# Add custom analyzers to string fields in an Azure AI Search index

A *custom analyzer* is a component of lexical analysis over plain text content. It's a user-defined combination of one tokenizer, one or more token filters, and one or more character filters. A custom analyzer is specified within a search index, and then referenced by name on field definitions that require custom analysis. A custom analyzer is invoked on a per-field basis. Attributes on the field determine whether it's used for indexing, queries, or both.

In a custom analyzer, character filters prepare the input text before it's processed by the tokenizer (for example, removing markup). Next, the tokenizer breaks text into tokens. Finally, token filters modify the tokens emitted by the tokenizer. For concepts and examples, see Analyzers in Azure AI Search and Tutorial: Create a custom analyzer for phone numbers.

## Why use a custom analyzer?

A custom analyzer gives you control over the process of converting plain text into indexable and searchable tokens by allowing you to choose which types of analysis or filtering to invoke, and the order in which they occur.

Create and assign a custom analyzer if none of default (Standard Lucene), built-in, or language analyzers are sufficient for your needs. You might also create a custom analyzer if you want to use a built-in analyzer with custom options. For example, if you wanted to change the `maxTokenLength` on Standard Lucene, you would create a custom analyzer, with a user-defined name, to set that option.

Scenarios where custom analyzers can be helpful include:

- Using character filters to remove HTML markup before text inputs are tokenized, or replace certain characters or symbols.
- Phonetic search. Add a phonetic filter to enable searching based on how a word sounds, not how it’s spelled.
- Disable lexical analysis. Use the Keyword analyzer to create searchable fields that aren't analyzed.
- Fast prefix/suffix search. Add the Edge N-gram token filter to index prefixes of words to enable fast prefix matching. Combine it with the Reverse token filter to do suffix matching.
- Custom tokenization. For example, use the Whitespace tokenizer to break sentences into tokens using whitespace as a delimiter
- ASCII folding. Add the Standard ASCII folding filter to normalize diacritics like ö or ê in search terms.

> [!NOTE]
> Custom analyzers aren't exposed in the Azure portal. The only way to add a custom analyzer is through code that creates an index schema.

## Create a custom analyzer

To create a custom analyzer, specify it in the `analyzers` section of an index at design time, and then reference it on searchable, `Edm.String` fields using either the `analyzer` property, or the `indexAnalyzer` and `searchAnalyzer` pair.

An analyzer definition includes a name, type, one or more character filters, a maximum of one tokenizer, and one or more token filters for post-tokenization processing. Character filters are applied before tokenization. Token filters and character filters are applied from left to right.

- Names in a custom analyzer must be unique and can't be the same as any of the built-in analyzers, tokenizers, token filters, or characters filters. Names consist of letters, digits, spaces, dashes, or underscores. Names must start and end with plain text characters. Names must be under 128 characters in length.
- Type must be #Microsoft.Azure.Search.CustomAnalyzer.
- `charFilters` can be one or more filters from Character Filters, processed before tokenization, in the order provided. Some character filters have options, which can be set in a `charFilters` section. Character filters are optional.
- `tokenizer` is exactly one Tokenizer. A value is required. If you need more than one tokenizer, you can create multiple custom analyzers and assign them on a field-by-field basis in your index schema.
- `tokenFilters` can be one or more filters from Token Filters, processed after tokenization, in the order provided. For token filters that have options, add a `tokenFilter` section to specify the configuration. Token filters are optional.

Analyzers must not produce tokens longer than 300 characters, or indexing will fail. To trim long token or to exclude them, use the **TruncateTokenFilter** and the **LengthTokenFilter** respectively. See **Token filters** for reference.

```
"analyzers":(optional)[
   {
      "name":"name of analyzer",
      "@odata.type":"#Microsoft.Azure.Search.CustomAnalyzer",
      "charFilters":[
         "char_filter_name_1",
         "char_filter_name_2"
      ],
      "tokenizer":"tokenizer_name",
      "tokenFilters":[
         "token_filter_name_1",
         "token_filter_name_2"
      ]
   },
   {
      "name":"name of analyzer",
      "@odata.type":"#analyzer_type",
      "option1":value1,
      "option2":value2,
      ...
   }
```

```
  ],
  "charFilters":(optional)[
     {
        "name":"char_filter_name",
        "@odata.type":"#char_filter_type",
        "option1":value1,
        "option2":value2,
        ...
     }
  ],
  "tokenizers":(optional)[
     {
        "name":"tokenizer_name",
        "@odata.type":"#tokenizer_type",
        "option1":value1,
        "option2":value2,
        ...
     }
  ],
  "tokenFilters":(optional)[
     {
        "name":"token_filter_name",
        "@odata.type":"#token_filter_type",
        "option1":value1,
        "option2":value2,
        ...
     }
  ]
```

Within an index definition, you can place this section anywhere in the body of a create index request but usually it goes at the end:

```
{
  "name": "name_of_index",
  "fields": [ ],
  "suggesters": [ ],
  "scoringProfiles": [ ],
  "defaultScoringProfile": (optional) "...",
  "corsOptions": (optional) { },
  "analyzers":(optional)[ ],
  "charFilters":(optional)[ ],
  "tokenizers":(optional)[ ],
  "tokenFilters":(optional)[ ]
}
```

The analyzer definition is a part of the larger index. Definitions for char filters, tokenizers, and token filters are added to the index only if you're setting custom options. To use an existing filter or tokenizer as-is, specify it by name in the analyzer definition. For more information, see Create Index (REST). For more examples, see Add analyzers in Azure AI Search.

## Test custom analyzers

You can use the Test Analyzer (REST) to see how an analyzer breaks given text into tokens.

**Request**

```
  POST https://[search service name].search.windows.net/indexes/[index
name]/analyze?api-version=[api-version]
    Content-Type: application/json
    api-key: [admin key]

  {
     "analyzer":"my_analyzer",
     "text": "Vis-à-vis means Opposite"
  }
```

**Response**

```
  {
    "tokens": [
      {
        "token": "vis_a_vis",
        "startOffset": 0,
        "endOffset": 9,
        "position": 0
      },
      {
        "token": "vis_à_vis",
        "startOffset": 0,
        "endOffset": 9,
        "position": 0
      },
      {
        "token": "means",
        "startOffset": 10,
        "endOffset": 15,
        "position": 1
      },
      {
        "token": "opposite",
        "startOffset": 16,
        "endOffset": 24,
        "position": 2
      }
    ]
  }
```

## Update custom analyzers

Once an analyzer, a tokenizer, a token filter, or a character filter is defined, it can't be modified. New ones can be added to an existing index only if the `allowIndexDowntime` flag is set to true in the index update request:

```
PUT https://[search service name].search.windows.net/indexes/[index name]?api-version=[api-version]&allowIndexDowntime=true
```

This operation takes 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, but these effects are temporary and eventually resolve on their own.

## Built-in analyzers

If you want to use a built-in analyzer with custom options, creating a custom analyzer is the mechanism by which you specify those options. In contrast, to use a built-in analyzer as-is, you simply need to reference it by name in the field definition.

| **analyzer_name** | **analyzer_type** [1] | **Description and Options** |
|---|---|---|
| keyword | (type applies only when options are available) | Treats the entire content of a field as a single token. This is useful for data like zip codes, IDs, and some product names. |
| pattern | PatternAnalyzer | Flexibly separates text into terms via a regular expression pattern.<br><br>**Options**<br><br>lowercase (type: bool) - Determines whether terms are lowercased. The default is true.<br><br>pattern (type: string) - A regular expression pattern to match token separators. The default is `\W+`, which matches non-word characters.<br><br>flags (type: string) - Regular expression flags. The default is an empty string. Allowed values: CANON_EQ, CASE_INSENSITIVE, COMMENTS, DOTALL, LITERAL, MULTILINE, UNICODE_CASE, UNIX_LINES<br><br>stopwords (type: string array) - A list of stopwords. The default is an empty list. |
| simple | (type applies only when options are available) | Divides text at non-letters and converts them to lower case. |

| analyzer_name | analyzer_type [1] | Description and Options |
|---|---|---|
| standard<br>(Also referred to as standard.lucene) | StandardAnalyzer | Standard Lucene analyzer, composed of the standard tokenizer, lowercase filter, and stop filter.<br><br>**Options**<br><br>maxTokenLength (type: int) - The maximum token length. The default is 255. Tokens longer than the maximum length are split. Maximum token length that can be used is 300 characters.<br><br>stopwords (type: string array) - A list of stopwords. The default is an empty list. |
| standardasciifolding.lucene | (type applies only when options are available) | Standard analyzer with Ascii folding filter. |
| stop | StopAnalyzer | Divides text at non-letters, applies the lowercase and stopword token filters.<br><br>**Options**<br><br>stopwords (type: string array) - A list of stopwords. The default is a predefined list for English. |
| whitespace | (type applies only when options are available) | An analyzer that uses the whitespace tokenizer. Tokens that are longer than 255 characters are split. |

[1] Analyzer Types are always prefixed in code with `#Microsoft.Azure.Search` such that `PatternAnalyzer` would actually be specified as `#Microsoft.Azure.Search.PatternAnalyzer`. We removed the prefix for brevity, but the prefix is required in your code.

The analyzer_type is only provided for analyzers that can be customized. If there are no options, as is the case with the keyword analyzer, there's no associated #Microsoft.Azure.Search type.

## Character filters

Character filters add processing before a string reaches the tokenizer.

Azure AI Search supports character filters in the following list. More information about each one can be found in the Lucene API reference.

| char_filter_name | char_filter_type [1] | Description and Options |
|---|---|---|
| html_strip | (type applies only when options are available) | A char filter that attempts to strip out HTML constructs. |

| char_filter_name | char_filter_type [1] | Description and Options |
|---|---|---|
| mapping | MappingCharFilter | A char filter that applies mappings defined with the mappings option. Matching is greedy (longest pattern matching at a given point wins). Replacement is allowed to be the empty string.<br><br>**Options**<br><br>mappings (type: string array) - A list of mappings of the following format: `a=>b` (all occurrences of the character `a` are replaced with character `b`). Required. |
| pattern_replace | PatternReplaceCharFilter | A char filter that replaces characters in the input string. It uses a regular expression to identify character sequences to preserve and a replacement pattern to identify characters to replace. For example, input text = `aa bb aa bb`, pattern=`(aa)\\\s+(bb)` replacement=`$1#$2`, result = `aa#bb aa#bb`.<br><br>**Options**<br><br>pattern (type: string) - Required.<br><br>replacement (type: string) - Required. |

[1] Char Filter Types are always prefixed in code with `#Microsoft.Azure.Search` such that `MappingCharFilter` would actually be specified as `#Microsoft.Azure.Search.MappingCharFilter`. We removed the prefix to reduce the width of the table, but remember to include it in your code. Notice that char_filter_type is only provided for filters that can be customized. If there are no options, as is the case with html_strip, there's no associated #Microsoft.Azure.Search type.

## Tokenizers

A tokenizer divides continuous text into a sequence of tokens, such as breaking a sentence into words, or a word into root forms.

Azure AI Search supports tokenizers in the following list. More information about each one can be found in the Lucene API reference.

| tokenizer_name | tokenizer_type [1] | Description and Options |
|---|---|---|

| tokenizer_name | tokenizer_type [1] | Description and Options |
|---|---|---|
| classic | ClassicTokenizer | Grammar based tokenizer that is suitable for processing most European-language documents.<br><br>**Options**<br><br>maxTokenLength (type: int) - The maximum token length. Default: 255, maximum: 300. Tokens longer than the maximum length are split. |
| edgeNGram | EdgeNGramTokenizer | Tokenizes the input from an edge into n-grams of given sizes.<br><br>**Options**<br><br>minGram (type: int) - Default: 1, maximum: 300.<br><br>maxGram (type: int) - Default: 2, maximum: 300. Must be greater than minGram.<br><br>tokenChars (type: string array) - Character classes to keep in the tokens. Allowed values: `letter`, `digit`, `whitespace`, `punctuation`, `symbol`. Defaults to an empty array - keeps all characters. |

| tokenizer_name | tokenizer_type [1] | Description and Options |
|---|---|---|
| keyword_v2 | KeywordTokenizerV2 | Emits the entire input as a single token.<br><br>**Options**<br><br>maxTokenLength (type: int) - The maximum token length. Default: 256, maximum: 300. Tokens longer than the maximum length are split. |
| letter | (type applies only when options are available) | Divides text at non-letters. Tokens that are longer than 255 characters are split. |
| lowercase | (type applies only when options are available) | Divides text at non-letters and converts them to lower case. Tokens that are longer than 255 characters are split. |
| microsoft_language_tokenizer | MicrosoftLanguageTokenizer | Divides text using language-specific rules.<br><br>**Options**<br><br>maxTokenLength (type: int) - The maximum token length, default: 255, maximum: 300. Tokens longer than the maximum length are split. Tokens longer than 300 characters are first split into tokens of length 300 and then each of those tokens is split based on the maxTokenLength set.<br><br>isSearchTokenizer (type: |

| tokenizer_name | tokenizer_type [1] | Description and Options |
|---|---|---|
| | | bool) - Set to true if used as the search tokenizer, set to false if used as the indexing tokenizer.<br><br>language (type: string) - Language to use, default `english`. Allowed values include: `bangla`, `bulgarian`, `catalan`, `chineseSimplified`, `chineseTraditional`, `croatian`, `czech`, `danish`, `dutch`, `english`, `french`, `german`, `greek`, `gujarati`, `hindi`, `icelandic`, `indonesian`, `italian`, `japanese`, `kannada`, `korean`, `malay`, `malayalam`, `marathi`, `norwegianBokmaal`, `polish`, `portuguese`, `portugueseBrazilian`, `punjabi`, `romanian`, `russian`, `serbianCyrillic`, `serbianLatin`, `slovenian`, `spanish`, `swedish`, `tamil`, `telugu`, `thai`, `ukrainian`, `urdu`, `vietnamese` |
| microsoft_language_stemming_tokenizer | MicrosoftLanguageStemmingTokenizer | Divides text using language-specific rules and reduces words to their base forms. This tokenizer performs lemmatization.<br><br>**Options** |

| tokenizer_name | tokenizer_type [1] | Description and Options |
|---|---|---|
| | | maxTokenLength (type: int) - The maximum token length, default: 255, maximum: 300. Tokens longer than the maximum length are split. Tokens longer than 300 characters are first split into tokens of length 300 and then each of those tokens is split based on the maxTokenLength set.<br><br>isSearchTokenizer (type: bool) - Set to true if used as the search tokenizer, set to false if used as the indexing tokenizer.<br><br>language (type: string) - Language to use, default `english`. Allowed values include: `arabic`, `bangla`, `bulgarian`, `catalan`, `croatian`, `czech`, `danish`, `dutch`, `english`, `estonian`, `finnish`, `french`, `german`, `greek`, `gujarati`, `hebrew`, `hindi`, `hungarian`, `icelandic`, `indonesian`, `italian`, `kannada`, `latvian`, `lithuanian`, `malay`, `malayalam`, `marathi`, `norwegianBokmaal`, `polish`, `portuguese`, `portugueseBrazilian`, `punjabi`, `romanian`, |

| tokenizer_name | tokenizer_type [1] | Description and Options |
|---|---|---|
| | | `russian`, `serbianCyrillic`, `serbianLatin`, `slovak`, `slovenian`, `spanish`, `swedish`, `tamil`, `telugu`, `turkish`, `ukrainian`, `urdu` |
| nGram | NGramTokenizer | Tokenizes the input into n-grams of the given sizes.<br><br>**Options**<br><br>minGram (type: int) - Default: 1, maximum: 300.<br><br>maxGram (type: int) - Default: 2, maximum: 300. Must be greater than minGram.<br><br>tokenChars (type: string array) - Character classes to keep in the tokens. Allowed values: `letter`, `digit`, `whitespace`, `punctuation`, `symbol`. Defaults to an empty array - keeps all characters. |

| tokenizer_name | tokenizer_type [1] | Description and Options |
|---|---|---|
| path_hierarchy_v2 | PathHierarchyTokenizerV2 | Tokenizer for path-like hierarchies. **Options**<br><br>delimiter (type: string) - Default: '/.<br><br>replacement (type: string) - If set, replaces the delimiter character. Default same as the value of delimiter.<br><br>maxTokenLength (type: int) - The maximum token length. Default: 300, maximum: 300. Paths longer than maxTokenLength are ignored.<br><br>reverse (type: bool) - If true, generates token in reverse order. Default: false.<br><br>skip (type: bool) - Initial tokens to skip. The default is 0. |

| tokenizer_name | tokenizer_type [1] | Description and Options |
|---|---|---|
| pattern | PatternTokenizer | This tokenizer uses regex pattern matching to construct distinct tokens.<br><br>**Options**<br><br>pattern (type: string) - Regular expression pattern to match token separators. The default is `\W+`, which matches non-word characters.<br><br>flags (type: string) - Regular expression flags. The default is an empty string. Allowed values: CANON_EQ, CASE_INSENSITIVE, COMMENTS, DOTALL, LITERAL, MULTILINE, UNICODE_CASE, UNIX_LINES<br><br>group (type: int) - Which group to extract into tokens. The default is -1 (split). |
| standard_v2 | StandardTokenizerV2 | Breaks text following the Unicode Text Segmentation rules.<br><br>**Options**<br><br>maxTokenLength (type: int) - The maximum token length. Default: 255, maximum: 300. Tokens longer than the maximum length are split. |

| tokenizer_name | tokenizer_type [1] | Description and Options |
|---|---|---|
| uax_url_email | UaxUrlEmailTokenizer | Tokenizes urls and emails as one token.<br><br>**Options**<br><br>maxTokenLength (type: int) - The maximum token length. Default: 255, maximum: 300. Tokens longer than the maximum length are split. |
| whitespace | (type applies only when options are available) | Divides text at whitespace. Tokens that are longer than 255 characters are split. |

[1] Tokenizer Types are always prefixed in code with `#Microsoft.Azure.Search` such that `ClassicTokenizer` would actually be specified as `#Microsoft.Azure.Search.ClassicTokenizer`. We removed the prefix to reduce the width of the table, but remember to include it in your code. Notice that tokenizer_type is only provided for tokenizers that can be customized. If there are no options, as is the case with the letter tokenizer, there's no associated #Microsoft.Azure.Search type.

## Token filters

A token filter is used to filter out or modify the tokens generated by a tokenizer. For example, you can specify a lowercase filter that converts all characters to lowercase. You can have multiple token filters in a custom analyzer. Token filters run in the order in which they're listed.

In the following table, the token filters that are implemented using Apache Lucene are linked to the Lucene API documentation.

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| arabic_normalization | (type applies only when options are available) | A token filter that applies the Arabic normalizer to normalize the orthography. |
| apostrophe | (type applies only when options are available) | Strips all characters after an apostrophe (including the apostrophe itself). |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| asciifolding | AsciiFoldingTokenFilter | Converts alphabetic, numeric, and symbolic Unicode characters which aren't in the first 127 ASCII characters (the `Basic Latin` Unicode block) into their ASCII equivalents, if one exists.<br><br>**Options**<br><br>preserveOriginal (type: bool) - If true, the original token is kept. The default is false. |
| cjk_bigram | CjkBigramTokenFilter | Forms bigrams of CJK terms that are generated from StandardTokenizer.<br><br>**Options**<br><br>ignoreScripts (type: string array) - Scripts to ignore. Allowed values include: `han`, `hiragana`, `katakana`, `hangul`. The default is an empty list.<br><br>outputUnigrams (type: bool) - Set to true if you always want to output both unigrams and bigrams. The default is false. |
| cjk_width | (type applies only when options are available) | Normalizes CJK width differences. Folds full width ASCII variants into the equivalent basic Latin and half-width Katakana variants into the equivalent kana. |
| classic | (type applies only when options are available) | Removes the English possessives, and dots from acronyms. |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| common_grams | CommonGramTokenFilter | Construct bigrams for frequently occurring terms while indexing. Single terms are still indexed too, with bigrams overlaid.<br><br>**Options**<br><br>commonWords (type: string array) - The set of common words. The default is an empty list. Required.<br><br>ignoreCase (type: bool) - If true, matching is case insensitive. The default is false.<br><br>queryMode (type: bool) - Generates bigrams then removes common words and single terms followed by a common word. The default is false. |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| dictionary_decompounder | DictionaryDecompounderTokenFilter | Decomposes compound words found in many Germanic languages.<br><br>**Options**<br><br>wordList (type: string array) - The list of words to match against. The default is an empty list. Required.<br><br>minWordSize (type: int) - Only words longer than this will be processed. The default is 5.<br><br>minSubwordSize (type: int) - Only subwords longer than this will be outputted. The default is 2.<br><br>maxSubwordSize (type: int) - Only subwords shorter than this will be outputted. The default is 15.<br><br>onlyLongestMatch (type: bool) - Add only the longest matching subword to output. The default is false. |
| edgeNGram_v2 | EdgeNGramTokenFilterV2 | Generates n-grams of the given sizes from starting from the front or the back of an input token.<br><br>**Options**<br><br>minGram (type: int) - Default: 1, maximum: 300.<br><br>maxGram (type: int) - Default: 2, maximum 300. Must be greater than minGram.<br><br>side (type: string) - Specifies which side of the input the n-gram should be generated from. Allowed values: `front`, `back` |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| elision | ElisionTokenFilter | Removes elisions. For example, `l'avion` (the plane) is converted to `avion` (plane).<br><br>**Options**<br><br>articles (type: string array) - A set of articles to remove. The default is an empty list. If there's no list of articles set, by default all French articles are removed. |
| german_normalization | (type applies only when options are available) | Normalizes German characters according to the heuristics of the German2 snowball algorithm . |
| hindi_normalization | (type applies only when options are available) | Normalizes text in Hindi to remove some differences in spelling variations. |
| indic_normalization | IndicNormalizationTokenFilter | Normalizes the Unicode representation of text in Indian languages. |
| keep | KeepTokenFilter | A token filter that only keeps tokens with text contained in specified list of words.<br><br>**Options**<br><br>keepWords (type: string array) - A list of words to keep. The default is an empty list. Required.<br><br>keepWordsCase (type: bool) - If true, lower case all words first. The default is false. |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| keyword_marker | KeywordMarkerTokenFilter | Marks terms as keywords.<br><br>**Options**<br><br>keywords (type: string array) - A list of words to mark as keywords. The default is an empty list. Required.<br><br>ignoreCase (type: bool) - If true, lower case all words first. The default is false. |
| keyword_repeat | (type applies only when options are available) | Emits each incoming token twice once as keyword and once as non-keyword. |
| kstem | (type applies only when options are available) | A high-performance `kstem` filter for English. |
| length | LengthTokenFilter | Removes words that are too long or too short.<br><br>**Options**<br><br>min (type: int) - The minimum number. Default: 0, maximum: 300.<br><br>max (type: int) - The maximum number. Default: 300, maximum: 300. |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| limit | Microsoft.Azure.Search.LimitTokenFilter | Limits the number of tokens while indexing.<br><br>**Options**<br><br>maxTokenCount (type: int) - Max number of tokens to produce. The default is 1.<br><br>consumeAllTokens (type: bool) - Whether all tokens from the input must be consumed even if maxTokenCount is reached. The default is false. |
| lowercase | (type applies only when options are available) | Normalizes token text to lower case. |
| nGram_v2 | NGramTokenFilterV2 | Generates n-grams of the given sizes.<br><br>**Options**<br><br>minGram (type: int) - Default: 1, maximum: 300.<br><br>maxGram (type: int) - Default: 2, maximum 300. Must be greater than minGram. |
| pattern_capture | PatternCaptureTokenFilter | Uses Java regexes to emit multiple tokens, one for each capture group in one or more patterns.<br><br>**Options**<br><br>patterns (type: string array) - A list of patterns to match against each token. Required.<br><br>preserveOriginal (type: bool) - Set to true to return the original token even if one of the patterns matches, default: true |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| pattern_replace | PatternReplaceTokenFilter | A token filter which applies a pattern to each token in the stream, replacing match occurrences with the specified replacement string.<br><br>**Options**<br><br>pattern (type: string) - Required.<br><br>replacement (type: string) - Required. |
| persian_normalization | (type applies only when options are available) | Applies normalization for Persian. |
| phonetic | PhoneticTokenFilter | Create tokens for phonetic matches.<br><br>**Options**<br><br>encoder (type: string) - Phonetic encoder to use. Allowed values include: `metaphone`, `doubleMetaphone`, `soundex`, `refinedSoundex`, `caverphone1`, `caverphone2`, `cologne`, `nysiis`, `koelnerPhonetik`, `haasePhonetik`, `beiderMorse`. Default: `metaphone`. Default is metaphone.<br><br>See encoder for more information.<br><br>replace (type: bool) - True if encoded tokens should replace original tokens, false if they should be added as synonyms. The default is true. |
| porter_stem | (type applies only when options are available) | Transforms the token stream as per the Porter stemming algorithm. |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| reverse | (type applies only when options are available) | Reverses the token string. |
| scandinavian_normalization | (type applies only when options are available) | Normalizes use of the interchangeable Scandinavian characters. |
| scandinavian_folding | (type applies only when options are available) | Folds Scandinavian characters åÅäæÄÆinto a and öÖøØinto o. It also discriminates against use of double vowels aa, ae, ao, oe and oo, leaving just the first one. |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| shingle | ShingleTokenFilter | Creates combinations of tokens as a single token. **Options** maxShingleSize (type: int) - Defaults to 2. minShingleSize (type: int) - Defaults to 2. outputUnigrams (type: bool) - if true, the output stream contains the input tokens (unigrams) as well as shingles. The default is true. outputUnigramsIfNoShingles (type: bool) - If true, override the behavior of outputUnigrams==false for those times when no shingles are available. The default is false. tokenSeparator (type: string) - The string to use when joining adjacent tokens to form a shingle. The default is a single empty space ` `. filterToken (type: string) - The string to insert for each position for which there's no token. The default is `_`. |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| snowball | SnowballTokenFilter | Snowball Token Filter.<br><br>**Options**<br><br>language (type: string) - Allowed values include: `armenian`, `basque`, `catalan`, `danish`, `dutch`, `english`, `finnish`, `french`, `german`, `german2`, `hungarian`, `italian`, `kp`, `lovins`, `norwegian`, `porter`, `portuguese`, `romanian`, `russian`, `spanish`, `swedish`, `turkish` |
| sorani_normalization | SoraniNormalizationTokenFilter | Normalizes the Unicode representation of `Sorani` text.<br><br>**Options**<br><br>None. |
| stemmer | StemmerTokenFilter | Language-specific stemming filter.<br><br>**Options**<br><br>language (type: string) - Allowed values include:<br>- `arabic`<br>- `armenian`<br>- `basque`<br>- `brazilian`<br>- `bulgarian`<br>- `catalan`<br>- `czech`<br>- `danish`<br>- `dutch`<br>- `dutchKp`<br>- `english`<br>- `lightEnglish`<br>- `minimalEnglish`<br>- `possessiveEnglish`<br>- `porter2`<br>- `lovins`<br>- `finnish` |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| | | - `lightFinnish`<br>- `french`<br>- `lightFrench`<br>- `minimalFrench`<br>- `galician`<br>- `minimalGalician`<br>- `german`<br>- `german2`<br>- `lightGerman`<br>- `minimalGerman`<br>- `greek`<br>- `hindi`<br>- `hungarian`<br>- `lightHungarian`<br>- `indonesian`<br>- `irish`<br>- `italian`<br>- `lightItalian`<br>- `sorani`<br>- `latvian`<br>- `norwegian`<br>- `lightNorwegian`<br>- `minimalNorwegian`<br>- `lightNynorsk`<br>- `minimalNynorsk`<br>- `portuguese`<br>- `lightPortuguese`<br>- `minimalPortuguese`<br>- `portugueseRslp`<br>- `romanian`<br>- `russian`<br>- `lightRussian`<br>- `spanish`<br>- `lightSpanish`<br>- `swedish`<br>- `lightSwedish`<br>- `turkish` |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| | | Any dictionary-Stemmed terms are marked as keywords, which prevents stemming down the chain. Must be placed before any stemming filters. |
| stemmer_override | StemmerOverrideTokenFilter | **Options**<br><br>rules (type: string array) - Stemming rules in the following format `word => stem` for example `ran => run`. The default is an empty list. Required. |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| stopwords | StopwordsTokenFilter | Removes stop words from a token stream. By default, the filter uses a predefined stop word list for English.<br><br>**Options**<br><br>stopwords (type: string array) - A list of stopwords. Can't be specified if a stopwordsList is specified.<br><br>stopwordsList (type: string) - A predefined list of stopwords. Can't be specified if `stopwords` is specified. Allowed values include:`arabic`, `armenian`, `basque`, `brazilian`, `bulgarian`, `catalan`, `czech`, `danish`, `dutch`, `english`, `finnish`, `french`, `galician`, `german`, `greek`, `hindi`, `hungarian`, `indonesian`, `irish`, `italian`, `latvian`, `norwegian`, `persian`, `portuguese`, `romanian`, `russian`, `sorani`, `spanish`, `swedish`, `thai`, `turkish`, default: `english`. Can't be specified if `stopwords` is specified.<br><br>ignoreCase (type: bool) - If true, all words are lower cased first. The default is false.<br><br>removeTrailing (type: bool) - If true, ignore the last search term if it's a stop word. The default is true. |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| synonym | SynonymTokenFilter | Matches single or multi word synonyms in a token stream.<br><br>**Options**<br><br>synonyms (type: string array) - Required. List of synonyms in one of the following two formats:<br><br>-incredible, unbelievable, fabulous => amazing - all terms on the left side of => symbol are replaced with all terms on its right side.<br><br>-incredible, unbelievable, fabulous, amazing - A comma-separated list of equivalent words. Set the expand option to change how this list is interpreted.<br><br>ignoreCase (type: bool) - Case-folds input for matching. The default is false.<br><br>expand (type: bool) - If true, all words in the list of synonyms (if => notation isn't used) map to one another.<br>The following list: incredible, unbelievable, fabulous, amazing is equivalent to: incredible, unbelievable, fabulous, amazing => incredible, unbelievable, fabulous, amazing<br><br>- If false, the following list: incredible, unbelievable, fabulous, amazing are equivalent to: incredible, unbelievable, fabulous, amazing => incredible. |
| trim | (type applies only when options are available) | Trims leading and trailing whitespace from tokens. |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| truncate | TruncateTokenFilter | Truncates the terms into a specific length.<br><br>**Options**<br><br>length (type: int) - Default: 300, maximum: 300. Required. |
| unique | UniqueTokenFilter | Filters out tokens with same text as the previous token.<br><br>**Options**<br><br>onlyOnSamePosition (type: bool) - If set, remove duplicates only at the same position. The default is true. |
| uppercase | (type applies only when options are available) | Normalizes token text to upper case. |
| word_delimiter | WordDelimiterTokenFilter | Splits words into subwords and performs optional transformations on subword groups.<br><br>**Options**<br><br>generateWordParts (type: bool) - Causes parts of words to be generated, for example `AzureSearch` becomes `Azure` `Search`. The default is true.<br><br>generateNumberParts (type: bool) - Causes number subwords to be generated. The default is true.<br><br>catenateWords (type: bool) - Causes maximum runs of word parts to be catenated, for example `Azure-Search` becomes `AzureSearch`. The default is false.<br><br>catenateNumbers (type: bool) - Causes maximum runs of number |

| token_filter_name | token_filter_type [1] | Description and Options |
|---|---|---|
| | | parts to be catenated, for example `1-2` becomes `12`. The default is false.<br><br>catenateAll (type: bool) - Causes all subword parts to be catenated, e.g `Azure-Search-1` becomes `AzureSearch1`. The default is false.<br><br>splitOnCaseChange (type: bool) - If true, splits words on caseChange, for example `AzureSearch` becomes `Azure Search`. The default is true.<br><br>preserveOriginal - Causes original words to be preserved and added to the subword list. The default is false.<br><br>splitOnNumerics (type: bool) - If true, splits on numbers, for example `Azure1Search` becomes `Azure 1 Search`. The default is true.<br><br>stemEnglishPossessive (type: bool) - Causes trailing `'s` to be removed for each subword. The default is true.<br><br>protectedWords (type: string array) - Tokens to protect from being delimited. The default is an empty list. |

[1] Token Filter Types are always prefixed in code with `#Microsoft.Azure.Search` such that `ArabicNormalizationTokenFilter` would actually be specified as `#Microsoft.Azure.Search.ArabicNormalizationTokenFilter`. We removed the prefix to reduce the width of the table, but remember to include it in your code.

## See also

- Azure AI Search REST APIs

- Analyzers in Azure AI Search (Examples)
- Create Index (REST)