Tokenizer reference

edit

A *tokenizer* receives a stream of characters, breaks it up into individual *tokens* (usually individual words), and outputs a stream of *tokens*. For instance, a `whitespace` tokenizer breaks text into tokens whenever it sees any whitespace. It would convert the text `"Quick brown fox!"` into the terms `[Quick, brown, fox!]`.

The tokenizer is also responsible for recording the following:

- Order or *position* of each term (used for phrase and word proximity queries)
- Start and end *character offsets* of the original word which the term represents (used for highlighting search snippets).
- *Token type*, a classification of each term produced, such as `<ALPHANUM>`, `<HANGUL>`, or `<NUM>`. Simpler analyzers only produce the `word` token type.

Elasticsearch has a number of built in tokenizers which can be used to build custom analyzers.

Word Oriented Tokenizers

edit

The following tokenizers are usually used for tokenizing full text into individual words:

Standard Tokenizer
: The `standard` tokenizer divides text into terms on word boundaries, as defined by the Unicode Text Segmentation algorithm. It removes most punctuation symbols. It is the best choice for most languages.

Letter Tokenizer
: The `letter` tokenizer divides text into terms whenever it encounters a character which is not a letter.

Lowercase Tokenizer
: The `lowercase` tokenizer, like the `letter` tokenizer, divides text into terms whenever it encounters a character which is not a letter, but it also lowercases all terms.

Whitespace Tokenizer
: The `whitespace` tokenizer divides text into terms whenever it encounters any whitespace character.

UAX URL Email Tokenizer

The `uax_url_email` tokenizer is like the `standard` tokenizer except that it recognises URLs and email addresses as single tokens.

Classic Tokenizer
: The `classic` tokenizer is a grammar based tokenizer for the English Language.

Thai Tokenizer
: The `thai` tokenizer segments Thai text into words.

Partial Word Tokenizers

edit

These tokenizers break up text or words into small fragments, for partial word matching:

N-Gram Tokenizer
: The `ngram` tokenizer can break up text into words when it encounters any of a list of specified characters (e.g. whitespace or punctuation), then it returns n-grams of each word: a sliding window of continuous letters, e.g. `quick` → `[qu, ui, ic, ck]`.

Edge N-Gram Tokenizer
: The `edge_ngram` tokenizer can break up text into words when it encounters any of a list of specified characters (e.g. whitespace or punctuation), then it returns n-grams of each word which are anchored to the start of the word, e.g. `quick` → `[q, qu, qui, quic, quick]`.

Structured Text Tokenizers

edit

The following tokenizers are usually used with structured text like identifiers, email addresses, zip codes, and paths, rather than with full text:

Keyword Tokenizer
: The `keyword` tokenizer is a “noop” tokenizer that accepts whatever text it is given and outputs the exact same text as a single term. It can be combined with token filters like `lowercase` to normalise the analysed terms.

Pattern Tokenizer
: The `pattern` tokenizer uses a regular expression to either split text into terms whenever it matches a word separator, or to capture matching text as terms.

Simple Pattern Tokenizer
: The `simple_pattern` tokenizer uses a regular expression to capture matching text as terms. It uses a restricted subset of regular expression features and is generally faster than the `pattern` tokenizer.

Char Group Tokenizer
: The `char_group` tokenizer is configurable through sets of characters to split on, which is usually less expensive than running regular expressions.

Simple Pattern Split Tokenizer
: The `simple_pattern_split` tokenizer uses the same restricted regular expression subset as the `simple_pattern` tokenizer, but splits the input at matches rather than returning the matches as terms.

Path Tokenizer
: The `path_hierarchy` tokenizer takes a hierarchical value like a filesystem path, splits on the path separator, and emits a term for each component in the tree, e.g. `/foo/bar/baz` → `[/foo, /foo/bar, /foo/bar/baz ]`.

« Whitespace analyzer Character group tokenizer »

Register now for free for ElasticON Global 2023

Our biggest user conference of the year is happening March 7-8. Join us virtually for all new technical deep dives, live workshops, demos, and more!

Learn more