Docs

NOTE: You are looking at documentation for an older release. For the latest information, see the current release documentation.

Highlighting

edit

Highlighters enable you to get highlighted snippets from one or more fields in your search results so you can show users where the query matches are. When you request highlights, the response contains an additional `highlight` element for each search hit that includes the highlighted fields and the highlighted fragments.

NOTE Highlighters don't reflect the boolean logic of a query when extracting terms to highlight. Thus, for some complex boolean queries (e.g nested boolean queries, queries using `minimum_should_match` etc.), parts of documents may be highlighted that don't correspond to query matches.

Highlighting requires the actual content of a field. If the field is not stored (the mapping does not set `store` to `true`), the actual `_source` is loaded and the relevant field is extracted from `_source`.

NOTE The `_all` field cannot be extracted from `_source`, so it can only be used for highlighting if it is explicitly stored.

For example, to get highlights for the `content` field in each search hit using the default highlighter, include a `highlight` object in the request body that specifies the `content` field:

```
GET /_search
{
    "query" : {
        "match": { "content": "kimchy" }
    },
    "highlight" : {
        "fields" : {
            "content" : {}
        }
    }
}
```

Copy as cURL View in Console

Elasticsearch supports three highlighters: `unified`, `plain`, and `fvh` (fast vector highlighter). You can specify the highlighter `type` you want to use for each field.

Unified highlighter

edit

The `unified` highlighter uses the Lucene Unified Highlighter. This highlighter breaks the text into sentences and uses the BM25 algorithm to score individual sentences as if they were documents in the corpus. It also supports accurate

On this page

phrase and multi-term (fuzzy, prefix, regex) highlighting. This is the default highlighter.

Plain highlighter

edit

The `plain` highlighter uses the standard Lucene highlighter. It attempts to reflect the query matching logic in terms of understanding word importance and any word positioning criteria in phrase queries.

WARNING The `plain` highlighter works best for highlighting simple query matches in a single field. To accurately reflect query logic, it creates a tiny in-memory index and re-runs the original query criteria through Lucene's query execution planner to get access to low-level match information for the current document. This is repeated for every field and every document that needs to be highlighted. If you want to highlight a lot of fields in a lot of documents with complex queries, we recommend using the `unified` highlighter on `postings` or `term_vector` fields.

Fast vector highlighter

edit

The `fvh` highlighter uses the Lucene Fast Vector highlighter. This highlighter can be used on fields with `term_vector` set to `with_positions_offsets` in the mapping. The fast vector highlighter:

- Can be customized with a `boundary_scanner`.
- Requires setting `term_vector` to `with_positions_offsets` which increases the size of the index
- Can combine matches from multiple fields into one result. See `matched_fields`
- Can assign different weights to matches at different positions allowing for things like phrase matches being sorted above term matches when highlighting a Boosting Query that boosts phrase matches over term matches

WARNING The `fvh` highlighter does not support span queries. If you need support for span queries, try an alternative highlighter, such as the `unified` highlighter.

Offsets Strategy

edit

To create meaningful search snippets from the terms being queried, the highlighter needs to know the start and end character offsets of each word in the original text. These offsets can be obtained from:

- The postings list. If `index_options` is set to `offsets` in the mapping, the `unified` highlighter uses this information to highlight documents without re-analyzing the text. It re-runs the original query directly on the postings and extracts the matching offsets from the index, limiting the collection to the highlighted documents. This is important if you have large fields because it doesn't require reanalyzing the text to be highlighted. It also requires less disk space than using `term_vectors`.
- Term vectors. If `term_vector` information is provided by setting `term_vector` to `with_positions_offsets` in the mapping, the `unified` highlighter automatically uses the `term_vector` to highlight the field. It's fast especially for large fields (> `1MB`) and for highlighting multi-term queries like `prefix` or

`wildcard` because it can access the dictionary of terms for each document. The `fvh` highlighter always uses term vectors.

- Plain highlighting. This mode is used by the `unified` when there is no other alternative. It creates a tiny in-memory index and re-runs the original query criteria through Lucene's query execution planner to get access to low-level match information on the current document. This is repeated for every field and every document that needs highlighting. The `plain` highlighter always uses plain highlighting.

WARNING Plain highlighting for large texts may require substantial amount of time and memory. To protect against this, the maximum number of text characters to be analyzed will be limited to 1000000 in the next major Elastic version. The default limit is not set for this version, but can be set for a particular index with the index setting `index.highlight.max_analyzed_offset`.

Set up a cluster for high availability
Secure a cluster
Alerting on Cluster and Index Events
Command line tools
How To
Testing
Glossary of terms
X-Pack APIs
Release Highlights
Breaking changes
Release Notes

Highlighting Settings

edit

Highlighting settings can be set on a global level and overridden at the field level.

boundary_chars
A string that contains each boundary character. Defaults to `.,!? \t\n`.

boundary_max_scan
How far to scan for boundary characters. Defaults to `20`.

boundary_scanner
Specifies how to break the highlighted fragments: `chars`, `sentence`, or `word`. Only valid for the `unified` and `fvh` highlighters. Defaults to `sentence` for the `unified` highlighter. Defaults to `chars` for the `fvh` highlighter.

`chars`
Use the characters specified by `boundary_chars` as highlighting boundaries. The `boundary_max_scan` setting controls how far to scan for boundary characters. Only valid for the `fvh` highlighter.

`sentence`
Break highlighted fragments at the next sentence boundary, as determined by Java's BreakIterator. You can specify the locale to use with `boundary_scanner_locale`.

NOTE When used with the `unified` highlighter, the `sentence` scanner splits sentences bigger than `fragment_size` at the first word boundary next to `fragment_size`. You can set `fragment_size` to 0 to never split any sentence.

`word`
Break highlighted fragments at the next word boundary, as determined by Java's BreakIterator. You can specify the locale to use with `boundary_scanner_locale`.

boundary_scanner_locale
Controls which locale is used to search for sentence and word boundaries. This parameter takes a form of a language tag, e.g. `"en-US"`, `"fr-FR"`, `"ja-JP"`. More info can be found in the Locale Language Tag documentation. The default value is Locale.ROOT.

encoder
Indicates if the snippet should be HTML encoded: `default` (no encoding) or `html` (HTML-escape the snippet text and then insert the highlighting tags)

fields

Specifies the fields to retrieve highlights for. You can use wildcards to specify fields. For example, you could specify `comment_*` to get highlights for all text and keyword fields that start with `comment_`.

NOTE Only text and keyword fields are highlighted when you use wildcards. If you use a custom mapper and want to highlight on a field anyway, you must explicitly specify that field name.

force_source

Highlight based on the source even if the field is stored separately. Defaults to `false`.

fragmenter

Specifies how text should be broken up in highlight snippets: `simple` or `span`. Only valid for the `plain` highlighter. Defaults to `span`.

`simple`

Breaks up text into same-sized fragments.

`span`

Breaks up text into same-sized fragments, but tries to avoid breaking up text between highlighted terms. This is helpful when you're querying for phrases. Default.

fragment_offset

Controls the margin from which you want to start highlighting. Only valid when using the `fvh` highlighter.

fragment_size

The size of the highlighted fragment in characters. Defaults to 100.

highlight_query

Highlight matches for a query other than the search query. This is especially useful if you use a rescore query because those are not taken into account by highlighting by default.

IMPORTANT Elasticsearch does not validate that `highlight_query` contains the search query in any way so it is possible to define it so legitimate query results are not highlighted. Generally, you should include the search query as part of the `highlight_query`.

matched_fields

Combine matches on multiple fields to highlight a single field. This is most intuitive for multifields that analyze the same string in different ways. All `matched_fields` must have `term_vector` set to `with_positions_offsets`, but only the field to which the matches are combined is loaded so only that field benefits from having `store` set to `yes`. Only valid for the `fvh` highlighter.

no_match_size

The amount of text you want to return from the beginning of the field if there are no matching fragments to highlight. Defaults to 0 (nothing is returned).

number_of_fragments

The maximum number of fragments to return. If the number of fragments is set to 0, no fragments are returned. Instead, the entire field contents are highlighted and returned. This can be handy when you need to highlight short texts such as a title or address, but fragmentation is not required. If `number_of_fragments` is 0, `fragment_size` is ignored. Defaults to 5.

order

Sorts highlighted fragments by score when set to `score`. By default, fragments will be output in the order they appear in the field (order: `none`). Setting this option to `score` will output the most relevant fragments first. Each highlighter applies its own logic to compute relevancy scores. See the document How highlighters work internally for more details how different highlighters find the best fragments.

phrase_limit
: Controls the number of matching phrases in a document that are considered. Prevents the `fvh` highlighter from analyzing too many phrases and consuming too much memory. When using `matched_fields`, `phrase_limit` phrases per matched field are considered. Raising the limit increases query time and consumes more memory. Only supported by the `fvh` highlighter. Defaults to 256.

pre_tags
: Use in conjunction with `post_tags` to define the HTML tags to use for the highlighted text. By default, highlighted text is wrapped in `<em>` and `</em>` tags. Specify as an array of strings.

post_tags
: Use in conjunction with `pre_tags` to define the HTML tags to use for the highlighted text. By default, highlighted text is wrapped in `<em>` and `</em>` tags. Specify as an array of strings.

require_field_match
: By default, only fields that contains a query match are highlighted. Set `require_field_match` to `false` to highlight all fields. Defaults to `true`.

tags_schema
: Set to `styled` to use the built-in tag schema. The `styled` schema defines the following `pre_tags` and defines `post_tags` as `</em>`.

```
<em class="hlt1">, <em class="hlt2">, <em class="hlt3">,
<em class="hlt4">, <em class="hlt5">, <em class="hlt6">,
<em class="hlt7">, <em class="hlt8">, <em class="hlt9">,
<em class="hlt10">
```

type
: The highlighter to use: `unified`, `plain`, or `fvh`. Defaults to `unified`.

Highlighting Examples

edit

- Override global settings
- Specify a highlight query
- Set highlighter type
- Configure highlighting tags
- Highlight source
- Combine matches on multiple fields
- Explicitly order highlighted fields
- Control highlighted fragments
- Highlight using the postings list
- Specify a fragmenter for the plain highlighter

Override global settings

edit

You can specify highlighter settings globally and selectively override them for individual fields.

```
GET /_search
{
    "query" : {
        "match": { "user": "kimchy" }
    },
    "highlight" : {
        "number_of_fragments" : 3,
        "fragment_size" : 150,
        "fields" : {
            "_all" : { "pre_tags" : ["<em>"], "post_tags" : ["
</em>"] },
            "blog.title" : { "number_of_fragments" : 0 },
            "blog.author" : { "number_of_fragments" : 0 },
            "blog.comment" : { "number_of_fragments" : 5, "order" :
"score" }
        }
    }
}
```

Copy as cURL View in Console

Specify a highlight query

edit

You can specify a `highlight_query` to take additional information into account when highlighting. For example, the following query includes both the search query and rescore query in the `highlight_query`. Without the `highlight_query`, highlighting would only take the search query into account.

```
GET /_search
{
    "stored_fields": [ "_id" ],
    "query" : {
        "match": {
            "comment": {
                "query": "foo bar"
            }
        }
    },
    "rescore": {
        "window_size": 50,
        "query": {
            "rescore_query" : {
                "match_phrase": {
                    "comment": {
                        "query": "foo bar",
                        "slop": 1
                    }
                }
            },
            "rescore_query_weight" : 10
        }
    },
    "highlight" : {
        "order" : "score",
        "fields" : {
            "comment" : {
                "fragment_size" : 150,
                "number_of_fragments" : 3,
                "highlight_query": {
                    "bool": {
                        "must": {
                            "match": {
                                "comment": {
                                    "query": "foo bar"
                                }
                            }
                        },
                        "should": {
                            "match_phrase": {
                                "comment": {
                                    "query": "foo bar",
                                    "slop": 1,
                                    "boost": 10.0
                                }
                            }
                        },
                        "minimum_should_match": 0
                    }
                }
            }
        }
    }
}
```

Copy as cURL View in Console

Set highlighter type

edit

The `type` field allows to force a specific highlighter type. The allowed values are: `unified`, `plain` and `fvh`. The following is an example that forces the use of the plain highlighter:

```
GET /_search
{
    "query" : {
        "match": { "user": "kimchy" }
    },
    "highlight" : {
        "fields" : {
            "comment" : {"type" : "plain"}
        }
    }
}
```

Copy as cURL View in Console

Configure highlighting tags

edit

By default, the highlighting will wrap highlighted text in `<em>` and `</em>`. This can be controlled by setting `pre_tags` and `post_tags`, for example:

```
GET /_search
{
    "query" : {
        "match": { "user": "kimchy" }
    },
    "highlight" : {
        "pre_tags" : ["<tag1>"],
        "post_tags" : ["</tag1>"],
        "fields" : {
            "_all" : {}
        }
    }
}
```

Copy as cURL View in Console

When using the fast vector highlighter, you can specify additional tags and the "importance" is ordered.

```
GET /_search
{
    "query" : {
        "match": { "user": "kimchy" }
    },
    "highlight" : {
        "pre_tags" : ["<tag1>", "<tag2>"],
        "post_tags" : ["</tag1>", "</tag2>"],
        "fields" : {
            "_all" : {}
        }
    }
}
```

Copy as cURL View in Console

You can also use the built-in `styled` tag schema:

```
GET /_search
{
    "query" : {
        "match": { "user": "kimchy" }
    },
    "highlight" : {
        "tags_schema" : "styled",
        "fields" : {
            "comment" : {}
        }
    }
}
```

Copy as cURL View in Console

Highlight on source

edit

Forces the highlighting to highlight fields based on the source even if fields are stored separately. Defaults to `false`.

```
GET /_search
{
    "query" : {
        "match": { "user": "kimchy" }
    },
    "highlight" : {
        "fields" : {
            "comment" : {"force_source" : true}
        }
    }
}
```

Copy as cURL View in Console

Combine matches on multiple fields

edit

¯his is only supported by the `fvh` highlighter

WARNING

The Fast Vector Highlighter can combine matches on multiple fields to highlight a single field. This is most intuitive for multifields that analyze the same string in different ways. All `matched_fields` must have `term_vector` set to `with_positions_offsets` but only the field to which the matches are combined is loaded so only that field would benefit from having `store` set to `yes`.

In the following examples, `comment` is analyzed by the `english` analyzer and `comment.plain` is analyzed by the `standard` analyzer.

```
GET /_search
{
    "query": {
        "query_string": {
            "query": "comment.plain:running scissors",
            "fields": ["comment"]
        }
    },
    "highlight": {
        "order": "score",
        "fields": {
            "comment": {
                "matched_fields": ["comment", "comment.plain"],
                "type" : "fvh"
            }
        }
    }
}
```

Copy as cURL View in Console

The above matches both "run with scissors" and "running with scissors" and would highlight "running" and "scissors" but not "run". If both phrases appear in a large document then "running with scissors" is sorted above "run with scissors" in the fragments list because there are more matches in that fragment.

```
GET /_search
{
    "query": {
        "query_string": {
            "query": "running scissors",
            "fields": ["comment", "comment.plain^10"]
        }
    },
    "highlight": {
        "order": "score",
        "fields": {
            "comment": {
                "matched_fields": ["comment", "comment.plain"],
                "type" : "fvh"
            }
        }
    }
}
```

Copy as cURL View in Console

The above highlights "run" as well as "running" and "scissors" but still sorts "running with scissors" above "run with scissors" because the plain match ("running") is boosted.

```
GET /_search
{
    "query": {
        "query_string": {
            "query": "running scissors",
            "fields": ["comment", "comment.plain^10"]
        }
    },
    "highlight": {
        "order": "score",
        "fields": {
            "comment": {
                "matched_fields": ["comment.plain"],
                "type" : "fvh"
            }
        }
    }
}
```

Copy as cURL View in Console

The above query wouldn't highlight "run" or "scissor" but shows that it is just fine not to list the field to which the matches are combined (`comment`) in the matched fields.

NOTE Technically it is also fine to add fields to `matched_fields` that don't share the same underlying string as the field to which the matches are combined. The results might not make much sense and if one of the matches is off the end of the text then the whole query will fail.

NOTE There is a small amount of overhead involved with setting `matched_fields` to a non-empty array so always prefer

```
"highlight": {
    "fields": {
        "comment": {}
    }
}
```

to

```
"highlight": {
    "fields": {
        "comment": {
            "matched_fields": ["comment"],
            "type" : "fvh"
        }
    }
}
```

Explicitly order highlighted fields

edit

Elasticsearch highlights the fields in the order that they are sent, but per the JSON spec, objects are unordered. If you need to be explicit about the order in which fields are highlighted specify the `fields` as an array:

```
GET /_search
{
    "highlight": {
        "fields": [
            { "title": {} },
            { "text": {} }
        ]
    }
}
```

Copy as cURL View in Console

None of the highlighters built into Elasticsearch care about the order that the fields are highlighted but a plugin might.

Control highlighted fragments

edit

Each field highlighted can control the size of the highlighted fragment in characters (defaults to `100`), and the maximum number of fragments to return (defaults to `5`). For example:

```
GET /_search
{
    "query" : {
        "match": { "user": "kimchy" }
    },
    "highlight" : {
        "fields" : {
            "comment" : {"fragment_size" : 150,
"number_of_fragments" : 3}
        }
    }
}
```

Copy as cURL View in Console

On top of this it is possible to specify that highlighted fragments need to be sorted by score:

```
GET /_search
{
    "query" : {
        "match": { "user": "kimchy" }
    },
    "highlight" : {
        "order" : "score",
        "fields" : {
            "comment" : {"fragment_size" : 150,
"number_of_fragments" : 3}
        }
    }
}
```

Copy as cURL View in Console

If the `number_of_fragments` value is set to `0` then no fragments are produced, instead the whole content of the field is returned, and of course it is highlighted. This can be very handy if short texts (like document title or address) need to be

highlighted but no fragmentation is required. Note that `fragment_size` is ignored in this case.

```
GET /_search
{
    "query" : {
        "match": { "user": "kimchy" }
    },
    "highlight" : {
        "fields" : {
            "_all" : {},
            "blog.title" : {"number_of_fragments" : 0}
        }
    }
}
```

Copy as cURL View in Console

When using `fvh` one can use `fragment_offset` parameter to control the margin to start highlighting from.

In the case where there is no matching fragment to highlight, the default is to not return anything. Instead, we can return a snippet of text from the beginning of the field by setting `no_match_size` (default `0`) to the length of the text that you want returned. The actual length may be shorter or longer than specified as it tries to break on a word boundary.

```
GET /_search
{
    "query" : {
        "match": { "user": "kimchy" }
    },
    "highlight" : {
        "fields" : {
            "comment" : {
                "fragment_size" : 150,
                "number_of_fragments" : 3,
                "no_match_size": 150
            }
        }
    }
}
```

Copy as cURL View in Console

Highlight using the postings list

edit

Here is an example of setting the `comment` field in the index mapping to allow for highlighting using the postings:

```
PUT /example
{
  "mappings": {
    "doc" : {
      "properties": {
        "comment" : {
          "type": "text",
          "index_options" : "offsets"
        }
      }
    }
  }
}
```

Copy as cURL View in Console

Here is an example of setting the `comment` field to allow for highlighting using the `term_vectors` (this will cause the index to be bigger):

```
PUT /example
{
  "mappings": {
    "doc" : {
      "properties": {
        "comment" : {
          "type": "text",
          "term_vector" : "with_positions_offsets"
        }
      }
    }
  }
}
```

Copy as cURL View in Console

Specify a fragmenter for the plain highlighter

edit

When using the `plain` highlighter, you can choose between the `simple` and `span` fragmenters:

```
GET twitter/_search
{
    "query" : {
        "match_phrase": { "message": "number 1" }
    },
    "highlight" : {
        "fields" : {
            "message" : {
                "type": "plain",
                "fragment_size" : 15,
                "number_of_fragments" : 3,
                "fragmenter": "simple"
            }
        }
    }
}
```

Copy as cURL View in Console

Response:

```
{
    ...
    "hits": {
        "total": 1,
        "max_score": 1.601195,
        "hits": [
            {
                "_index": "twitter",
                "_type": "_doc",
                "_id": "1",
                "_score": 1.601195,
                "_source": {
                    "user": "test",
                    "message": "some message with the number 1",
                    "date": "2009-11-15T14:12:12",
                    "likes": 1
                },
                "highlight": {
                    "message": [
                        " with the <em>number</em>",
                        " <em>1</em>"
                    ]
                }
            }
        ]
    }
}
```

```
GET twitter/_search
{
    "query" : {
        "match_phrase": { "message": "number 1" }
    },
    "highlight" : {
        "fields" : {
            "message" : {
                "type": "plain",
                "fragment_size" : 15,
                "number_of_fragments" : 3,
                "fragmenter": "span"
            }
        }
    }
}
```

Copy as cURL View in Console

Response:

```
{
    ...
    "hits": {
        "total": 1,
        "max_score": 1.601195,
        "hits": [
            {
                "_index": "twitter",
                "_type": "_doc",
                "_id": "1",
                "_score": 1.601195,
                "_source": {
                    "user": "test",
                    "message": "some message with the number 1",
                    "date": "2009-11-15T14:12:12",
                    "likes": 1
                },
                "highlight": {
                    "message": [
                        " with the <em>number</em> <em>1</em>"
                    ]
                }
            }
        ]
    }
}
```

If the `number_of_fragments` option is set to `0`, `NullFragmenter` is used which does not fragment the text at all. This is useful for highlighting the entire contents of a document or field.

How highlighters work internally

edit

Given a query and a text (the content of a document field), the goal of a highlighter is to find the best text fragments for the query, and highlight the query terms in the found fragments. For this, a highlighter needs to address several questions:

- How break a text into fragments?
- How to find the best fragments among all fragments?
- How to highlight the query terms in a fragment?

How to break a text into fragments?

edit

Relevant settings: `fragment_size`, `fragmenter`, `type` of highlighter, `boundary_chars`, `boundary_max_scan`, `boundary_scanner`, `boundary_scanner_locale`.

Plain highlighter begins with analyzing the text using the given analyzer, and creating a token stream from it. Plain highlighter uses a very simple algorithm to break the token stream into fragments. It loops through terms in the token stream, and every time the current term's end_offset exceeds `fragment_size` multiplied by the number of created fragments, a new fragment is created. A little more computation is done with using `span` fragmenter to avoid breaking up text between highlighted terms. But overall, since the breaking is done only by `fragment_size`, some fragments can be quite odd, e.g. beginning with a punctuation mark.

Unified or FVH highlighters do a better job of breaking up a text into fragments by utilizing Java's `BreakIterator`. This ensures that a fragment is a valid sentence as long as `fragment_size` allows for this.

How to find the best fragments?

edit

Relevant settings: `number_of_fragments`.

To find the best, most relevant, fragments, a highlighter needs to score each fragment in respect to the given query. The goal is to score only those terms that participated in generating the *hit* on the document. For some complex queries, this is still work in progress.

The plain highlighter creates an in-memory index from the current token stream, and re-runs the original query criteria through Lucene's query execution planner to get access to low-level match information for the current text. For more complex queries the original query could be converted to a span query, as span queries can handle phrases more accurately. Then this obtained low-level match information is used to score each individual fragment. The scoring method of the plain highlighter is quite simple. Each fragment is scored by the number of unique query terms found in this fragment. The score of individual term is equal to its boost, which is by default is 1. Thus, by default, a fragment that contains one unique query term, will get a score of 1; and a fragment that contains two unique query terms, will get a score of 2 and so on. The fragments are then sorted by their scores, so the highest scored fragments will be output first.

FVH doesn't need to analyze the text and build an in-memory index, as it uses pre-indexed document term vectors, and finds among them terms that correspond to the query. FVH scores each fragment by the number of query terms found in this fragment. Similarly to plain highlighter, score of individual term is equal to its boost value. In contrast to plain highlighter, all query terms are counted, not only unique terms.

Unified highlighter can use pre-indexed term vectors or pre-indexed terms offsets, if they are available. Otherwise, similar to Plain Highlighter, it has to create an in-memory index from the text. Unified highlighter uses the BM25 scoring model to score fragments.

How to highlight the query terms in a fragment?

edit

Relevant settings: `pre-tags`, `post-tags`.

The goal is to highlight only those terms that participated in generating the *hit* on the document. For some complex boolean queries, this is still work in progress, as highlighters don't reflect the boolean logic of a query and only extract leaf (terms, phrases, prefix etc) queries.

Plain highlighter given the token stream and the original text, recomposes the original text to highlight only terms from the token stream that are contained in the low-level match information structure from the previous step.

FVH and unified highlighter use intermediate data structures to represent fragments in some raw form, and then populate them with actual text.

A highlighter uses `pre-tags`, `post-tags` to encode highlighted terms.

An example of the work of the unified highlighter

edit

Let's look in more details how unified highlighter works.

First, we create a index with a text field `content`, that will be indexed using `english` analyzer, and will be indexed without offsets or term vectors.

```
PUT test_index
{
    "mappings": {
        "_doc": {
            "properties": {
                "content" : {
                    "type" : "text",
                    "analyzer" : "english"
                }
            }
        }
    }
}
```

We put the following document into the index:

```
PUT test_index/_doc/doc1
{
  "content" : "For you I'm only a fox like a hundred thousand other
foxes. But if you tame me, we'll need each other. You'll be the
only boy in the world for me. I'll be the only fox in the world for
you."
}
```

And we ran the following query with a highlight request:

```
GET test_index/_search
{
    "query": {
        "match_phrase" : {"content" : "only fox"}
    },
    "highlight": {
        "type" : "unified",
        "number_of_fragments" : 3,
        "fields": {
            "content": {}
        }
    }
}
```

After `doc1` is found as a hit for this query, this hit will be passed to the unified highlighter for highlighting the field `content` of the document. Since the field `content` was not indexed either with offsets or term vectors, its raw field value will be analyzed, and in-memory index will be built from the terms that match the query:

```
{"token":"onli","start_offset":12,"end_offset":16,"position":3},
{"token":"fox","start_offset":19,"end_offset":22,"position":5},
{"token":"fox","start_offset":53,"end_offset":58,"position":11},
{"token":"onli","start_offset":117,"end_offset":121,"position":24},
{"token":"onli","start_offset":159,"end_offset":163,"position":34},
{"token":"fox","start_offset":164,"end_offset":167,"position":35}
```

Our complex phrase query will be converted to the span query: `spanNear([text:onli, text:fox], 0, true)`, meaning that we are looking for terms "onli: and "fox" within 0 distance from each other, and in the given order. The span

query will be run against the created before in-memory index, to find the following match:

```
{"term":"onli", "start_offset":159, "end_offset":163},
{"term":"fox", "start_offset":164, "end_offset":167}
```

In our example, we have got a single match, but there could be several matches. Given the matches, the unified highlighter breaks the text of the field into so called "passages". Each passage must contain at least one match. The unified highlighter with the use of Java's `BreakIterator` ensures that each passage represents a full sentence as long as it doesn't exceed `fragment_size`. For our example, we have got a single passage with the following properties (showing only a subset of the properties here):

```
Passage:
    startOffset: 147
    endOffset: 189
    score: 3.7158387
    matchStarts: [159, 164]
    matchEnds: [163, 167]
    numMatches: 2
```

Notice how a passage has a score, calculated using the BM25 scoring formula adapted for passages. Scores allow us to choose the best scoring passages if there are more passages available than the requested by the user `number_of_fragments`. Scores also let us to sort passages by `order: "score"` if requested by the user.

As the final step, the unified highlighter will extract from the field's text a string corresponding to each passage:

```
"I'll be the only fox in the world for you."
```

and will format with the tags and all matches in this string using the passages's `matchStarts` and `matchEnds` information:

```
I'll be the <em>only</em> <em>fox</em> in the world for you.
```

This kind of formatted strings are the final result of the highlighter returned to the user.

« Post filter Rescoring »