


Azure Search provides a suggestions feature that allows users to view a list of potential search terms in response to partial string inputs.

If you wish to use search suggestions on your website, you need to add a *suggester* construction in your search indexes. The suggester specifies a list of fields that are used as sources for the content of suggestions.

 For detailed information, refer to the following articles:

- [Suggesters](#)
- [Suggestions \(Azure Search API\)](#)

To add a suggester for an Azure Search index managed by Kentico, you need to [customize](#) the functionality that Kentico uses to build the indexes:

1. Open your Kentico solution in Visual Studio.
2. Create a [custom module class](#).
3. Override the module's **OnInit** method and assign a handler to the **SearchServiceManager.CreatingOrUpdatingIndex.Execute** event.
4. Perform the following in the event's handler method:
  - a. Access the **Microsoft.Azure.Search.Models.Index** object representing the processed index via the **Index** property of the handler's *CreateOrUpdateIndexEventArgs* parameter.
  - b. Write conditions to add different suggesters for specific indexes.
  - c. Prepare a **Microsoft.Azure.Search.Models.Suggester** object according to your requirements and add it to the **Suggesters** list of the processed index.



#### Important

The system triggers the **CreatingOrUpdatingIndex** event both when building new indexes and when updating indexes that already exist under the specified Azure Search service. Depending on the number of indexed pages or objects and the used batch size, the event may occur multiple times when building a single search index (separately for each batch of processed search documents that include a new field not yet contained by the index).

Your code needs to handle the following scenarios:

- The index already contains the suggester you are adding.
- The index does not yet contain all possible fields (for example in cases where the first processed batch of search documents does not include an object with the required fields).

See the code of the example below.

5. Sign in to the Kentico administration interface.
6. Open the **Smart search** application and **Rebuild** any related Azure Search indexes.

The system creates the customized Azure Search indexes with the specified suggester. You can see the suggester when viewing the details of index fields in the [Microsoft Azure portal](#).

You can now adjust your [search components](#) to retrieve and display the suggestions – you need to call the **ISearchIndexClient.Documents.Suggest** method when users input search text and process the retrieved **DocumentsSuggestResult** data.

## Example

The following example demonstrates how to create a suggester for an Azure Search index named *dg-store*. The sample suggester uses the *skuname* field (containing product names) as the source for suggestions.

Start by preparing a separate project in your Kentico solution for the custom module class:



1. Open your Kentico solution in Visual Studio.
2. Create a new *Class Library* project in the Kentico solution named **SearchCustomization**.
3. Add references to the required Kentico libraries (DLLs) for the new project:
  - a. Right-click the project and select **Add -> Reference**.
  - b. Switch to the **Browse** tab, click **Browse**, and navigate to the **Lib** folder of your Kentico web project.
  - c. Add references to the following libraries:
    - **CMS.Base.dll**
    - **CMS.Core.dll**
    - **CMS.DataEngine.dll**
    - **CMS.Search.Azure.dll**
4. Right-click the *SearchCustomization* project in the **Solution Explorer** and select **Manage NuGet Packages**.
5. Install the **Microsoft.Azure.Search** package.
6. Reference the *SearchCustomization* project from the Kentico web project (*CMSApp* or *CMS*).
7. Edit the *SearchCustomization* project's **AssemblyInfo.cs** file (in the *Properties* folder).
8. Add the **AssemblyDiscoverable** assembly attribute:

```
using CMS;

[assembly:AssemblyDiscoverable]
```

Continue by implementing the custom module class and rebuilding your search index:

1. Create a new class named **CustomAzureSearchModule** under the *SearchCustomization* project, with the following code:

```
using System;
using System.Collections.Generic;
using System.Linq;

using CMS;
using CMS.DataEngine;
using CMS.Search.Azure;

using Microsoft.Azure.Search.Models;

// Registers the custom module into the system
[assembly: RegisterModule(typeof(CustomAzureSearchModule))]

public class CustomAzureSearchModule : Module
{
    // Module class constructor, the system registers the module under the name
    "CustomAzureSearch"
    public CustomAzureSearchModule()
        : base("CustomAzureSearch")
    {
    }

    // Contains initialization code that is executed when the application starts
    protected override void OnInit()
    {
        base.OnInit();

        // Assigns a handler to the CreatingOrUpdatingIndex event for Azure
        Search indexes
        SearchServiceManager.CreatingOrUpdatingIndex.Execute += AddSuggester;
    }
}
```



```
private void AddSuggester(object sender, CreateOrUpdateIndexEventArgs e)
{
    Microsoft.Azure.Search.Models.Index index = e.Index;

    // Ends the handler method if the index name is not 'dg-store'
    if (!index.Name.Equals("dg-store", StringComparison.
InvariantCultureIgnoreCase))
    {
        return;
    }

    // Initializes the index's list of suggesters (if it does not exist)
    if (index.Suggesters == null)
    {
        index.Suggesters = new List<Suggester>();
    }

    // Used to determine whether a new suggester was created and needs to be
added to the index
    bool newSuggester = false;

    // Checks whether the index already contains a suggester named
'productnamesuggester'
    Suggester suggester = index.Suggesters.FirstOrDefault(s => s.Name ==
"productnamesuggester");

    // Creates a new suggester if it does not exist
    if (suggester == null)
    {
        suggester = new Suggester
        {
            Name = "productnamesuggester",
            SourceFields = new List<string>()
        };
        // A new suggester was created, it needs to be added to the index
        newSuggester = true;
    }

    // Creates a dictionary containing the index's fields
    Dictionary<string, Field> indexFields = index.Fields.ToDictionary(f => f.
Name, StringComparer.InvariantCultureIgnoreCase);

    // Confirms that the index contains the 'skuname' field and that it is
not yet added as a suggester source field
    if (indexFields.ContainsKey("skuname") && !suggester.SourceFields.Contains
("skuname", StringComparer.InvariantCultureIgnoreCase))
    {
        // Adds the 'skuname' fields as a source for suggestions
        suggester.SourceFields.Add("skuname");
    }

    // If a new suggester was created and is not empty, adds it to the index
    if (newSuggester && suggester.SourceFields.Count > 0)
    {
        index.Suggesters.Add(suggester);
    }
}
```

2. Save all changes and **Build** the *SearchCustomization* project.
3. Sign in to the Kentico administration interface.
4. Open the **Smart search** application and **Rebuild** the *dg-store* index.

The *dg-store* Azure Search index now contains the *productnamesuggester* suggester, using *skuname* as the source field for the content of suggestions. To retrieve and display the suggestions, you need to implement the required logic in your search components (see [Integrating Azure Search into pages](#) for general information). Call the **ISearchIndexClient.Documents.Suggest** method and process the retrieved **DocumentSuggestResult** data.

```
using System;

using Microsoft.Azure.Search;
using Microsoft.Azure.Search.Models;

...

if (!String.IsNullOrEmpty(searchString))
{
    // Retrieves suggestions based on search input using 'productnamesuggester'
    DocumentSuggestResult suggestResult = searchIndexClient.Documents.Suggest
(searchString, "productnamesuggester");
}
```