_list, hocon::config_delayed_merge_object, hocon::config_delayed_merge, and hocon::config_concatenation.

The documentation for this class was generated from the following file:

- internal/container.hpp

7.47 hocon::default_transformer Class Reference

Static Public Member Functions

- static shared_value **transform** (shared_value value, config_value::type requested)

7.47.1 Detailed Description

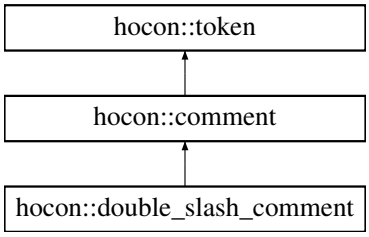
Definition at line 7 of file default_transformer.hpp.

The documentation for this class was generated from the following file:

- internal/default_transformer.hpp

7.48 hocon::double_slash_comment Class Reference

Inheritance diagram for hocon::double_slash_comment:



Public Member Functions

- **double_slash_comment** (shared_origin origin, std::string text)
- std::string **token_text** () const override
- std::string **text** () const
- std::string **to_string** () const override
- bool **operator==** (const token &other) const override
- virtual token_type **get_token_type** () const
- virtual shared_origin const & **origin** () const
- int **line_number** () const

7.48.1 Detailed Description

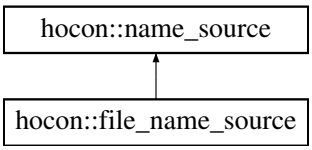
Definition at line 82 of file tokens.hpp.

The documentation for this class was generated from the following file:

- internal/tokens.hpp

7.49 hocon::file_name_source Class Reference

Inheritance diagram for hocon::file_name_source:



Public Member Functions

- **file_name_source** (shared_include_context context)
- shared_parseable **name_to_parseable** (std::string name, config_parse_options parse_options) const override
- void **set_context** (shared_include_context context)
- shared_include_context **get_context** () const
- bool **context_initialized** () const

7.49.1 Detailed Description

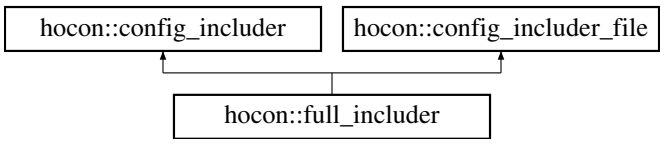
Definition at line 98 of file simple_includer.hpp.

The documentation for this class was generated from the following file:

- internal/simple_includer.hpp

7.50 hocon::full_includer Class Reference

Inheritance diagram for hocon::full_includer:



Public Member Functions

- virtual shared_includer with_fallback (shared_includer fallback) const =0

  Returns a new includer that falls back to the given includer.
- virtual shared_object include (shared_include_context context, std::string what) const =0

  Parses another item to be included.
- virtual shared_object include_file (shared_include_context context, std::string what) const =0

  Parses another item to be included.

7.50.1 Detailed Description

Definition at line 9 of file full_includer.hpp.

7.50.2 Member Function Documentation

7.50.2.1 include()

```
virtual shared_object hocon::config_includer::include (
            shared_include_context context,
            std::string what ) const  [pure virtual], [inherited]
```

Parses another item to be included.

The returned object typically would not have substitutions resolved. You can throw a config_exception here to abort parsing, or return an empty object, but may not return null.

This method is used for a "heuristic" include statement that does not specify file, or URL resource. If the include statement does specify, then the same class implementing config_includer must also implement config_includer_file or config_includer_URL as needed, or a default includer will be used.

Parameters

context	some info about the include context
what	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

context	some info about the include context
what	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

fallback	the previous includer for chaining

Returns

a new includer

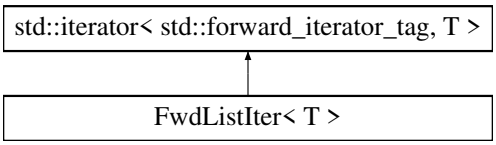
Implemented in hocon::simple_includer.

The documentation for this class was generated from the following file:

- internal/full_includer.hpp

7.51 FwdListIter< T > Class Template Reference

Inheritance diagram for FwdListIter< T >:



Public Member Functions

- **FwdListIter** (List< T > const &lst)
- T **operator*** () const
- FwdListIter & **operator++** ()
- bool **operator==** (FwdListIter< T > const &other)
- bool **operator!=** (FwdListIter< T > const &other)

7.51.1 Detailed Description

template<class T>
class FwdListIter< T >

Definition at line 117 of file functional_list.hpp.

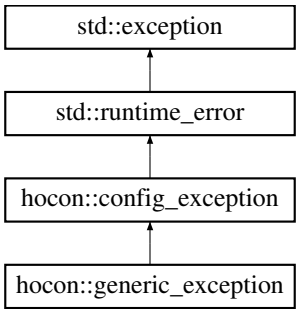
The documentation for this class was generated from the following file:

- hocon/functional_list.hpp

7.52 hocon::generic_exception Struct Reference

Exception that doesn't fall into any other category.

Inheritance diagram for hocon::generic_exception:



Public Member Functions

- **generic_exception** (std::string const &message)

7.52.1 Detailed Description

Exception that doesn't fall into any other category.

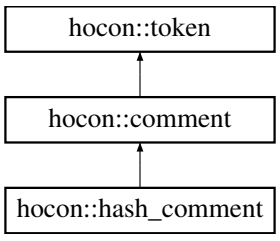
Definition at line 181 of file config_exception.hpp.

The documentation for this struct was generated from the following file:

- hocon/config_exception.hpp

7.53 hocon::hash_comment Class Reference

Inheritance diagram for hocon::hash_comment:



Public Member Functions

- **hash_comment** (shared_origin origin, std::string text)
- std::string **token_text** () const override
- std::string **text** () const
- std::string **to_string** () const override
- bool **operator==** (const token &other) const override
- virtual token_type **get_token_type** () const
- virtual shared_origin const & **origin** () const
- int **line_number** () const

7.53.1 Detailed Description

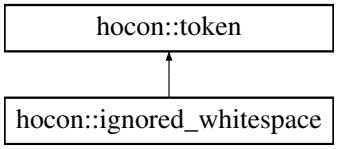
Definition at line 89 of file tokens.hpp.

The documentation for this class was generated from the following file:

- internal/tokens.hpp

7.54 hocon::ignored_whitespace Class Reference

Inheritance diagram for hocon::ignored_whitespace:



Public Member Functions

- **ignored_whitespace** (shared_origin origin, std::string whitespace)
- std::string **to_string** () const override
- bool **operator==** (const token &other) const override
- virtual token_type **get_token_type** () const
- virtual std::string **token_text** () const
- virtual shared_origin const & **origin** () const
- int **line_number** () const

7.54.1 Detailed Description

Definition at line 42 of file tokens.hpp.

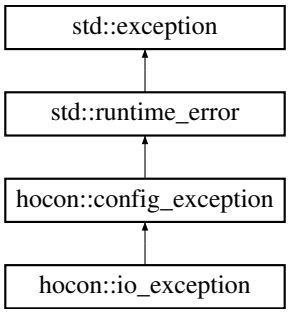
The documentation for this class was generated from the following file:

- internal/tokens.hpp

7.55 hocon::io_exception Struct Reference

Exception indicating that there was an IO error.

Inheritance diagram for hocon::io_exception:



Public Member Functions

- **io_exception** (config_origin const &origin, std::string const &message)

7.55.1 Detailed Description

Exception indicating that there was an IO error.

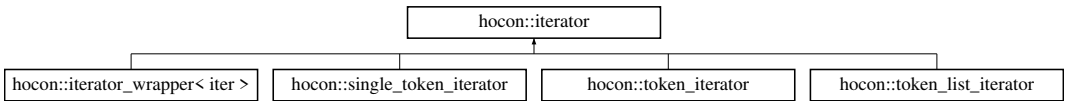
Definition at line 93 of file config_exception.hpp.

The documentation for this struct was generated from the following file:

- hocon/config_exception.hpp

7.56 hocon::iterator Class Reference

Inheritance diagram for hocon::iterator:



Public Member Functions

- virtual bool **has_next** ()=0
- virtual shared_token **next** ()=0

7.56.1 Detailed Description

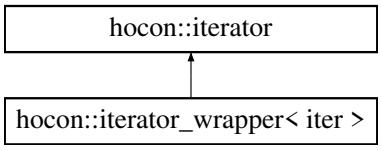
Definition at line 24 of file tokenizer.hpp.

The documentation for this class was generated from the following file:

- internal/tokenizer.hpp

7.57 hocon::iterator_wrapper< iter > Class Template Reference

Inheritance diagram for hocon::iterator_wrapper< iter >:



Public Member Functions

- **iterator_wrapper** (iter begin, iter end)
- bool **has_next** () override
- shared_token **next** () override

7.57.1 Detailed Description

template<typename iter>
class hocon::iterator_wrapper< iter >

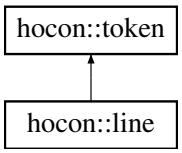
Definition at line 31 of file tokenizer.hpp.

The documentation for this class was generated from the following file:

- internal/tokenizer.hpp

7.58 hocon::line Class Reference

Inheritance diagram for hocon::line:



Public Member Functions

- **line** (shared_origin origin)
- std::string **to_string** () const override
- bool **operator==** (token const &other) const override
- virtual token_type **get_token_type** () const
- virtual std::string **token_text** () const
- virtual shared_origin const & **origin** () const
- int **line_number** () const

7.58.1 Detailed Description

Definition at line 24 of file tokens.hpp.

The documentation for this class was generated from the following file:

- internal/tokens.hpp

7.59 List< T > Class Template Reference

Public Member Functions

- **List** (T v, List const &tail)
- **List** (T v)
- bool **isEmpty** () const
- T **front** () const
- List **popped_front** () const
- List **pushed_front** (T v) const
- List **take** (int n)
- List **insertedAt** (int i, T v) const
- List **removed** (T v) const
- List **removed1** (T v) const
- bool **member** (T v) const
- template<class F >
  void **forEach** (F f) const
- int **headCount** () const

Friends

- class **FwdListIter**< **T** >

7.59.1 Detailed Description

template<**class T**>
class List< **T** >

Definition at line 13 of file functional_list.hpp.

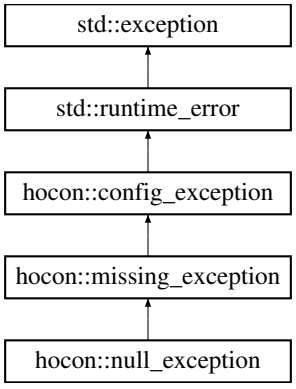
The documentation for this class was generated from the following file:

- hocon/functional_list.hpp

7.60 hocon::missing_exception Struct Reference

Exception indicates that the setting was never set to anything, not even null.

Inheritance diagram for hocon::missing_exception:



Public Member Functions

- **missing_exception** (std::string const &path)
- **config_exception** (config_origin const &origin, std::string const &message)
- **config_exception** (std::string const &message)
- **config_exception** (std::string const &message, std::exception const &e)

7.60.1 Detailed Description

Exception indicates that the setting was never set to anything, not even null.

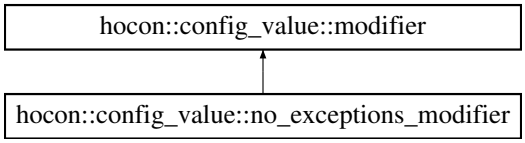
Definition at line 38 of file config_exception.hpp.

The documentation for this struct was generated from the following file:

- hocon/config_exception.hpp

7.61 hocon::config_value::modifier Class Reference

Inheritance diagram for hocon::config_value::modifier:



Public Member Functions

- virtual shared_value **modify_child_may_throw** (std::string const &key_or_null, shared_value v)=0

7.61.1 Detailed Description

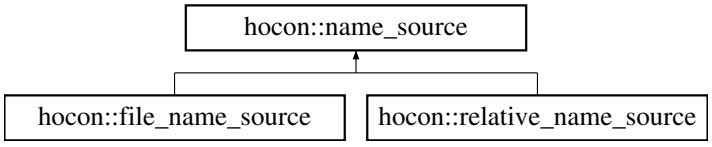
Definition at line 212 of file config_value.hpp.

The documentation for this class was generated from the following file:

- hocon/config_value.hpp

7.62 hocon::name_source Class Reference

Inheritance diagram for hocon::name_source:



Public Member Functions

- **name_source** (shared_include_context context)
- virtual shared_parseable **name_to_parseable** (std::string name, config_parse_options parse_options) const =0
- void **set_context** (shared_include_context context)
- shared_include_context **get_context** () const
- bool **context_initialized** () const

7.62.1 Detailed Description

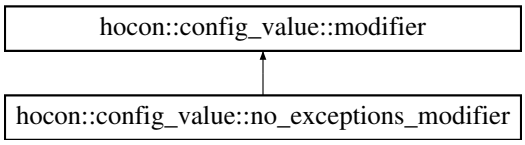
Definition at line 57 of file simple_includer.hpp.

The documentation for this class was generated from the following file:

- internal/simple_includer.hpp

7.63 hocon::config_value::no_exceptions_modifier Class Reference

Inheritance diagram for hocon::config_value::no_exceptions_modifier:



Public Member Functions

- **no_exceptions_modifier** (std::string prefix)
- shared_value **modify_child_may_throw** (std::string const &key_or_null, shared_value v) override
- shared_value **modify_child** (std::string const &key, shared_value v) const

7.63.1 Detailed Description

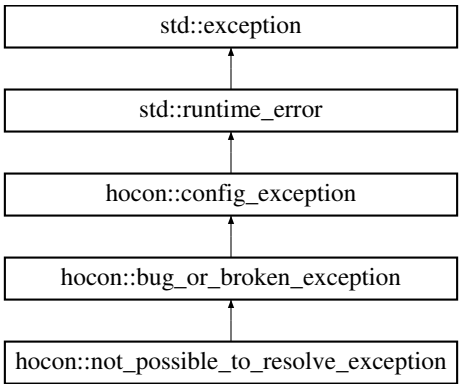
Definition at line 217 of file config_value.hpp.

The documentation for this class was generated from the following file:

- hocon/config_value.hpp

7.64 hocon::not_possible_to_resolve_exception Struct Reference

Inheritance diagram for hocon::not_possible_to_resolve_exception:



Public Member Functions

- **not_possible_to_resolve_exception** (std::string const &message)

7.64.1 Detailed Description

Definition at line 127 of file config_exception.hpp.

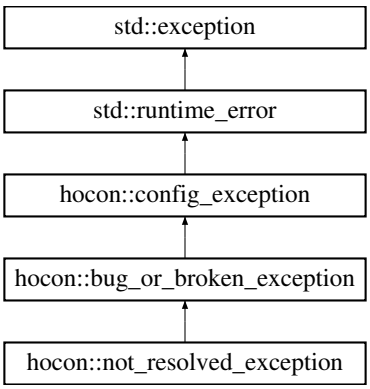
The documentation for this struct was generated from the following file:

- hocon/config_exception.hpp

7.65 hocon::not_resolved_exception Struct Reference

Exception indicating that you tried to use a function that requires substitutions to be resolved, but substitutions have not been resolved (that is, config#resolve was not called).

Inheritance diagram for hocon::not_resolved_exception:



Public Member Functions

- **not_resolved_exception** (std::string const &message)

7.65.1 Detailed Description

Exception indicating that you tried to use a function that requires substitutions to be resolved, but substitutions have not been resolved (that is, config#resolve was not called).

This is always a bug in either application code or the library; it's wrong to write a handler for this exception because you should be able to fix the code to avoid it by adding calls to config#resolve.

Definition at line 123 of file config_exception.hpp.

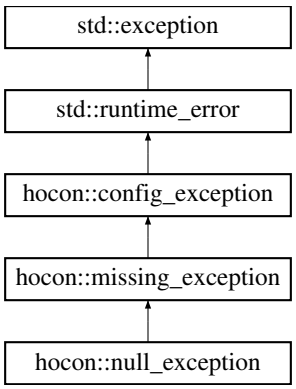
The documentation for this struct was generated from the following file:

- hocon/config_exception.hpp

7.66 hocon::null_exception Struct Reference

Exception indicates that the setting was treated as missing because it was set to null.

Inheritance diagram for hocon::null_exception:



Public Member Functions

- **null_exception** (config_origin const &origin, std::string const &path, std::string const &expected="")
- **config_exception** (config_origin const &origin, std::string const &message)
- **config_exception** (std::string const &message)
- **config_exception** (std::string const &message, std::exception const &e)

7.66.1 Detailed Description

Exception indicates that the setting was treated as missing because it was set to null.

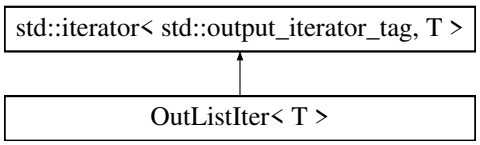
Definition at line 48 of file config_exception.hpp.

The documentation for this struct was generated from the following file:

- hocon/config_exception.hpp

7.67 OutListIter< T > Class Template Reference

Inheritance diagram for OutListIter< T >:



Public Member Functions

- T & **operator*** ()
- OutListIter & **operator++** ()
- List< T > **getList** () const

7.67.1 Detailed Description

template<class T>
class OutListIter< T >

Definition at line 142 of file functional_list.hpp.

The documentation for this class was generated from the following file:

- hocon/functional_list.hpp

7.68 hocon::config_document_parser::parse_context Class Reference

Public Member Functions

- **parse_context** (config_syntax flavor, shared_origin origin, token_iterator tokens)
- std::shared_ptr< config_node_root > **parse** ()
- shared_node_value parse_single_value ()

  Parse a given input stream into a single value node.

7.68.1 Detailed Description

Definition at line 22 of file config_document_parser.hpp.

7.68.2 Member Function Documentation

7.68.2.1 parse_single_value()

```
shared_node_value hocon::config_document_parser::parse_context::parse_single_value ( )
```

Parse a given input stream into a single value node.

Used when doing a replace inside a ConfigDocument.

The documentation for this class was generated from the following file:

- internal/config_document_parser.hpp

7.69 hocon::config_parser::parse_context Class Reference

Public Member Functions

- **parse_context** (config_syntax flavor, shared_origin origin, std::shared_ptr< const config_node_root > document, std::shared_ptr< const full_includer > includer, shared_include_context include_context)
- shared_value **parse** ()

Public Attributes

- int **array_count**

7.69.1 Detailed Description

Definition at line 27 of file config_parser.hpp.

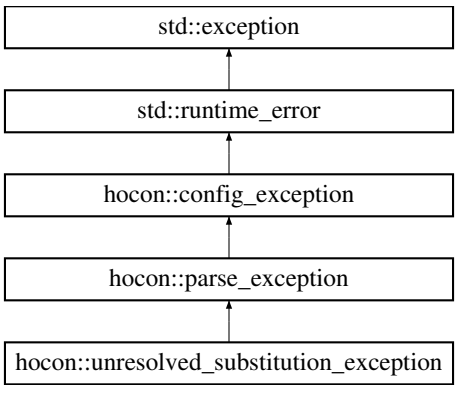
The documentation for this class was generated from the following file:

- internal/config_parser.hpp

7.70 hocon::parse_exception Struct Reference

Exception indicating that there was a parse error.

Inheritance diagram for hocon::parse_exception:



Public Member Functions

- **parse_exception** (config_origin const &origin, std::string const &message)

7.70.1 Detailed Description

Exception indicating that there was a parse error.

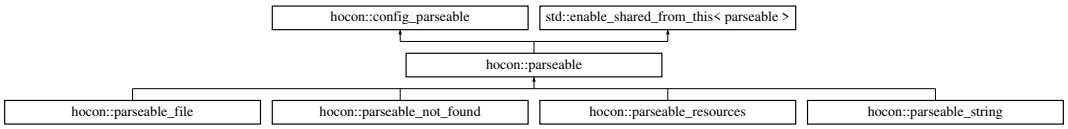
Definition at line 101 of file config_exception.hpp.

The documentation for this struct was generated from the following file:

- hocon/config_exception.hpp

7.71 hocon::parseable Class Reference

Inheritance diagram for hocon::parseable:



Public Member Functions

- void **post_construct** (config_parse_options const &base_options)
- std::shared_ptr< config_document > **parse_config_document** ()
- shared_object parse (config_parse_options const &options) const override

  Parse whatever it is.
- shared_object **parse** () const
- shared_value **parse_value** () const
- config_parse_options const & options () const override

  Get the initial options, which can be modified then passed to parse().
- std::shared_ptr< const config_origin > origin () const override

  Returns a config_origin describing the origin of the paresable 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

options	parse options, should be based on the ones from config_parseable#options()

Returns

the parsed object

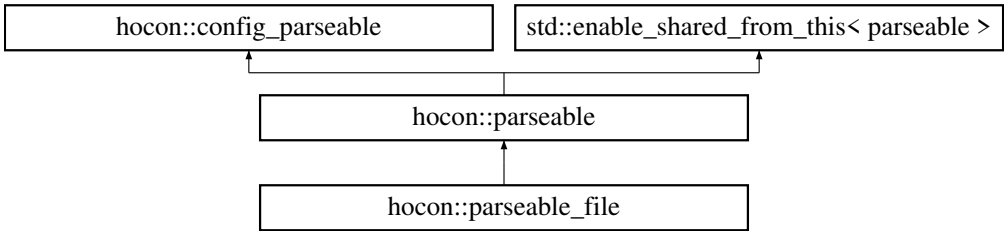
Implements hocon::config_parseable.

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 paresable item.
- virtual std::unique_ptr< std::istream > **reader** (config_parse_options const &options) const
- virtual config_syntax **content_type** () const
- virtual std::shared_ptr< config_parseable > **relative_to** (std::string file_name) const
- std::string **to_string** () const
- std::string **get_cur_dir** () const
- void **set_cur_dir** (std::string dir) const
- void **separate_filepath** (const std::string &path, std::string *file_dir, std::string *file_name) const

Static Public Member Functions

- static std::shared_ptr< parseable > **new_file** (std::string input_file_path, config_parse_options options)
- static std::shared_ptr< parseable > **new_string** (std::string s, config_parse_options options)
- static std::shared_ptr< parseable > **new_not_found** (std::string what_not_found, std::string message, config_parse_options options)
- static config_syntax **syntax_from_extension** (std::string name)

7.72.1 Detailed Description

Definition at line 79 of file parseable.hpp.

7.72.2 Member Function Documentation

7.72.2.1 options()

```
config_parse_options const& hocon::parseable::options ( ) const  [override], [virtual], [inherited]
```

Get the initial options, which can be modified then passed to parse().

These options will have the right description, includer, and other parameters already set up.

Returns

the initial options

Implements hocon::config_parseable.

7.72.2.2 origin()

```
std::shared_ptr<const config_origin> hocon::parseable::origin ( ) const  [override], [virtual],
[inherited]
```

Returns a config_origin describing the origin of the paresable item.

Implements hocon::config_parseable.

7.72.2.3 parse()

```
shared_object hocon::parseable::parse (
            config_parse_options const & options ) const  [override], [virtual], [inherited]
```

Parse whatever it is.

The options should come from config_parseable#options() but you could tweak them if you like.

Parameters

options	parse options, should be based on the ones from config_parseable#options()

Returns

the parsed object

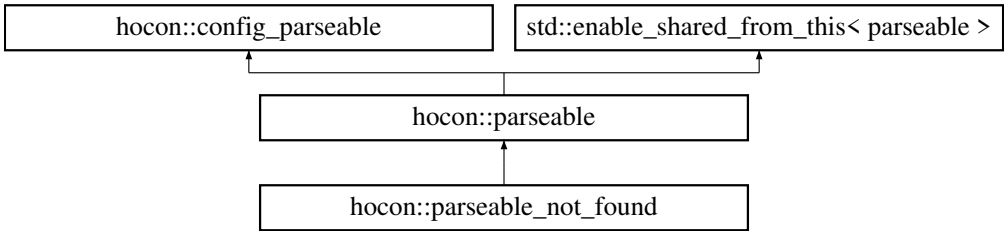
Implements hocon::config_parseable.

The documentation for this class was generated from the following file:

- internal/parseable.hpp

7.73 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 paresable item.
- virtual std::unique_ptr< std::istream > **reader** (config_parse_options const &options) const
- virtual config_syntax **guess_syntax** () const
- virtual config_syntax **content_type** () const
- virtual std::shared_ptr< config_parseable > **relative_to** (std::string file_name) const
- std::string **to_string** () const
- std::string **get_cur_dir** () const
- void **set_cur_dir** (std::string dir) const
- void **separate_filepath** (const std::string &path, std::string *file_dir, std::string *file_name) const

Static Public Member Functions

- static std::shared_ptr< parseable > **new_file** (std::string input_file_path, config_parse_options options)
- static std::shared_ptr< parseable > **new_string** (std::string s, config_parse_options options)
- static std::shared_ptr< parseable > **new_not_found** (std::string what_not_found, std::string message, config_parse_options options)
- static config_syntax **syntax_from_extension** (std::string name)

7.73.1 Detailed Description

Definition at line 120 of file parseable.hpp.

7.73.2 Member Function Documentation

7.73.2.1 options()

```
config_parse_options const& hocon::parseable::options ( ) const  [override], [virtual], [inherited]
```

Get the initial options, which can be modified then passed to parse().

These options will have the right description, includer, and other parameters already set up.

Returns

the initial options

Implements hocon::config_parseable.

7.73.2.2 origin()

```
std::shared_ptr<const config_origin> hocon::parseable::origin ( ) const  [override], [virtual],
[inherited]
```

Returns a config_origin describing the origin of the paresable item.

Implements hocon::config_parseable.

7.73.2.3 parse()

```
shared_object hocon::parseable::parse (
            config_parse_options const & options ) const  [override], [virtual], [inherited]
```

Parse whatever it is.

The options should come from config_parseable#options() but you could tweak them if you like.

Parameters

options	parse options, should be based on the ones from config_parseable#options()

Returns

the parsed object

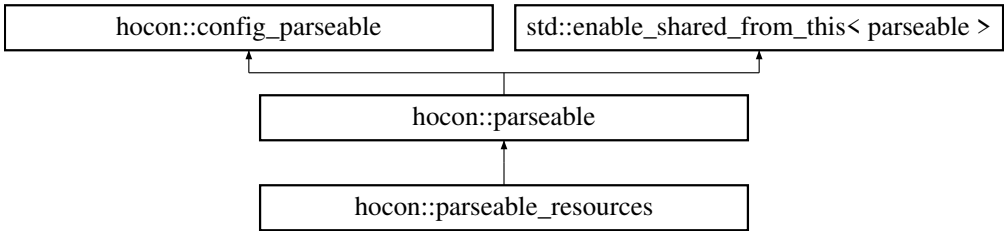
Implements hocon::config_parseable.

The documentation for this class was generated from the following file:

- internal/parseable.hpp

7.74 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 paresable item.
- virtual std::unique_ptr< std::istream > **reader** (config_parse_options const &options) const
- virtual config_syntax **guess_syntax** () const
- virtual config_syntax **content_type** () const
- virtual std::shared_ptr< config_parseable > **relative_to** (std::string file_name) const
- std::string **to_string** () const
- std::string **get_cur_dir** () const
- void **set_cur_dir** (std::string dir) const
- void **separate_filepath** (const std::string &path, std::string *file_dir, std::string *file_name) const

Static Public Member Functions

- static std::shared_ptr< parseable > **new_file** (std::string input_file_path, config_parse_options options)
- static std::shared_ptr< parseable > **new_string** (std::string s, config_parse_options options)
- static std::shared_ptr< parseable > **new_not_found** (std::string what_not_found, std::string message, config_parse_options options)
- static config_syntax **syntax_from_extension** (std::string name)

7.74.1 Detailed Description

Definition at line 107 of file parseable.hpp.

7.74.2 Member Function Documentation

7.74.2.1 options()

```
config_parse_options const& hocon::parseable::options ( ) const  [override], [virtual], [inherited]
```

Get the initial options, which can be modified then passed to parse().

These options will have the right description, includer, and other parameters already set up.

Returns

the initial options

Implements hocon::config_parseable.

7.74.2.2 origin()

```
std::shared_ptr<const config_origin> hocon::parseable::origin ( ) const  [override], [virtual],
[inherited]
```

Returns a config_origin describing the origin of the paresable item.

Implements hocon::config_parseable.

7.74.2.3 parse()

```
shared_object hocon::parseable::parse (
            config_parse_options const & options ) const  [override], [virtual], [inherited]
```

Parse whatever it is.

The options should come from config_parseable#options() but you could tweak them if you like.

Parameters

options	parse options, should be based on the ones from config_parseable#options()

Returns

the parsed object

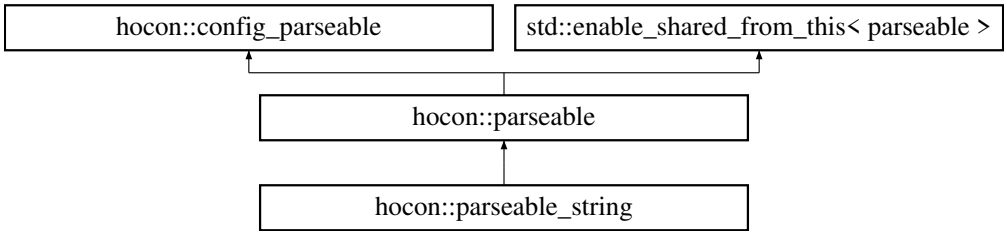
Implements hocon::config_parseable.

The documentation for this class was generated from the following file:

- internal/parseable.hpp

7.75 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 paresable item.
- virtual std::unique_ptr< std::istream > **reader** (config_parse_options const &options) const
- virtual config_syntax **guess_syntax** () const
- virtual config_syntax **content_type** () const
- virtual std::shared_ptr< config_parseable > **relative_to** (std::string file_name) const
- std::string **to_string** () const
- std::string **get_cur_dir** () const
- void **set_cur_dir** (std::string dir) const
- void **separate_filepath** (const std::string &path, std::string *file_dir, std::string *file_name) const

Static Public Member Functions

- static std::shared_ptr< parseable > **new_file** (std::string input_file_path, config_parse_options options)
- static std::shared_ptr< parseable > **new_string** (std::string s, config_parse_options options)
- static std::shared_ptr< parseable > **new_not_found** (std::string what_not_found, std::string message, config_parse_options options)
- static config_syntax **syntax_from_extension** (std::string name)

7.75.1 Detailed Description

Definition at line 90 of file parseable.hpp.

7.75.2 Member Function Documentation

7.75.2.1 options()

```
config_parse_options const& hocon::parseable::options ( ) const  [override], [virtual], [inherited]
```

Get the initial options, which can be modified then passed to parse().

These options will have the right description, includer, and other parameters already set up.

Returns

the initial options

Implements hocon::config_parseable.

7.75.2.2 origin()

```
std::shared_ptr<const config_origin> hocon::parseable::origin ( ) const  [override], [virtual],
[inherited]
```

Returns a config_origin describing the origin of the paresable item.

Implements hocon::config_parseable.

7.75.2.3 parse()

```
shared_object hocon::parseable::parse (
            config_parse_options const & options ) const  [override], [virtual], [inherited]
```

Parse whatever it is.

The options should come from config_parseable#options() but you could tweak them if you like.

Parameters

options	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

7.76 hocon::path Class Reference

Public Member Functions

- **path** (std::string first, path const &remainder)
- **path** (std::vector< std::string > elements)
- **path** (std::vector< path > paths_to_concat)
- shared_string **first** () const
- path remainder () const

  Returns the path minus the first element, or null if no more elements.
- path parent () const

  Returns the path minus the last element or null if we have just one element.
- bool **has_remainder** () const
- bool **empty** () const
- shared_string **last** () const
- path **prepend** (path prefix)
- int **length** () const
- path **sub_path** (int remove_from_front)
- path **sub_path** (int start_index, int end_index)
- bool **starts_with** (path other) const
- bool **operator==** (path const &other) const
- bool **operator!=** (path const &other) const
- void **append_to_string** (std::string &base) const
- std::string to_string () const

  For debugging.
- std::string render () const

  Human-readable error-message-oriented string representation of path.

Static Public Member Functions

- static bool has_funky_chars (std::string const &s)

  Signals whether quotes and other noise need to be removed/ignored.
- static path **new_key** (std::string key)
- static path **new_path** (std::string path)

7.76.1 Detailed Description

Definition at line 13 of file path.hpp.

7.76.2 Member Function Documentation

7.76.2.1 has_funky_chars()

```
static bool hocon::path::has_funky_chars (
            std::string const & s )  [static]
```

Signals whether quotes and other noise need to be removed/ignored.

7.76.2.2 parent()

```
path hocon::path::parent ( ) const
```

Returns the path minus the last element or null if we have just one element.

7.76.2.3 remainder()

```
path hocon::path::remainder ( ) const
```

Returns the path minus the first element, or null if no more elements.

7.76.2.4 render()

```
std::string hocon::path::render ( ) const
```

Human-readable error-message-oriented string representation of path.

7.76.2.5 to_string()

```
std::string hocon::path::to_string ( ) const
```

For debugging.

The documentation for this class was generated from the following file:

- hocon/path.hpp

7.77 hocon::path_builder Class Reference

Public Member Functions

- void **append_key** (std::string key)
- void **append_path** (path path_to_append)
- path result ()

  Returns null if keys is empty.

7.77.1 Detailed Description

Definition at line 8 of file path_builder.hpp.

7.77.2 Member Function Documentation

7.77.2.1 result()

```
path hocon::path_builder::result ( )
```

Returns null if keys is empty.

The documentation for this class was generated from the following file:

- internal/path_builder.hpp

7.78 hocon::path_parser Class Reference

Static Public Member Functions

- static config_node_path **parse_path_node** (std::string const &path_string, config_syntax flavor=config_↩ syntax::CONF)
- static path **parse_path** (std::string const &path_string)
- static path **parse_path_expression** (iterator &expression, shared_origin origin, std::string const &original↩ _text="", token_list *path_tokens=nullptr, config_syntax flavor=config_syntax::CONF)
- static config_node_path **parse_path_node_expression** (iterator &expression, shared_origin origin, std↩ ::string const &original_text="", config_syntax flavor=config_syntax::CONF)

7.78.1 Detailed Description

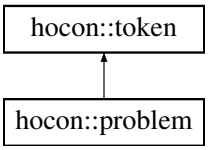
Definition at line 14 of file path_parser.hpp.

The documentation for this class was generated from the following file:

- internal/path_parser.hpp

7.79 hocon::problem Class Reference

Inheritance diagram for hocon::problem:



Public Member Functions

- **problem** (shared_origin origin, std::string what, std::string message, bool suggest_quotes)
- std::string **what** () const
- std::string **message** () const
- bool **suggest_quotes** () const
- std::string **to_string** () const override
- bool **operator==** (const token &other) const override
- virtual token_type **get_token_type** () const
- virtual std::string **token_text** () const
- virtual shared_origin const & **origin** () const
- int **line_number** () const

7.79.1 Detailed Description

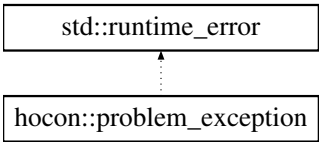
Definition at line 51 of file tokens.hpp.

The documentation for this class was generated from the following file:

- internal/tokens.hpp

7.80 hocon::problem_exception Class Reference

Inheritance diagram for hocon::problem_exception:



Public Member Functions

- **problem_exception** (problem prob)
- problem const & **get_problem** () const

7.80.1 Detailed Description

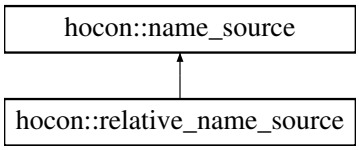
Definition at line 15 of file tokenizer.hpp.

The documentation for this class was generated from the following file:

- internal/tokenizer.hpp

7.81 hocon::relative_name_source Class Reference

Inheritance diagram for hocon::relative_name_source:



Public Member Functions

- **relative_name_source** (shared_include_context context)
- shared_parseable **name_to_parseable** (std::string name, config_parse_options parse_options) const override
- void **set_context** (shared_include_context context)
- shared_include_context **get_context** () const
- bool **context_initialized** () const

7.81.1 Detailed Description

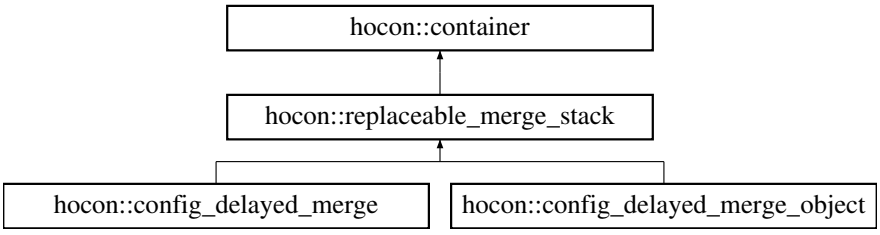
Definition at line 90 of file simple_includer.hpp.

The documentation for this class was generated from the following file:

- internal/simple_includer.hpp

7.82 hocon::replaceable_merge_stack Class Reference

Inheritance diagram for hocon::replaceable_merge_stack:



Public Member Functions

- virtual shared_value **make_replacement** (resolve_context const &context, int skipping) const =0
- virtual shared_value replace_child (shared_value const &child, shared_value replacement) const =0

  Replace a child of this value.
- virtual bool has_descendant (shared_value const &descendant) const =0

  Super-expensive full traversal to see if descendant is anywhere underneath this container.

7.82.1 Detailed Description

Definition at line 9 of file replaceable_merge_stack.hpp.

7.82.2 Member Function Documentation

7.82.2.1 has_descendant()

```
virtual bool hocon::container::has_descendant (
            shared_value const & descendant ) const  [pure virtual], [inherited]
```

Super-expensive full traversal to see if descendant is anywhere underneath this container.

Implemented in hocon::simple_config_object, hocon::simple_config_list, hocon::config_delayed_merge_object, hocon::config_delayed_merge, and hocon::config_concatenation.

7.82.2.2 replace_child()

```
virtual shared_value hocon::container::replace_child (
            shared_value const & child,
            shared_value replacement ) const  [pure virtual], [inherited]
```

Replace a child of this value.

CAUTION if replacement is null, delete the child, which may also delete the parent, or make the parent into a non-container.

Implemented in hocon::simple_config_object, hocon::simple_config_list, hocon::config_delayed_merge_object, hocon::config_delayed_merge, and hocon::config_concatenation.

The documentation for this class was generated from the following file:

- internal/replaceable_merge_stack.hpp

7.83 hocon::resolve_context Class Reference

Public Member Functions

- **resolve_context** (config_resolve_options options, path restrict_to_child, std::vector< shared_value > cycle_markers)
- **resolve_context** (config_resolve_options options, path restrict_to_child)
- bool **is_restricted_to_child** () const
- config_resolve_options **options** () const
- resolve_result< shared_value > **resolve** (shared_value original, resolve_source const &source) const
- path **restrict_to_child** () const
- resolve_context **add_cycle_marker** (shared_value value) const
- resolve_context **remove_cycle_marker** (shared_value value)
- resolve_context **restrict** (path restrict_to) const
- resolve_context **unrestricted** () const

Static Public Member Functions

- static shared_value **resolve** (shared_value value, shared_object root, config_resolve_options options)

7.83.1 Detailed Description

Definition at line 16 of file resolve_context.hpp.

The documentation for this class was generated from the following file:

- internal/resolve_context.hpp

7.84 hocon::resolve_result< V > Struct Template Reference

Public Member Functions

- **resolve_result** (resolve_context c, V v)

Public Attributes

- resolve_context **context**
- V **value**

7.84.1 Detailed Description

template<typename V>
struct hocon::resolve_result< V >

Definition at line 8 of file resolve_result.hpp.

The documentation for this struct was generated from the following files:

- hocon/config_value.hpp
- internal/resolve_result.hpp

7.85 hocon::resolve_source Class Reference

Classes

- struct result_with_path

Public Types

- typedef std::list< std::shared_ptr< const container > > **node**

Public Member Functions

- **resolve_source** (shared_object root)
- **resolve_source** (shared_object root, node path_from_root)
- resolve_source **push_parent** (std::shared_ptr< const container > parent) const
- result_with_path **lookup_subst** (resolve_context context, std::shared_ptr< substitution_expression > subst, int prefix_length) const
- resolve_source **replace_current_parent** (std::shared_ptr< const container > old, std::shared_ptr< const container > replacement) const
- resolve_source **replace_within_current_parent** (shared_value old, shared_value replacement) const
- resolve_source **reset_parents** () const

7.85.1 Detailed Description

Definition at line 15 of file resolve_source.hpp.

The documentation for this class was generated from the following file:

- internal/resolve_source.hpp

7.86 hocon::resolve_source::result_with_path Struct Reference

Public Member Functions

- **result_with_path** (resolve_result< shared_value > result_value, node path_from_root_value)

Public Attributes

- resolve_result< shared_value > **result**
- node **path_from_root**

7.86.1 Detailed Description

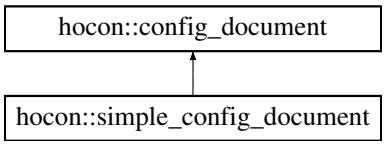
Definition at line 19 of file resolve_source.hpp.

The documentation for this struct was generated from the following file:

- internal/resolve_source.hpp

7.87 hocon::simple_config_document Class Reference

Inheritance diagram for hocon::simple_config_document:



Public Member Functions

- **simple_config_document** (std::shared_ptr< const config_node_root > root, config_parse_options opts)
- std::unique_ptr< config_document > with_value_text (std::string path, std::string new_value) const override

  Returns a new config_document that is a copy of the current config_document, but with the desired value set at the desired path.
- std::unique_ptr< config_document > with_value (std::string path, std::shared_ptr< config_value > new_value) const override

  Returns a new config_document that is a copy of the current config_document, but with the desired value set at the desired path.
- std::unique_ptr< config_document > without_path (std::string path) const override

  Returns a new config_document that is a copy of the current config_document, but with all values at the desired path removed.
- bool has_path (std::string const &path) const override

  Returns a boolean indicating whether or not a config_document has a value at the desired path.
- std::string render () const override

  The original text of the input, modified if necessary with any replaced or added values.

7.87.1 Detailed Description

Definition at line 10 of file simple_config_document.hpp.

7.87.2 Member Function Documentation

7.87.2.1 has_path()

```
bool hocon::simple_config_document::has_path (
            std::string const & path ) const  [override], [virtual]
```

Returns a boolean indicating whether or not a config_document has a value at the desired path.

null counts as a value for purposes of this check.

Parameters

path	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

path	the path at which to set the desired value
new_value	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

path	the path at which to set the desired value
new_value	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

path	the path to remove from the document

Returns

a copy of the config_document with the desired value removed from the document.

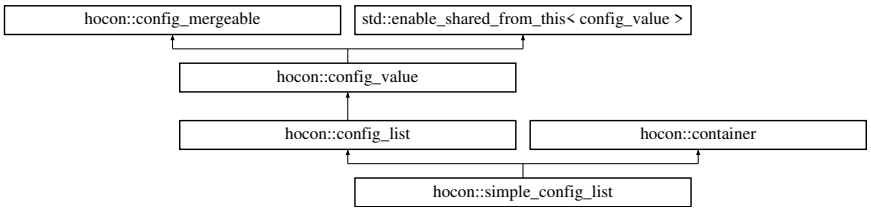
Implements hocon::config_document.

The documentation for this class was generated from the following file:

- internal/simple_config_document.hpp

7.88 hocon::simple_config_list Class Reference

Inheritance diagram for hocon::simple_config_list:



Public Types

- using **iterator** = std::vector< shared_value >::const_iterator
- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

  The type of a configuration value (following the JSON type schema).

Public Member Functions

- **simple_config_list** (shared_origin origin, std::vector< shared_value > value)
- **simple_config_list** (shared_origin origin, std::vector< shared_value > value, resolve_status status)
- config_value::type value_type () const override

  The type of the value; matches the JSON type schema.
- resolve_status **get_resolve_status** () const override
- shared_value replace_child (shared_value const &child, shared_value replacement) const override

  Replace a child of this value.
- bool has_descendant (shared_value const &descendant) const override

  Super-expensive full traversal to see if descendant is anywhere underneath this container.
- shared_value relativized (const std::string prefix) const override

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.
- bool **contains** (shared_value v) const
- bool **contains_all** (std::vector< shared_value >) const
- int **index_of** (shared_value v)
- bool **is_empty** () const override
- size_t **size** () const override
- shared_value **operator[ ]** (size_t index) const override
- shared_value **get** (size_t index) const override
- iterator **begin** () const override
- iterator **end** () const override
- std::shared_ptr< const simple_config_list > **concatenate** (std::shared_ptr< const simple_config_list > other) const
- unwrapped_value **unwrapped** () const override
- bool **operator==** (config_value const &other) const override
- virtual shared_origin const & origin () const

  The origin of the value (file, line number, etc.), for debugging and error messages.
- char const * value_type_name () const

  The printable name of the value type.

- virtual std::string render () const

  Renders the config value as a HOCON string.
- virtual std::string render (config_render_options options) const

  Renders the config value to a string, using the provided options.
- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.
- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.
- virtual shared_value with_origin (shared_origin origin) const

  Returns a.
- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

  Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.
- virtual std::string **transform_to_string** () const

Static Public Member Functions

- static char const * **type_name** (type t)

Protected Member Functions

- resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const override
- shared_value **new_copy** (shared_origin origin) const override
- void **render** (std::string &result, int indent, bool at_root, config_render_options options) const override
- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- void **require_not_ignoring_fallbacks** () const
- virtual bool **ignores_fallbacks** () const
- virtual shared_value **with_fallbacks_ignored** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- virtual shared_value **merged_with_object** (shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- virtual shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

7.88.1 Detailed Description

Definition at line 15 of file simple_config_list.hpp.

7.88.2 Member Enumeration Documentation

7.88.2.1 type

```
enum hocon::config_value::type  [strong], [inherited]
```

The type of a configuration value (following the JSON type schema).

Definition at line 56 of file config_value.hpp.

7.88.3 Member Function Documentation

7.88.3.1 at_key()

```
shared_config hocon::config_value::at_key (
            std::string const & key ) const  [inherited]
```

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	key to store this value at.

Returns

a
config
instance containing this value at the given key.

7.88.3.2 at_path()

```
shared_config hocon::config_value::at_path (
            std::string const & path_expression ) const  [inherited]
```

Places the value inside a config at the given path.

See also config_value#at_key(String).

Parameters

path	path to store this value at.

Returns

a
`config`
instance containing this value at the given path.

7.88.3.3 has_descendant()

```
bool hocon::simple_config_list::has_descendant (
            shared_value const & descendant ) const  [override], [virtual]
```

Super-expensive full traversal to see if descendant is anywhere underneath this container.

Implements hocon::container.

7.88.3.4 origin()

```
virtual shared_origin const& hocon::config_value::origin ( ) const  [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

Returns

where the value came from

7.88.3.5 relativized()

```
shared_value hocon::simple_config_list::relativized (
            const std::string prefix ) const  [override], [virtual]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at foo.bar in the parent, and the included file as a substitution ${a.b.c}, the included substitution now needs to be ${foo.bar.a.b.c} because we resolve substitutions globally only after parsing everything.

Parameters

prefix	

Returns

value relativized to the given path or the same value if nothing to do

Reimplemented from hocon::config_value.

7.88.3.6 render() [1/2]

```
virtual std::string hocon::config_value::render ( ) const  [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see config#resolve), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to
`render(config_render_options())`

.

Returns

the rendered value

7.88.3.7 render() [2/2]

```
virtual std::string hocon::config_value::render (
            config_render_options options ) const  [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see config#resolve), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

Parameters

options	the rendering options

Returns

the rendered value

7.88.3.8 replace_child()

```
shared_value hocon::simple_config_list::replace_child (
            shared_value const & child,
            shared_value replacement ) const  [override], [virtual]
```

Replace a child of this value.

CAUTION if replacement is null, delete the child, which may also delete the parent, or make the parent into a non-container.

Implements hocon::container.

7.88.3.9 to_fallback_value()

```
shared_value hocon::config_value::to_fallback_value ( ) const  [override], [protected], [virtual],
[inherited]
```

Converts a config to its root object and a config_value to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements hocon::config_mergeable.

7.88.3.10 value_type()

```
config_value::type hocon::simple_config_list::value_type ( ) const  [inline], [override],
[virtual]
```

The type of the value; matches the JSON type schema.

Returns

value's type

Implements hocon::config_value.

Definition at line 20 of file simple_config_list.hpp.

7.88.3.11 value_type_name()

```
char const* hocon::config_value::value_type_name ( ) const  [inline], [inherited]
```

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.88.3.12 with_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
            std::shared_ptr< const config_mergeable > other ) const  [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the spec for HOCON. Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

a new object (or the original one, if the fallback doesn't get used)

Implements hocon::config_mergeable.

7.88.3.13 with_origin()

```
virtual shared_value hocon::config_value::with_origin (
            shared_origin origin ) const  [virtual], [inherited]
```

Returns a.
config_value

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single config_value.

Parameters

origin	the origin set on the returned value

Returns

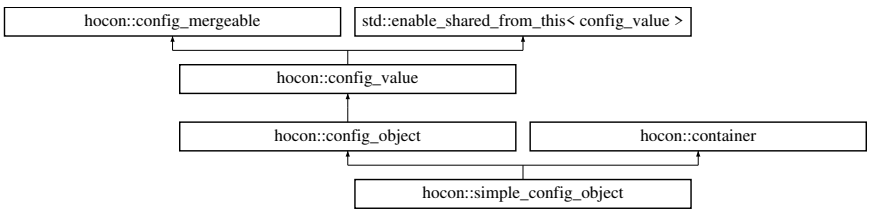
the new config_value with the given origin

The documentation for this class was generated from the following file:

- internal/values/simple_config_list.hpp

7.89 hocon::simple_config_object Class Reference

Inheritance diagram for hocon::simple_config_object:



Public Types

- using **iterator** = std::unordered_map< std::string, shared_value >::const_iterator
- enum class type {
  OBJECT , **LIST** , **NUMBER** , **BOOLEAN** ,
  CONFIG_NULL , **STRING** , **UNSPECIFIED** }

  The type of a configuration value (following the JSON type schema).

Public Member Functions

- **simple_config_object** (shared_origin origin, std::unordered_map< std::string, shared_value > value, resolve_status status, bool ignores_fallbacks)
- **simple_config_object** (shared_origin origin, std::unordered_map< std::string, shared_value > value)
- shared_value attempt_peek_with_partial_resolve (std::string const &key) const override

  Look up the key on an only-partially-resolved object, with no transformation or type conversion of any kind; if 'this' is not resolved then try to look up the key anyway if possible.
- bool **is_empty** () const override
- size_t **size** () const override
- shared_value **operator[ ]** (std::string const &key) const override
- iterator **begin** () const override
- iterator **end** () const override
- unwrapped_value **unwrapped** () const override
- shared_value **get** (std::string const &key) const override
- std::unordered_map< std::string, shared_value > const & **entry_set** () const override
- resolve_status **get_resolve_status** () const override
- bool **ignores_fallbacks** () const override
- shared_value **with_fallbacks_ignored** () const override
- shared_value **merged_with_object** (shared_object fallback) const override
- shared_object **with_value** (path raw_path, shared_value value) const override
- shared_object **with_value** (std::string key, shared_value value) const override
- shared_object **without_path** (path raw_path) const override
- shared_object **with_only_path** (path raw_path) const override
- shared_object with_only_path_or_null (path raw_path) const override

  Gets the object with only the path if the path exists, otherwise null if it doesn't.
- shared_value replace_child (shared_value const &child, shared_value replacement) const override

  Replace a child of this value.
- bool has_descendant (shared_value const &descendant) const override

  Super-expensive full traversal to see if descendant is anywhere underneath this container.
- std::vector< std::string > key_set () const override

  Construct a list of keys in the _value map.
- std::vector< shared_value > value_set (std::unordered_map< std::string, shared_value > m) const

  Construct a list of the values from the provided map.
- bool **operator==** (config_value const &other) const override
- virtual std::shared_ptr< const config > to_config () const

  Converts this object to a Config instance, enabling you to use path expressions to find values in the object.
- config_value::type value_type () const override

  The type of the value; matches the JSON type schema.
- virtual shared_origin const & origin () const

  The origin of the value (file, line number, etc.), for debugging and error messages.
- char const * value_type_name () const

  The printable name of the value type.
- virtual std::string render () const

Renders the config value as a HOCON string.

- virtual std::string render (config_render_options options) const

  Renders the config value to a string, using the provided options.

- shared_config at_key (std::string const &key) const

  Places the value inside a config at the given key.

- shared_config at_path (std::string const &path_expression) const

  Places the value inside a config at the given path.

- virtual shared_value with_origin (shared_origin origin) const

  Returns a.

- virtual shared_value relativized (std::string prefix) const

  This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

- std::shared_ptr< const config_mergeable > with_fallback (std::shared_ptr< const config_mergeable > other) const override

  Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

- virtual std::string **transform_to_string** () const

Static Public Member Functions

- static std::shared_ptr< simple_config_object > **empty** ()
- static std::shared_ptr< simple_config_object > **empty** (shared_origin origin)
- static std::shared_ptr< simple_config_object > **empty_instance** ()
- static char const * **type_name** (type t)

Protected Member Functions

- resolve_result< shared_value > **resolve_substitutions** (resolve_context const &context, resolve_source const &source) const override
- shared_value **new_copy** (shared_origin) const override
- void **render** (std::string &s, int indent, bool at_root, config_render_options options) const override
- shared_value **peek_path** (path desired_path) const
- shared_value **peek_assuming_resolved** (std::string const &key, path original_path) const
- shared_value **construct_delayed_merge** (shared_origin origin, std::vector< shared_value > stack) const override
- virtual void **render** (std::string &result, int indent, bool at_root, std::string const &at_key, config_render_options options) const
- shared_config **at_key** (shared_origin origin, std::string const &key) const
- shared_config **at_path** (shared_origin origin, path raw_path) const
- void **require_not_ignoring_fallbacks** () const
- shared_value **merged_with_the_unmergeable** (std::vector< shared_value > stack, std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_the_unmergeable** (std::shared_ptr< const unmergeable > fallback) const
- shared_value **merged_with_object** (std::vector< shared_value > stack, shared_object fallback) const
- shared_value **merged_with_non_object** (std::vector< shared_value > stack, shared_value fallback) const
- shared_value **merged_with_non_object** (shared_value fallback) const
- shared_value to_fallback_value () const override

  Converts a config to its root object and a config_value to itself.

Static Protected Member Functions

- static shared_value **peek_path** (const config_object *self, path desired_path)
- static shared_origin **merge_origins** (std::vector< shared_value > const &stack)
- static void **indent** (std::string &result, int indent, config_render_options const &options)
- static std::vector< shared_value > **replace_child_in_list** (std::vector< shared_value > const &values, shared_value const &child, shared_value replacement)
- static bool **has_descendant_in_list** (std::vector< shared_value > const &values, shared_value const &descendant)
- template<typename T >
  static bool **equals** (config_value const &other, std::function< bool(T const &)> checker)

7.89.1 Detailed Description

Definition at line 11 of file simple_config_object.hpp.

7.89.2 Member Enumeration Documentation

7.89.2.1 type

```
enum hocon::config_value::type  [strong], [inherited]
```

The type of a configuration value (following the JSON type schema).

Definition at line 56 of file config_value.hpp.

7.89.3 Member Function Documentation

7.89.3.1 at_key()

```
shared_config hocon::config_value::at_key (
            std::string const & key ) const  [inherited]
```

Places the value inside a config at the given key.

See also config_value#at_path(string).

Parameters

key	key to store this value at.

Returns

a
`config`
instance containing this value at the given key.

7.89.3.2 at_path()

```
shared_config hocon::config_value::at_path (
            std::string const & path_expression ) const  [inherited]
```

Places the value inside a config at the given path.

See also config_value#at_key(String).

Parameters

path	path to store this value at.

Returns

a
`config`
instance containing this value at the given path.

7.89.3.3 attempt_peek_with_partial_resolve()

```
shared_value hocon::simple_config_object::attempt_peek_with_partial_resolve (
            std::string const & key ) const  [override], [virtual]
```

Look up the key on an only-partially-resolved object, with no transformation or type conversion of any kind; if 'this' is not resolved then try to look up the key anyway if possible.

Parameters

key	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.

7.89.3.4 has_descendant()

```
bool hocon::simple_config_object::has_descendant (
            shared_value const & descendant ) const  [override], [virtual]
```

Super-expensive full traversal to see if descendant is anywhere underneath this container.

Implements hocon::container.

7.89.3.5 key_set()

```
std::vector<std::string> hocon::simple_config_object::key_set ( ) const  [override], [virtual]
```

Construct a list of keys in the _value map.

Use a vector rather than set, because most of the time we just want to iterate over them.

Implements hocon::config_object.

7.89.3.6 origin()

```
virtual shared_origin const& hocon::config_value::origin ( ) const  [virtual], [inherited]
```

The origin of the value (file, line number, etc.), for debugging and error messages.

Returns

where the value came from

7.89.3.7 relativized()

```
virtual shared_value hocon::config_value::relativized (
            std::string prefix ) const  [inline], [virtual], [inherited]
```

This is used when including one file in another; the included file is relativized to the path it's included into in the parent file.

The point is that if you include a file at foo.bar in the parent, and the included file as a substitution ${a.b.c}, the included substitution now needs to be ${foo.bar.a.b.c} because we resolve substitutions globally only after parsing everything.

Parameters

prefix	

Returns

value relativized to the given path or the same value if nothing to do

Reimplemented in hocon::config_concatenation, and hocon::simple_config_list.

Definition at line 181 of file config_value.hpp.

7.89.3.8 render() [1/2]

```
virtual std::string hocon::config_value::render ( ) const  [virtual], [inherited]
```

Renders the config value as a HOCON string.

This method is primarily intended for debugging, so it tries to add helpful comments and whitespace.

If the config value has not been resolved (see config#resolve), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

This method is equivalent to
`render(config_render_options())`

.

Returns

the rendered value

7.89.3.9 render() [2/2]

```
virtual std::string hocon::config_value::render (
            config_render_options options ) const  [virtual], [inherited]
```

Renders the config value to a string, using the provided options.

If the config value has not been resolved (see config#resolve), it's possible that it can't be rendered as valid HOCON. In that case the rendering should still be useful for debugging but you might not be able to parse it. If the value has been resolved, it will always be parseable.

If the config value has been resolved and the options disable all HOCON-specific features (such as comments), the rendering will be valid JSON. If you enable HOCON-only features such as comments, the rendering will not be valid JSON.

Parameters

options	the rendering options

Returns

the rendered value

7.89.3.10 replace_child()

```
shared_value hocon::simple_config_object::replace_child (
            shared_value const & child,
            shared_value replacement ) const  [override], [virtual]
```

Replace a child of this value.

CAUTION if replacement is null, delete the child, which may also delete the parent, or make the parent into a non-container.

Implements hocon::container.

7.89.3.11 to_config()

```
virtual std::shared_ptr<const config> hocon::config_object::to_config ( ) const  [virtual],
[inherited]
```

Converts this object to a Config instance, enabling you to use path expressions to find values in the object.

This is a constant-time operation (it is not proportional to the size of the object).

Returns

a Config with this object as its root

7.89.3.12 to_fallback_value()

```
shared_value hocon::config_value::to_fallback_value ( ) const  [override], [protected], [virtual],
[inherited]
```

Converts a config to its root object and a config_value to itself.

Originally in the MergeableValue interface, squashing to ease C++ public API separation

Implements hocon::config_mergeable.

7.89.3.13 value_set()

```
std::vector<shared_value> hocon::simple_config_object::value_set (
            std::unordered_map< std::string, shared_value > m ) const
```

Construct a list of the values from the provided map.

Equivalent to Java's Collection.values() method.

7.89.3.14 value_type()

```
config_value::type hocon::config_object::value_type ( ) const  [override], [virtual], [inherited]
```

The type of the value; matches the JSON type schema.

Returns

value's type

Implements hocon::config_value.

7.89.3.15 value_type_name()

```
char const* hocon::config_value::value_type_name ( ) const  [inline], [inherited]
```

The printable name of the value type.

Returns

value's type's name

Definition at line 92 of file config_value.hpp.

7.89.3.16 with_fallback()

```
std::shared_ptr<const config_mergeable> hocon::config_value::with_fallback (
            std::shared_ptr< const config_mergeable > other ) const  [override], [virtual],
[inherited]
```

Returns a new value computed by merging this value with another, with keys in this value "winning" over the other one.

This associative operation may be used to combine configurations from multiple sources (such as multiple configuration files).

The semantics of merging are described in the spec for HOCON. Merging typically occurs when either the same object is created twice in the same file, or two config files are both loaded. For example:

```
foo = { a: 42 }
foo = { b: 43 }
```

Here, the two objects are merged as if you had written:

```
foo = { a: 42, b: 43 }
```

Only ConfigObject and Config instances do anything in this method (they need to merge the fallback keys into themselves). All other values just return the original value, since they automatically override any fallback. This means that objects do not merge "across" non-objects; if you write `object.withFallback(nonObject).withFallback(otherObject)`, then `otherObject` will simply be ignored. This is an intentional part of how merging works, because non-objects such as strings and integers replace (rather than merging with) any prior value:

```
foo = { a: 42 }
foo = 10
```

Here, the number 10 "wins" and the value of `foo` would be simply 10. Again, for details see the spec.

Parameters

other	an object whose keys should be used as fallbacks, if the keys are not present in this one

Returns

a new object (or the original one, if the fallback doesn't get used)

Implements hocon::config_mergeable.

7.89.3.17 with_only_path_or_null()

```
shared_object hocon::simple_config_object::with_only_path_or_null (
            path raw_path ) const  [override], [virtual]
```

Gets the object with only the path if the path exists, otherwise null if it doesn't.

this ensures that if we have { a : { b : 42 } } and do withOnlyPath("a.b.c") that we don't keep an empty "a" object.

Implements hocon::config_object.

7.89.3.18 with_origin()

```
virtual shared_value hocon::config_value::with_origin (
            shared_origin origin ) const  [virtual], [inherited]
```

Returns a.
config_value

based on this one, but with the given origin. This is useful when you are parsing a new format of file or setting comments for a single config_value.

Parameters

origin	the origin set on the returned value

Returns

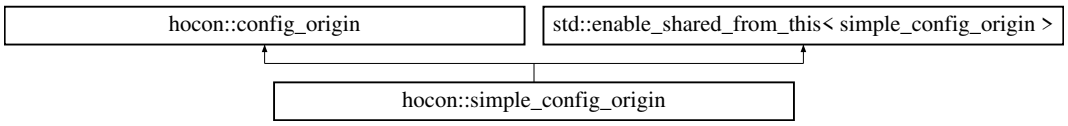
the new config_value with the given origin

The documentation for this class was generated from the following file:

- internal/values/simple_config_object.hpp

7.90 hocon::simple_config_origin Class Reference

Inheritance diagram for hocon::simple_config_origin:



Public Member Functions

- **simple_config_origin** (std::string description, int line_number, int end_line_number, origin_type org_type, std::string resource_or_null, std::vector< std::string > comments_or_null)
- simple_config_origin (std::string description, int line_number=-1, int end_line_number=-1, origin_type org_↩ type=origin_type::GENERIC)

  This constructor replaces the new_simple method in the original library.
- int line_number () const override

Returns a line number where the value or exception originated.

- std::string const & description () const override

  Returns a string describing the origin of a value or exception.

- std::vector< std::string > const & comments () const override

  Returns any comments that appeared to "go with" this place in the file.

- shared_origin with_line_number (int line_number) const override

  Returns a pointer to a copy of this origin with the specified line number as both starting and ending line.

- shared_origin with_comments (std::vector< std::string > comments) const override

  Returns a.

- std::shared_ptr< const simple_config_origin > **append_comments** (std::vector< std::string > comments) const
- std::shared_ptr< const simple_config_origin > **prepend_comments** (std::vector< std::string > comments) const
- bool **operator==** (const simple_config_origin &other) const
- bool **operator!=** (const simple_config_origin &other) const

Static Public Member Functions

- static shared_origin **merge_origins** (shared_origin a, shared_origin b)
- static shared_origin **merge_origins** (std::vector< shared_value > const &stack)
- static shared_origin **merge_origins** (std::vector< shared_origin > const &stack)

7.90.1 Detailed Description

Definition at line 13 of file simple_config_origin.hpp.

7.90.2 Constructor & Destructor Documentation

7.90.2.1 simple_config_origin()

```
hocon::simple_config_origin::simple_config_origin (
            std::string description,
            int line_number = -1,
            int end_line_number = -1,
            origin_type org_type = origin_type::GENERIC )
```

This constructor replaces the new_simple method in the original library.

7.90.3 Member Function Documentation

7.90.3.1 comments()

```
std::vector<std::string> const& hocon::simple_config_origin::comments ( ) const  [override], [virtual]
```

Returns any comments that appeared to "go with" this place in the file.

Often an empty list, but never null. The details of this are subject to change, but at the moment comments that are immediately before an array element or object field, with no blank line after the comment, "go with" that element or field.

Returns

any comments that seemed to "go with" this origin, empty list if none

Implements hocon::config_origin.

7.90.3.2 description()

```
std::string const& hocon::simple_config_origin::description ( ) const  [override], [virtual]
```

Returns a string describing the origin of a value or exception.

This will never return null.

Returns

string describing the origin

Implements hocon::config_origin.

7.90.3.3 line_number()

```
int hocon::simple_config_origin::line_number ( ) const  [override], [virtual]
```

Returns a line number where the value or exception originated.

This will return -1 if there's no meaningful line number.

Returns

line number or -1 if none is available

Implements hocon::config_origin.

7.90.3.4 with_comments()

```
shared_origin hocon::simple_config_origin::with_comments (
            std::vector< std::string > comments ) const  [override], [virtual]
```

Returns a.
config_origin

based on this one, but with the given comments. Does not modify this instance or any
config_value

s with this origin (since they are immutable). To set the returned origin to a
config_value

, use config_value#with_origin.

Note that when the given comments are equal to the comments on this object, a new instance may not be created and
this

is returned directly.

Since

1.3.0

Parameters

comments	the comments used on the returned origin

Returns

the config_origin with the given comments

Implements hocon::config_origin.

7.90.3.5 with_line_number()

```
shared_origin hocon::simple_config_origin::with_line_number (
            int line_number ) const  [override], [virtual]
```

Returns a pointer to a copy of this origin with the specified line number as both starting and ending line.

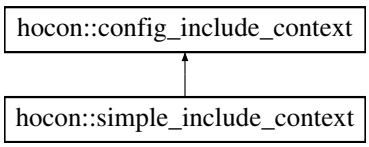
Implements hocon::config_origin.

The documentation for this class was generated from the following file:

- internal/simple_config_origin.hpp

7.91 hocon::simple_include_context Class Reference

Inheritance diagram for hocon::simple_include_context:



Public Member Functions

- **simple_include_context** (parseable const &parseable)
- shared_parseable relative_to (std::string file_name) const override

  Tries to find a name relative to whatever is doing the including, for example in the same directory as the file doing the including.
- config_parse_options parse_options () const override

  Parse options to use (if you use another method to get a config_parseable then use config_parseable#options() instead though).
- void **set_cur_dir** (std::string dir) const
- std::string **get_cur_dir** () const

Protected Attributes

- std::shared_ptr< std::string > **_cur_dir**

7.91.1 Detailed Description

Definition at line 8 of file simple_include_context.hpp.

7.91.2 Member Function Documentation

7.91.2.1 parse_options()

```
config_parse_options hocon::simple_include_context::parse_options ( ) const  [override], [virtual]
```

Parse options to use (if you use another method to get a config_parseable then use config_parseable#options() instead though).

Returns

the parse options

Implements hocon::config_include_context.

7.91.2.2 relative_to()

```
shared_parseable hocon::simple_include_context::relative_to (
            std::string file_name ) const  [override], [virtual]
```

Tries to find a name relative to whatever is doing the including, for example in the same directory as the file doing the including.

Returns null if it can't meaningfully create a relative name. The returned parseable may not exist; this function is not required to do any IO, just compute what the name would be.

The passed-in filename has to be a complete name (with extension), not just a basename. (Include statements in config files are allowed to give just a basename.)

Parameters

filename	the name to make relative to the resource doing the including

Returns

parseable item relative to the resource doing the including, or null

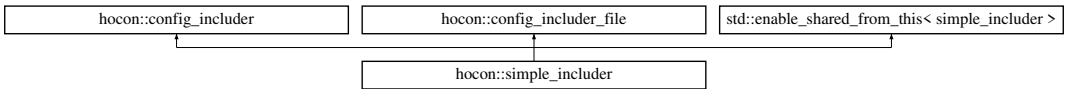
Implements hocon::config_include_context.

The documentation for this class was generated from the following file:

- internal/simple_include_context.hpp

7.92 hocon::simple_includer Class Reference

Inheritance diagram for hocon::simple_includer:



Public Member Functions

- **simple_includer** (shared_includer fallback)
- shared_includer with_fallback (shared_includer fallback) const override

  Returns a new includer that falls back to the given includer.
- shared_object include (shared_include_context context, std::string what) const override

  Parses another item to be included.
- shared_object **include_without_fallback** (shared_include_context context, std::string what) const
- shared_object include_file (shared_include_context context, std::string what) const override

  Parses another item to be included.

Static Public Member Functions

- static shared_object **include_file_without_fallback** (shared_include_context context, std::string what)
- static config_parse_options **clear_for_include** (config_parse_options const &options)
- static shared_object **from_basename** (std::shared_ptr< name_source > source, std::string name, config_parse_options options)
- static std::shared_ptr< const full_includer > **make_full** (std::shared_ptr< const config_includer > includer)

7.92.1 Detailed Description

Definition at line 12 of file simple_includer.hpp.

7.92.2 Member Function Documentation

7.92.2.1 include()

```
shared_object hocon::simple_includer::include (
            shared_include_context context,
            std::string what ) const  [override], [virtual]
```

Parses another item to be included.

The returned object typically would not have substitutions resolved. You can throw a config_exception here to abort parsing, or return an empty object, but may not return null.

This method is used for a "heuristic" include statement that does not specify file, or URL resource. If the include statement does specify, then the same class implementing config_includer must also implement config_includer_file or config_includer_URL as needed, or a default includer will be used.

Parameters

context	some info about the include context
what	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

context	some info about the include context
what	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

fallback	the previous includer for chaining

Returns

a new includer

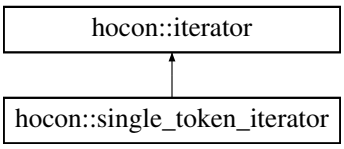
Implements hocon::config_includer.

The documentation for this class was generated from the following file:

- internal/simple_includer.hpp

7.93 hocon::single_token_iterator Class Reference

Inheritance diagram for hocon::single_token_iterator:



Public Member Functions

- **single_token_iterator** (shared_token token)
- bool **has_next** () override
- shared_token **next** () override

7.93.1 Detailed Description

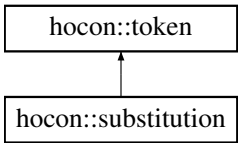
Definition at line 119 of file tokenizer.hpp.

The documentation for this class was generated from the following file:

- internal/tokenizer.hpp

7.94 hocon::substitution Class Reference

Inheritance diagram for hocon::substitution:



Public Member Functions

- **substitution** (shared_origin origin, bool optional, token_list expression)
- bool **optional** () const
- token_list const & **expression** () const
- std::string **token_text** () const override
- std::string **to_string** () const override
- bool **operator==** (const token &other) const override
- virtual token_type **get_token_type** () const
- virtual shared_origin const & **origin** () const
- int **line_number** () const

7.94.1 Detailed Description

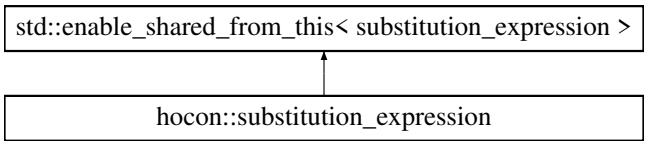
Definition at line 96 of file tokens.hpp.

The documentation for this class was generated from the following file:

- internal/tokens.hpp

7.95 hocon::substitution_expression Class Reference

Inheritance diagram for hocon::substitution_expression:



Public Member Functions

- **substitution_expression** (path the_path, bool optional)
- path **get_path** () const
- bool **optional** () const
- std::shared_ptr< substitution_expression > **change_path** (path new_path)
- std::string **to_string** () const
- bool **operator==** (substitution_expression const &other) const

7.95.1 Detailed Description

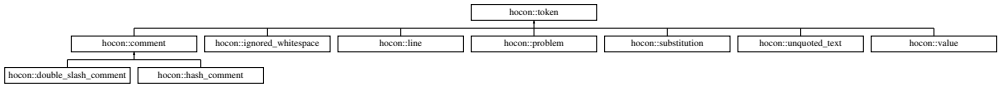
Definition at line 9 of file substitution_expression.hpp.

The documentation for this class was generated from the following file:

- internal/substitution_expression.hpp

7.96 hocon::token Class Reference

Inheritance diagram for hocon::token:



Public Member Functions

- **token** (token_type type, shared_origin origin=nullptr, std::string token_text="", std::string debug_string="")
- virtual token_type **get_token_type** () const
- virtual std::string **token_text** () const
- virtual std::string **to_string** () const
- virtual shared_origin const & **origin** () const
- int **line_number** () const
- virtual bool **operator==** (const token &other) const

7.96.1 Detailed Description

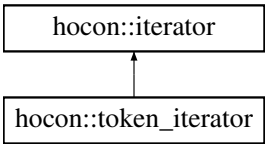
Definition at line 18 of file token.hpp.

The documentation for this class was generated from the following file:

- internal/token.hpp

7.97 hocon::token_iterator Class Reference

Inheritance diagram for hocon::token_iterator:



Public Member Functions

- **token_iterator** (shared_origin origin, std::unique_ptr< std::istream > input, bool allow_comments)
- **token_iterator** (shared_origin origin, std::unique_ptr< std::istream > input, config_syntax flavor)
- bool **has_next** () override
- shared_token **next** () override

Static Public Member Functions

- static std::string **render** (token_list tokens)

7.97.1 Detailed Description

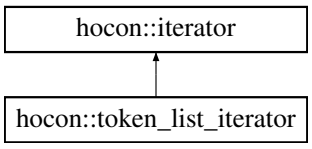
Definition at line 49 of file tokenizer.hpp.

The documentation for this class was generated from the following file:

- internal/tokenizer.hpp

7.98 hocon::token_list_iterator Class Reference

Inheritance diagram for hocon::token_list_iterator:



Public Member Functions

- **token_list_iterator** (token_list tokens)
- bool **has_next** () override
- shared_token **next** () override

7.98.1 Detailed Description

Definition at line 131 of file tokenizer.hpp.

The documentation for this class was generated from the following file:

- internal/tokenizer.hpp

7.99 hocon::tokens Class Reference

Static Public Member Functions

- static shared_token const & start_token ()

  Singleton tokens.
- static shared_token const & **end_token** ()
- static shared_token const & **comma_token** ()
- static shared_token const & **equals_token** ()
- static shared_token const & **colon_token** ()
- static shared_token const & **open_curly_token** ()
- static shared_token const & **close_curly_token** ()
- static shared_token const & **open_square_token** ()
- static shared_token const & **close_square_token** ()
- static shared_token const & **plus_equals_token** ()
- static bool **is_newline** (shared_token)
- static bool **is_ignored_whitespace** (shared_token)
- static bool **is_value_with_type** (shared_token t, config_value::type type)
- static shared_value **get_value** (shared_token t)

7.99.1 Detailed Description

Definition at line 113 of file tokens.hpp.

7.99.2 Member Function Documentation

7.99.2.1 start_token()

```
static shared_token const& hocon::tokens::start_token ( )  [static]
```

Singleton tokens.

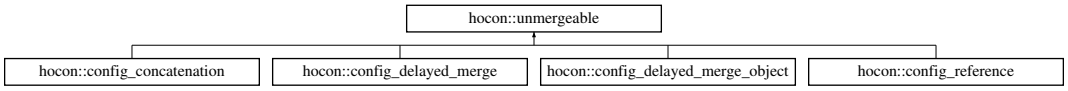
The documentation for this class was generated from the following file:

- internal/tokens.hpp

7.100 hocon::unmergeable Class Reference

Interface that tags a ConfigValue that is not mergeable until after substitutions are resolved.

Inheritance diagram for hocon::unmergeable:



Public Member Functions

- virtual std::vector< shared_value > **unmerged_values** () const =0

7.100.1 Detailed Description

Interface that tags a ConfigValue that is not mergeable until after substitutions are resolved.

Basically these are special ConfigValue that never appear in a resolved tree, like ConfigSubstitution and ConfigDelayedMerge.

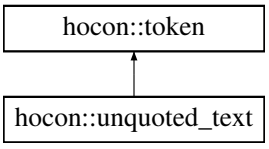
Definition at line 14 of file unmergeable.hpp.

The documentation for this class was generated from the following file:

- internal/unmergeable.hpp

7.101 hocon::unquoted_text Class Reference

Inheritance diagram for hocon::unquoted_text:



Public Member Functions

- **unquoted_text** (shared_origin origin, std::string text)
- std::string **to_string** () const override
- bool **operator==** (const token &other) const override
- virtual token_type **get_token_type** () const
- virtual std::string **token_text** () const
- virtual shared_origin const & **origin** () const
- int **line_number** () const

7.101.1 Detailed Description

Definition at line 33 of file tokens.hpp.

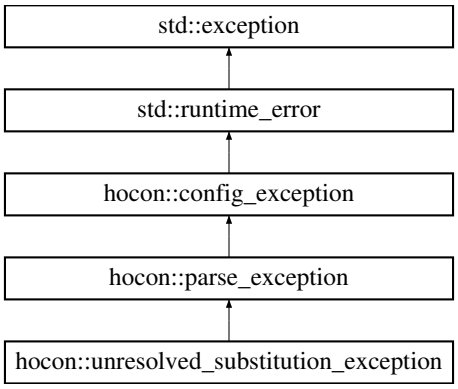
The documentation for this class was generated from the following file:

- internal/tokens.hpp

7.102 hocon::unresolved_substitution_exception Struct Reference

Exception indicating that a substitution did not resolve to anything.

Inheritance diagram for hocon::unresolved_substitution_exception:



Public Member Functions

- **unresolved_substitution_exception** (config_origin const &origin, std::string const &detail)

7.102.1 Detailed Description

Exception indicating that a substitution did not resolve to anything.

Thrown by config#resolve.

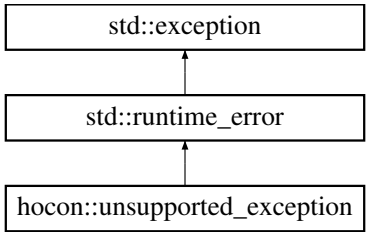
Definition at line 110 of file config_exception.hpp.

The documentation for this struct was generated from the following file:

- hocon/config_exception.hpp

7.103 hocon::unsupported_exception Struct Reference

Inheritance diagram for hocon::unsupported_exception:



Public Member Functions

- **unsupported_exception** (std::string const &message)

7.103.1 Detailed Description

Definition at line 14 of file token.hpp.

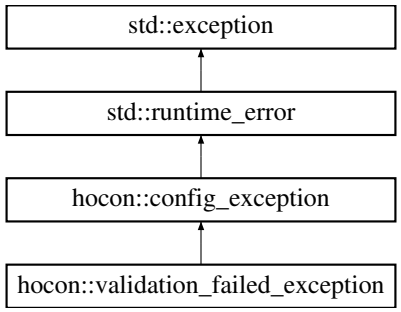
The documentation for this struct was generated from the following file:

- internal/token.hpp

7.104 hocon::validation_failed_exception Struct Reference

Exception indicating that config#check_valid found validity problems.

Inheritance diagram for hocon::validation_failed_exception:



Public Member Functions

- **validation_failed_exception** (std::vector< validation_problem > problems_)

Public Attributes

- const std::vector< validation_problem > **problems**

7.104.1 Detailed Description

Exception indicating that config#check_valid found validity problems.

The problems are available via the problems() method. The `get_message()` of this exception is a potentially very long string listing all the problems found.

Definition at line 155 of file config_exception.hpp.

The documentation for this struct was generated from the following file:

- hocon/config_exception.hpp

7.105 hocon::validation_problem Struct Reference

Information about a problem that occurred in config#check_valid.

Public Member Functions

- **validation_problem** (std::string path_, shared_origin origin_, std::string problem_)
- std::string **to_string** ()

Public Attributes

- const std::string **path**
- const shared_origin **origin**
- const std::string **problem**

7.105.1 Detailed Description

Information about a problem that occurred in config#check_valid.

A validation_failed_exception thrown from `check_valid()` includes a list of problems encountered.

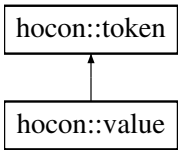
Definition at line 136 of file config_exception.hpp.

The documentation for this struct was generated from the following file:

- hocon/config_exception.hpp

7.106 hocon::value Class Reference

Inheritance diagram for hocon::value:



Public Member Functions

- **value** (shared_value value)
- **value** (shared_value value, std::string original_text)
- std::string **to_string** () const override
- shared_origin const & **origin** () const override
- shared_value **get_value** () const
- bool **operator==** (const token &other) const override
- virtual token_type **get_token_type** () const
- virtual std::string **token_text** () const
- int **line_number** () const

7.106.1 Detailed Description

Definition at line 8 of file tokens.hpp.

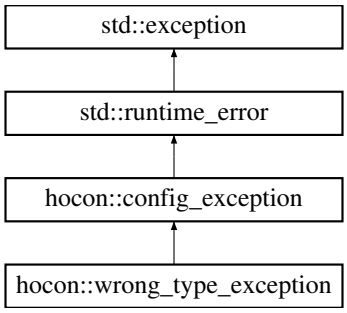
The documentation for this class was generated from the following file:

- internal/tokens.hpp

7.107 hocon::wrong_type_exception Struct Reference

Exception indicating that the type of a value does not match the type you requested.

Inheritance diagram for hocon::wrong_type_exception:



Public Member Functions

- **wrong_type_exception** (config_origin const &origin, std::string const &path, std::string const &expected, std::string const &actual)
- **config_exception** (config_origin const &origin, std::string const &message)
- **config_exception** (std::string const &message)
- **config_exception** (std::string const &message, std::exception const &e)

7.107.1 Detailed Description

Exception indicating that the type of a value does not match the type you requested.

Definition at line 27 of file config_exception.hpp.

The documentation for this struct was generated from the following file:

- hocon/config_exception.hpp

Chapter 8

File Documentation

8.1 hocon/version.h File Reference

Declares macros for the cpp-hocon library version.

Macros

- #define CPP_HOCON_VERSION_MAJOR 0
  The cpp-hocon library major version.
- #define CPP_HOCON_VERSION_MINOR 3
  The cpp-hocon library minor version.
- #define CPP_HOCON_VERSION_PATCH 0
  The cpp-hocon library patch version.
- #define CPP_HOCON_VERSION "0.3.0"
  The cpp-hocon library version as a string (without commit SHA1).
- #define CPP_HOCON_VERSION_WITH_COMMIT "0.3.0"
  The cpp-hocon library version as a string (with commit SHA1).

8.1.1 Detailed Description

Declares macros for the cpp-hocon library version.

8.1.2 Macro Definition Documentation

8.1.2.1 CPP_HOCON_VERSION

```
#define CPP_HOCON_VERSION "0.3.0"
```

The cpp-hocon library version as a string (without commit SHA1).

Definition at line 23 of file version.h.

8.1.2.2 CPP_HOCON_VERSION_MAJOR

```
#define CPP_HOCON_VERSION_MAJOR 0
```

The cpp-hocon library major version.

Definition at line 10 of file version.h.

8.1.2.3 CPP_HOCON_VERSION_MINOR

```
#define CPP_HOCON_VERSION_MINOR 3
```

The cpp-hocon library minor version.

Definition at line 14 of file version.h.

8.1.2.4 CPP_HOCON_VERSION_PATCH

```
#define CPP_HOCON_VERSION_PATCH 0
```

The cpp-hocon library patch version.

Definition at line 18 of file version.h.

8.1.2.5 CPP_HOCON_VERSION_WITH_COMMIT

```
#define CPP_HOCON_VERSION_WITH_COMMIT "0.3.0"
```

The cpp-hocon library version as a string (with commit SHA1).

Definition at line 28 of file version.h.

Index