# Boost Spirit API

D. Gueorguiev 10/9/21

# Main classes and functionality

## Tokenization

### The tokenize() function

The [tokenize()](https://github.com/boostorg/spirit/blob/boost-1.75.0/include/boost/spirit/home/lex/tokenize_and_parse.hpp#L227-L327) function is one of the main Spirit API functions. It resides entirely in the header [tokenize\_and\_parse.hpp](https://github.com/boostorg/spirit/blob/boost-1.75.0/include/boost/spirit/home/lex/tokenize_and_parse.hpp).

It simplifies the usage of lexer to tokenize a given input sequence. Its main purpose is to use the lexer to tokenize all the input.

The second version of [tokenize()](https://github.com/boostorg/spirit/blob/boost-1.75.0/include/boost/spirit/home/lex/tokenize_and_parse.hpp#L313-L327) (shown below) discards all generated tokens afterwards. This is useful whenever all the needed functionality has been implemented directly inside the lexer semantic actions, which are being executed while the tokens are matched.

The function takes a pair of iterators spanning the underlying stream to scan, the lexer object built from the token definitions, and an optional functor being called for each of the generated tokens.

The function returns true if the scanning of the input succeeded which is the given input sequence has been successfully matched by the given token definitions. Here are the arguments:

first, last: The pair of iterators spanning the underlying input sequence to parse. These iterators must at least conform to the requirements of std::input\_iterator category.

*Side note 1*:

The base concept of std::input\_iterator is [std::input\_or\_output\_iterator](https://en.cppreference.com/w/cpp/iterator/input_or_output_iterator) (defined in header [<iterator>](https://en.cppreference.com/w/cpp/header/iterator)):

template <class I>

concept input\_or\_output\_iterator =

requires(I i) {

{ \*i } -> \_\_Referenceable;

} &&

std::weakly\_incrementable<I>;

The input\_or\_output\_iterator concept forms the basis of the iterator concept taxonomy; every iterator type satisfies the

input\_or\_output\_iterator requirements. The exposition-only concept \_\_Referenceable if and only if the expression \*std::declval<I&>() is valid and has referenceable type (in particular not void).

*Side node 2:*

*Equality Preservation*

An expression is *equality preserving* if it results in equal outputs given equal inputs.

The inputs to an expression consists of its operands

The outputs of an expression consists of its result and all operands modified by the expression (if any).

In specification of standard concepts, operands are defined as the largest subexpressions that include only:

* an id-expression *and*
* invocations of std::move, std::forward and std::declval

The cv-qualification and value category of each operand