Create a suggester to enable autocomplete and suggested results in a query

11/24/2020 • 7 minutes to read • +1

In this article

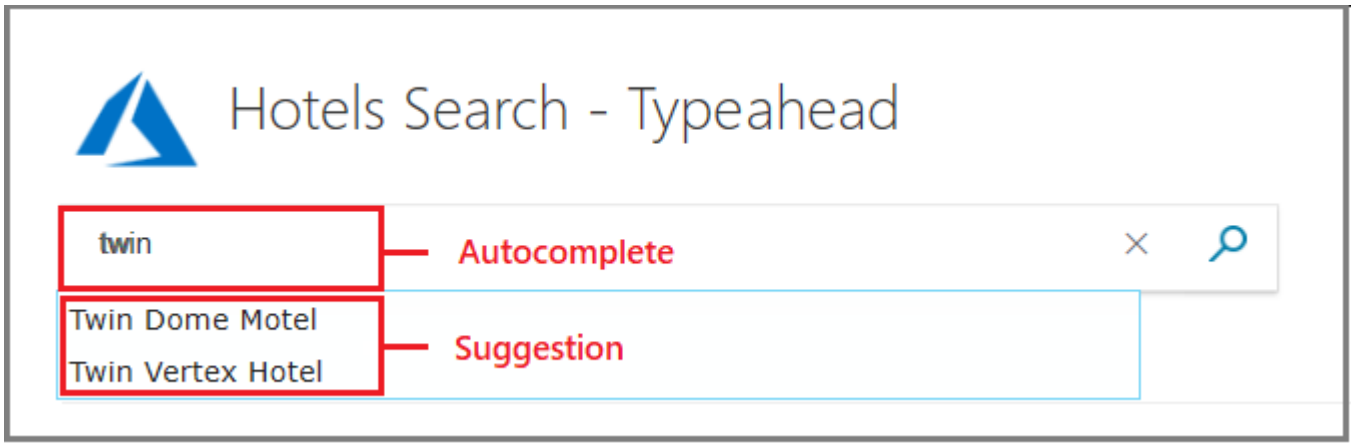
In Azure Cognitive Search, "search-as-you-type" is enabled through a *suggester*. A suggester is an internal data structure that consists of a fields collection. The fields undergo additional tokenization, generating prefix sequences to support matches on partial terms.

For example, if a suggester includes a City field, resulting prefix combinations of "sea", "seat", "seatt", and "seattl" would be created for the term "Seattle". Prefixes are stored in inverted indexes, one for each field specified in a suggester fields collection.

Typeahead experiences in Cognitive Search

A suggester supports two experiences: *autocomplete*, which completes a partial input for a whole term query, and *suggestions* that invite click through to a particular match. Autocomplete produces a query. Suggestions produce a matching document.

The following screenshot from Create your first app in C# illustrates both. Autocomplete anticipates a potential term, finishing "tw" with "in". Suggestions are mini search results, where a field like hotel name represents a matching hotel search document from the index. For suggestions, you can surface any field that provides descriptive information.



You can use these features separately or together. To implement these behaviors in Azure Cognitive Search, there is an index and query component.

- Add a suggester to a search index definition. The remainder of this article is focused on creating a suggester.
- Call a suggester-enabled query, in the form of a Suggestion request or Autocomplete request, using one of the APIs listed below.

Search-as-you-type support is enabled on a per-field basis for string fields. You can implement both typeahead behaviors within the same search solution if you want an experience similar to the one indicated in the screenshot. Both requests target the *documents* collection of specific index and responses are returned after a user has provided at least a three character input string.

How to create a suggester

To create a suggester, add one to an index definition. A suggester gets a name and a collection of fields over which the typeahead experience is enabled. and set each property. The best time to create a suggester is when you are also defining the field that will use it.

- Use string fields only.
- If the string field is part of a complex type (for example, a City field within Address), include the parent in the field: `"Address/City"` (REST and C# and Python), or `["Address"]["City"]` (JavaScript).
- Use the default standard Lucene analyzer (`"analyzer": null`) or a language analyzer (for example, `"analyzer": "en.Microsoft"`) on the field.

If you try to create a suggester using pre-existing fields, the API will disallow it. Prefixes are generated during indexing, when partial terms in two or more character combinations are tokenized alongside whole terms. Given that existing fields are already tokenized, you will have to rebuild the index if you want to add them to a suggester. For more information, see How to rebuild an Azure Cognitive Search index.

Choose fields

Although a suggester has several properties, it is primarily a collection of string fields for which you are enabling a search-as-you-type experience. There is one suggester for each index, so the suggester list must include all fields that contribute content for both suggestions and autocomplete.

Autocomplete benefits from a larger pool of fields to draw from because the additional content has more term completion potential.

Suggestions, on the other hand, produce better results when your field choice is selective. Remember that the suggestion is a proxy for a search document so you will want fields that best represent a single result. Names, titles, or other unique fields

that distinguish among multiple matches work best. If fields consist of repetitive values, the suggestions consist of identical results and a user won't know which one to click.

To satisfy both search-as-you-type experiences, add all of the fields that you need for autocomplete, but then use **$select**, **$top**, **$filter**, and **searchFields** to control results for suggestions.

Choose analyzers

Your choice of an analyzer determines how fields are tokenized and subsequently prefixed. For example, for a hyphenated string like "context-sensitive", using a language analyzer will result in these token combinations: "context", "sensitive", "context-sensitive". Had you used the standard Lucene analyzer, the hyphenated string would not exist.

When evaluating analyzers, consider using the Analyze Text API for insight into how terms are processed. Once you build an index, you can try various analyzers on a string to view token output.

Fields that use custom analyzers or predefined analyzers (with the exception of standard Lucene) are explicitly disallowed to prevent poor outcomes.

> **Note**
>
> If you need to work around the analyzer constraint, for example if you need a keyword or ngram analyzer for certain query scenarios, you should use two separate fields for the same content. This will allow one of the fields to have a suggester, while the other can be set up with a custom analyzer configuration.

Create using the portal

When using **Add Index** or the **Import data** wizard to create an index, you have the option of enabling a suggester:

1. In the index definition, enter a name for the suggester.

2. In each field definition for new fields, select a checkbox in the Suggester column. A checkbox is available on string fields only.

As previously noted, analyzer choice impacts tokenization and prefixing. Consider the entire field definition when enabling suggesters.

Create using REST

In the REST API, add suggesters through Create Index or Update Index.

JSON Copy

```
{
  "name": "hotels-sample-index",
  "fields": [
    . . .
        {
            "name": "HotelName",
            "type": "Edm.String",
            "facetable": false,
            "filterable": false,
            "key": false,
            "retrievable": true,
            "searchable": true,
            "sortable": false,
            "analyzer": "en.microsoft",
            "indexAnalyzer": null,
            "searchAnalyzer": null,
            "synonymMaps": [],
            "fields": []
        },
  ],
  "suggesters": [
    {
      "name": "sg",
```

```
      "searchMode": "analyzingInfixMatching",
      "sourceFields": ["HotelName"]
    }
  ],
  "scoringProfiles": [
    . . .
  ]
}
```

Create using .NET

In C#, define a SearchSuggester object. `Suggesters` is a collection on a SearchIndex object, but it can only take one item. Add a suggester to the index definition.

C# Copy

```
private static void CreateIndex(string indexName, SearchIndexClient indexClient)
{
    FieldBuilder fieldBuilder = new FieldBuilder();
    var searchFields = fieldBuilder.Build(typeof(Hotel));

    var definition = new SearchIndex(indexName, searchFields);

    var suggester = new SearchSuggester("sg", new[] { "HotelName", "Category", "Address/City",
"Address/StateProvince" });
    definition.Suggesters.Add(suggester);

    indexClient.CreateOrUpdateIndex(definition);
}
```

Property reference

Property	Description
`name`	Specified in the suggester definition, but also called on an Autocomplete or Suggestions request.
`sourceFields`	Specified in the suggester definition. It's a list of one or more fields in the index that are the source of the content for suggestions. Fields must be of type `Edm.String` and `Collection(Edm.String)`. If an analyzer is specified on the field, it must be a named lexical analyzer from this list (not a custom analyzer). As a best practice, specify only those fields that lend themselves to an expected and appropriate response, whether it's a completed string in a search bar or a dropdown list. A hotel name is a good candidate because it has precision. Verbose fields like descriptions and comments are too dense. Similarly, repetitive fields, such as categories and tags, are less effective. In the examples, we include "category" anyway to demonstrate that you can include multiple fields.
`searchMode`	REST-only parameter, but also visible in the portal. This parameter is not available in the .NET SDK. It indicates the strategy used to search for candidate phrases. The only mode currently supported is `analyzingInfixMatching`, which currently matches on the beginning of a term.

Use a suggester

A suggester is used in a query. After a suggester is created, call one of the following APIs for a search-as-you-type experience:

- Suggestions REST API
- Autocomplete REST API
- SuggestAsync method
- AutocompleteAsync method

In a search application, client code should leverage a library like jQuery UI Autocomplete to collect the partial query and provide the match. For more information about this task, see Add autocomplete or suggested results to client code.

API usage is illustrated in the following call to the Autocomplete REST API. There are two takeaways from this example. First, as with all queries, the operation is against the documents collection of an index and the query includes a **search** parameter, which in this case provides the partial query. Second, you must add **suggesterName** to the request. If a suggester is not defined in the index, a call to autocomplete or suggestions will fail.

HTTP Copy

```
POST /indexes/myxboxgames/docs/autocomplete?search&api-version=2020-06-30
{
  "search": "minecraf",
  "suggesterName": "sg"
}
```

Sample code

- Create your first app in C# (lesson 3 - Add search-as-you-type) sample demonstrates suggested queries, autocomplete, and faceted navigation. This code sample runs on a sandbox Azure Cognitive Search service and uses a pre-loaded Hotels index with a suggester already created, so all you have to do is press F5 to run the application. No subscription or sign-in is necessary.

Next steps

We recommend the following article to learn more about how requests formulation.

Add autocomplete and suggestions to client code