Simple query syntax in Azure Cognitive Search

04/24/2020 • 7 minutes to read •

In this article

Invoke simple parsing

Syntax fundamentals

Boolean search

Wildcard prefix matching (*, ?)

Phrase search "

See also

Azure Cognitive Search implements two Lucene-based query languages: Simple Query Parser and the Lucene Query Parser.

The simple parser is more flexible and will attempt to interpret a request even if it's not perfectly composed. Because of this flexibility, it is the default for queries in Azure Cognitive Search.

The simple syntax is used for query expressions passed in the `search` parameter of a Search Documents request, not to be confused with the OData syntax used for the $filter expressions parameter of the same Search Documents API. The `search` and `$filter` parameters have different syntax, with their own rules for constructing queries, escaping strings, and so on.

Although the simple parser is based on the Apache Lucene Simple Query Parser class, the implementation in Azure Cognitive Search excludes fuzzy search. If you need fuzzy search or other advanced query forms, consider the alternative full Lucene query syntax instead.

Invoke simple parsing

Simple syntax is the default. Invocation is only necessary if you are resetting the syntax from full to simple. To explicitly set the syntax, use the `queryType` search parameter. Valid values include `queryType=simple` or `queryType=full`, where `simple` is the default, and `full` invokes the full Lucene query parser for more advanced queries.

Syntax fundamentals

Any text with one or more terms is considered a valid starting point for query execution. Azure Cognitive Search will match documents containing any or all of the terms, including any variations found during analysis of the text.

As straightforward as this sounds, there is one aspect of query execution in Azure Cognitive Search that *might* produce unexpected results, increasing rather than decreasing search results as more terms and operators are added to the input string. Whether this expansion actually occurs depends on the inclusion of a NOT operator, combined with a **searchMode** parameter setting that determines how NOT is interpreted in terms of AND or OR behaviors. For more information, see NOT operator.

Precedence operators (grouping)

You can use parentheses to create subqueries, including operators within the parenthetical statement. For example, `motel+(wifi|luxury)` will search for documents containing the "motel" term and either "wifi" or "luxury" (or both).

Field grouping is similar but scopes the grouping to a single field. For example, `hotelAmenities:(gym+(wifi|pool))` searches the field "hotelAmenities" for "gym" and "wifi", or "gym" and "pool".

Escaping search operators

In the simple syntax, search operators include these characters: `+ | " ( ) ' \`

If any of these characters are part of a token in the index, escape it by prefixing it with a single backslash (`\`) in the query. For example, suppose you used a custom analyzer for whole term tokenization, and your index contains the string

"Luxury+Hotel". To get an exact match on this token, insert an escape character: `search=luxury\+hotel`.

To make things simple for the more typical cases, there are two exceptions to this rule where escaping is not needed:

- The NOT operator `-` only needs to be escaped if it's the first character after a whitespace. If the `-` appears in the middle (for example, in `3352CDD0-EF30-4A2E-A512-3B30AF40F3FD`), you can skip escaping.
- The suffix operator `*` only needs to be escaped if it's the last character before a whitespace. If the `*` appears in the middle (for example, in `4*4=16`), no escaping is needed.

> **Note**
>
> By default, the standard analyzer will delete and break words on hyphens, whitespace, ampersands, and other characters during **lexical analysis**. If you require special characters to remain in the query string, you might need an analyzer that preserves them in the index. Some choices include Microsoft natural **language analyzers**, which preserves hyphenated words, or a custom analyzer for more complex patterns. For more information, see **Partial terms, patterns, and special characters**.

Encoding unsafe and reserved characters in URLs

Please ensure all unsafe and reserved characters are encoded in a URL. For example, '#' is an unsafe character because it is a fragment/anchor identifier in a URL. The character must be encoded to `%23` if used in a URL. '&' and '=' are examples of reserved characters as they delimit parameters and specify values in Azure Cognitive Search. Please see RFC1738: Uniform Resource Locators (URL) for more details.

Unsafe characters are ``" ` < > # % { } | \ ^ ~ [ ]``. Reserved characters are `; / ? : @ = + &`.

Querying for special characters

In some circumstances, you may want to search for a special character, like the '❤' emoji or the '€' sign. In those case, make sure that the analyzer you use does not filter those characters out. The standard analyzer ignores many of the special characters so they would not become tokens in your index.

So the first step is to make sure you use an analyzer that will consider those elements tokens. For instance, the "whitespace" analyzer takes into consideration any character sequences separated by whitespaces as tokens, so the "❤" string would be considered a token. Also, an analyzer like the Microsoft English analyzer ("en.microsoft"), would take into consideration the "€" string as a token. You can test an analyzer to see what tokens it generates for a given query.

When using Unicode characters, make sure symbols are properly escaped in the query url (for instance for "❤" would use the escape sequence `%E2%9D%A4+`). Postman does this translation automatically.

Query size limits

There is a limit to the size of queries that you can send to Azure Cognitive Search. Specifically, you can have at most 1024 clauses (expressions separated by AND, OR, and so on). There is also a limit of approximately 32 KB on the size of any individual term in a query. If your application generates search queries programmatically, we recommend designing it in such a way that it does not generate queries of unbounded size.

Boolean search

You can embed Boolean operators (AND, OR, NOT) in a query string to build a rich set of criteria against which matching documents are found.

AND operator `+`

The AND operator is a plus sign. For example, `wifi + luxury` will search for documents containing both `wifi` and `luxury`.

OR operator `|`

The OR operator is a vertical bar or pipe character. For example, `wifi | luxury` will search for documents containing either `wifi` or `luxury` or both.

NOT operator `-`

The NOT operator is a minus sign. For example, `wifi –luxury` will search for documents that have the `wifi` term and/or do not have `luxury`.

The **searchMode** parameter on a query request controls whether a term with the NOT operator is ANDed or ORed with other terms in the query (assuming there is no `+` or `|` operator on the other terms). Valid values include `any` or `all`.

`searchMode=any` increases the recall of queries by including more results, and by default `-` will be interpreted as "OR NOT". For example, `wifi -luxury` will match documents that either contain the term `wifi` or those that do not contain the term `luxury`.

`searchMode=all` increases the precision of queries by including fewer results, and by default - will be interpreted as "AND NOT". For example, `wifi -luxury` will match documents that contain the term `wifi` and do not contain the term "luxury". This is arguably a more intuitive behavior for the `-` operator. Therefore, you should consider using `searchMode=all` instead of `searchMode=any` if you want to optimize searches for precision instead of recall, *and* Your users frequently use the `-` operator in searches.

When deciding on a **searchMode** setting, consider the user interaction patterns for queries in various applications. Users who are searching for information are more likely to include an operator in a query, as opposed to e-commerce sites that have more built-in navigation structures.

Wildcard prefix matching (*, ?)

For "starts with" queries, add a suffix operator as the placeholder for the remainder of a term. Use an asterisk `*` for multiple characters or `?` for single characters. For example, `lingui*` will match on "linguistic" or "linguini", ignoring case.

Similar to filters, a prefix query looks for an exact match. As such, there is no relevance scoring (all results receive a search score of 1.0). Be aware that prefix queries can be slow, especially if the index is large and the prefix consists of a small number of characters. An alternative methodology, such as edge n-gram tokenization, might perform faster.

For other wildcard query variants, such as suffix or infix matching against the end or middle of a term, use the full Lucene syntax for wildcard search.

Phrase search `"`

A term search is a query for one or more terms, where any of the terms are considered a match. A phrase search is an exact phrase enclosed in quotation marks `" "`. For example, while `Roach Motel` (without quotes) would search for documents containing `Roach` and/or `Motel` anywhere in any order, `"Roach Motel"` (with quotes) will only match documents that contain that whole phrase together and in that order (lexical analysis still applies).

See also

- How full text search works in Azure Cognitive Search
- Query examples for simple search
- Query examples for full Lucene search
- Search Documents REST API
- Lucene query syntax
- OData expression syntax