Analyzers for text processing in Azure Cognitive Search

06/20/2020 • 10 minutes to read • +3

In this article

An *analyzer* is a component of the full text search engine responsible for processing text in query strings and indexed documents. Text processing (also known as lexical analysis) is transformative, modifying a string through actions such as these:

- Remove non-essential words (stopwords) and punctuation
- Split up phrases and hyphenated words into component parts
- Lower-case any upper-case words
- Reduce words into primitive root forms for storage efficiency and so that matches can be found regardless of tense

Analysis applies to `Edm.String` fields that are marked as "searchable", which indicates full text search. For fields with this configuration, analysis occurs during indexing when tokens are created, and then again during query execution when queries are parsed and the engine scans for matching tokens. A match is more likely to occur when the same analyzer is used for both indexing and queries, but you can set the analyzer for each workload independently, depending on your requirements.

Query types that are not full text search, such as regular expression or fuzzy search, do not go through the analysis phase on the query side. Instead, the parser sends those strings directly to the search engine, using the pattern that you provide as the basis for the match. Typically, these query forms require whole-string tokens to make pattern matching work. To get whole terms tokens during indexing, you might need custom analyzers. For more information about when and why query terms are analyzed, see Full text search in Azure Cognitive Search.

For more background on lexical analysis, listen to the following video clip for a brief explanation.

Default analyzer

In Azure Cognitive Search queries, an analyzer is automatically invoked on all string fields marked as searchable.

By default, Azure Cognitive Search uses the Apache Lucene Standard analyzer (standard lucene), which breaks text into elements following the "Unicode Text Segmentation" rules. Additionally, the standard analyzer converts all characters to their lower case form. Both indexed documents and search terms go through the analysis during indexing and query processing.

You can override the default on a field-by-field basis. Alternative analyzers can be a language analyzer for linguistic processing, a custom analyzer, or a predefined analyzer from the list of available analyzers.

Types of analyzers

The following list describes which analyzers are available in Azure Cognitive Search.

Category	Description
Standard Lucene analyzer	Default. No specification or configuration is required. This general-purpose analyzer performs well for many languages and scenarios.
Predefined analyzers	Offered as a finished product intended to be used as-is. There are two types: specialized and language. What makes them "predefined" is that you reference them by name, with no configuration or customization. Specialized (language-agnostic) analyzers are used when text inputs require specialized processing or minimal processing. Non-language predefined analyzers include **Asciifolding**, **Keyword**, **Pattern**, **Simple**, **Stop**, **Whitespace**. Language analyzers are used when you need rich linguistic support for individual languages. Azure Cognitive Search supports 35 Lucene language analyzers and 50 Microsoft natural language processing analyzers.
Custom analyzers	Refers to a user-defined configuration of a combination of existing elements, consisting of one tokenizer (required) and optional filters (char or token).

A few predefined analyzers, such as **Pattern** or **Stop**, support a limited set of configuration options. To set these options, you effectively create a custom analyzer, consisting of the predefined analyzer and one of the alternative options documented in Predefined Analyzer Reference. As with any custom configuration, provide your new configuration with a name, such as *myPatternAnalyzer* to distinguish it from the Lucene Pattern analyzer.

How to specify analyzers

Setting an analyzer is optional. As a general rule, try using the default standard Lucene analyzer first to see how it performs. If queries fail to return the expected results, switching to a different analyzer is often the right solution.

1. When creating a field definition in the index, set the **analyzer** property to one of the following: a predefined analyzer such as `keyword`, a language analyzer such as `en.microsoft`, or a custom analyzer (defined in the same index schema).

   JSON

   ```json
    "fields": [
   {
     "name": "Description",
     "type": "Edm.String",
     "retrievable": true,
     "searchable": true,
     "analyzer": "en.microsoft",
     "indexAnalyzer": null,
     "searchAnalyzer": null
   },
   ```

   If you are using a language analyzer, you must use the **analyzer** property to specify it. The **searchAnalyzer** and **indexAnalyzer** properties do not support language analyzers.

2. Alternatively, set **indexAnalyzer** and **searchAnalyzer** to vary the analyzer for each workload. These properties are set together and replace the **analyzer** property, which must be null. You might use different analyzers for data preparation and retrieval if one of those activities required a specific transformation not needed by the other.

   JSON

   ```json
    "fields": [
   {
     "name": "Description",
   ```

```
    "type": "Edm.String",
    "retrievable": true,
    "searchable": true,
    "analyzer": null,
    "indexAnalyzer": "keyword",
    "searchAnalyzer": "whitespace"
  },
```

3. For custom analyzers only, create an entry in the **[analyzers]** section of the index, and then assign your custom analyzer to the field definition per either of the previous two steps. For more information, see Create Index and also Add custom analyzers.

When to add analyzers

The best time to add and assign analyzers is during active development, when dropping and recreating indexes is routine.

Because analyzers are used to tokenize terms, you should assign an analyzer when the field is created. In fact, assigning **analyzer** or **indexAnalyzer** to a field that has already been physically created is not allowed (although you can change the **searchAnalyzer** property at any time with no impact to the index).

To change the analyzer of an existing field, you'll have to rebuild the index entirely (you cannot rebuild individual fields). For indexes in production, you can defer a rebuild by creating a new field with the new analyzer assignment, and start using it in place of the old one. Use Update Index to incorporate the new field and mergeOrUpload to populate it. Later, as part of planned index servicing, you can clean up the index to remove obsolete fields.

To add a new field to an existing index, call Update Index to add the field, and mergeOrUpload to populate it.

To add a custom analyzer to an existing index, pass the **allowIndexDowntime** flag in Update Index if you want to avoid this error:

"Index update not allowed because it would cause downtime. In order to add new analyzers, tokenizers, token filters, or character filters to an existing index, set the 'allowIndexDowntime' query parameter to 'true' in the index update request. Note

that this operation will put your index offline for at least a few seconds, causing your indexing and query requests to fail. Performance and write availability of the index can be impaired for several minutes after the index is updated, or longer for very large indexes."

Recommendations for working with analyzers

This section offers advice on how to work with analyzers.

One analyzer for read-write unless you have specific requirements

Azure Cognitive Search lets you specify different analyzers for indexing and search via additional **indexAnalyzer** and **searchAnalyzer** field properties. If unspecified, the analyzer set with the **analyzer** property is used for both indexing and searching. If **analyzer** is unspecified, the default Standard Lucene analyzer is used.

A general rule is to use the same analyzer for both indexing and querying, unless specific requirements dictate otherwise. Be sure to test thoroughly. When text processing differs at search and indexing time, you run the risk of mismatch between query terms and indexed terms when the search and indexing analyzer configurations are not aligned.

Test during active development

Overriding the standard analyzer requires an index rebuild. If possible, decide on which analyzers to use during active development, before rolling an index into production.

Inspect tokenized terms

If a search fails to return expected results, the most likely scenario is token discrepancies between term inputs on the query, and tokenized terms in the index. If the tokens aren't the same, matches fail to materialize. To inspect tokenizer output, we recommend using the Analyze API as an investigation tool. The response consists of tokens, as generated by a specific analyzer.

REST examples

The examples below show analyzer definitions for a few key scenarios.

- Custom analyzer example
- Assign analyzers to a field example
- Mixing analyzers for indexing and search
- Language analyzer example

Custom analyzer example

This example illustrates an analyzer definition with custom options. Custom options for char filters, tokenizers, and token filters are specified separately as named constructs, and then referenced in the analyzer definition. Predefined elements are used as-is and simply referenced by name.

Walking through this example:

- Analyzers are a property of the field class for a searchable field.
- A custom analyzer is part of an index definition. It might be lightly customized (for example, customizing a single option in one filter) or customized in multiple places.
- In this case, the custom analyzer is "my_analyzer", which in turn uses a customized standard tokenizer "my_standard_tokenizer" and two token filters: lowercase and customized asciifolding filter "my_asciifolding".
- It also defines 2 custom char filters "map_dash" and "remove_whitespace". The first one replaces all dashes with underscores while the second one removes all spaces. Spaces need to be UTF-8 encoded in the mapping rules. The char filters are applied before tokenization and will affect the resulting tokens (the standard tokenizer breaks on dash and spaces but not on underscore).

JSON

Copy

```json
{
   "name":"myindex",
   "fields":[
      {
         "name":"id",
         "type":"Edm.String",
         "key":true,
         "searchable":false
      },
      {
         "name":"text",
         "type":"Edm.String",
         "searchable":true,
         "analyzer":"my_analyzer"
      }
   ],
   "analyzers":[
      {
         "name":"my_analyzer",
         "@odata.type":"#Microsoft.Azure.Search.CustomAnalyzer",
         "charFilters":[
            "map_dash",
            "remove_whitespace"
         ],
         "tokenizer":"my_standard_tokenizer",
         "tokenFilters":[
            "my_asciifolding",
            "lowercase"
         ]
      }
   ],
   "charFilters":[
      {
         "name":"map_dash",
         "@odata.type":"#Microsoft.Azure.Search.MappingCharFilter",
         "mappings":["-=>_"]
      },
      {
         "name":"remove_whitespace",
         "@odata.type":"#Microsoft.Azure.Search.MappingCharFilter",
         "mappings":["\\u0020=>"]
      }
   ],
   "tokenizers":[
      {
         "name":"my_standard_tokenizer",
         "@odata.type":"#Microsoft.Azure.Search.StandardTokenizerV2",
         "maxTokenLength":20
      }
   ],
   "tokenFilters":[
      {
         "name":"my_asciifolding",
         "@odata.type":"#Microsoft.Azure.Search.AsciiFoldingTokenFilter",
         "preserveOriginal":true
      }
   ]
}
```

Per-field analyzer assignment example

The Standard analyzer is the default. Suppose you want to replace the default with a different predefined analyzer, such as the pattern analyzer. If you are not setting custom options, you only need to specify it by name in the field definition.

The "analyzer" element overrides the Standard analyzer on a field-by-field basis. There is no global override. In this example, `text1` uses the pattern analyzer and `text2`, which doesn't specify an analyzer, uses the default.

JSON Copy

```
   {
      "name":"myindex",
      "fields":[
         {
            "name":"id",
            "type":"Edm.String",
            "key":true,
            "searchable":false
         },
         {
            "name":"text1",
            "type":"Edm.String",
            "searchable":true,
            "analyzer":"pattern"
         },
         {
            "name":"text2",
            "type":"Edm.String",
            "searchable":true
         }
      ]
   }
```

Mixing analyzers for indexing and search operations

The APIs include additional index attributes for specifying different analyzers for indexing and search. The **searchAnalyzer** and **indexAnalyzer** attributes must be specified as a pair, replacing the single **analyzer** attribute.

JSON

```json
 {
    "name":"myindex",
    "fields":[
       {
          "name":"id",
          "type":"Edm.String",
          "key":true,
          "searchable":false
       },
       {
          "name":"text",
          "type":"Edm.String",
          "searchable":true,
          "indexAnalyzer":"whitespace",
          "searchAnalyzer":"simple"
       },
    ],
 }
```

Language analyzer example

Fields containing strings in different languages can use a language analyzer, while other fields retain the default (or use some other predefined or custom analyzer). If you use a language analyzer, it must be used for both indexing and search operations. Fields that use a language analyzer cannot have different analyzers for indexing and search.

JSON

```
   {
      "name":"myindex",
      "fields":[
         {
            "name":"id",
            "type":"Edm.String",
            "key":true,
            "searchable":false
         },
         {
            "name":"text",
            "type":"Edm.String",
            "searchable":true,
            "indexAnalyzer":"whitespace",
            "searchAnalyzer":"simple"
```

```
          },
          {
             "name":"text_fr",
             "type":"Edm.String",
             "searchable":true,
             "analyzer":"fr.lucene"
          }
       ],
    }
```

C# examples

If you are using the .NET SDK code samples, you can append these examples to use or configure analyzers.

- Assign a built-in analyzer
- Configure an analyzer

Assign a language analyzer

Any analyzer that is used as-is, with no configuration, is specified on a field definition. There is no requirement for creating an entry in the **[analyzers]** section of the index.

This example assigns Microsoft English and French analyzers to description fields. It's a snippet taken from a larger definition of the hotels index, creating using the Hotel class in the hotels.cs file of the DotNetHowTo sample.

Call LexicalAnalyzer, specifying the LexicalAnalyzerName type providing a text analyzer supported in Azure Cognitive Search.

C#

```csharp
    public partial class Hotel
    {
       . . .
```

```
        [IsSearchable]
        [Analyzer(AnalyzerName.AsString.EnMicrosoft)]
        [JsonProperty("description")]
        public string Description { get; set; }

        [IsSearchable]
        [Analyzer(AnalyzerName.AsString.FrLucene)]
        [JsonProperty("description_fr")]
        public string DescriptionFr { get; set; }

      . . .
    }
```

Define a custom analyzer

When customization or configuration is required, you will need to add an analyzer construct to an index. Once you define it, you can add it the field definition as demonstrated in the previous example.

Create a CustomAnalyzer object. For more examples, see CustomAnalyzerTests.cs.

C#   Copy

```
{
   var definition = new Index()
   {
         Name = "hotels",
         Fields = FieldBuilder.BuildForType<Hotel>(),
         Analyzers = new[]
            {
               new CustomAnalyzer()
               {
                     Name = "url-analyze",
                     Tokenizer = TokenizerName.UaxUrlEmail,
                     TokenFilters = new[] { TokenFilterName.Lowercase }
            }
        },
    };

    serviceClient.Indexes.Create(definition);
```

Next steps

- Review our comprehensive explanation of how full text search works in Azure Cognitive Search. This article uses examples to explain behaviors that might seem counter-intuitive on the surface.
- Try additional query syntax from the Search Documents example section or from Simple query syntax in Search explorer in the portal.
- Learn how to apply language-specific lexical analyzers.
- Configure custom analyzers for either minimal processing or specialized processing on individual fields.

See also

Search Documents REST API

Simple query syntax

Full Lucene query syntax

Handle search results