When creating custom smart search indexes, you do not define the content on the **Indexed content** tab in the **Smart search** application. Instead, you must implement all functionality of the index by writing code. In the administration interface, you only need to specify the names of the assembly and class that contain the custom index logic.

To define a custom index, create a class that implements the **ICustomSearchIndex** interface (**CMS.Search** namespace).

To integrate the class into your application:

- Create a new assembly (Class library) in your web project and include the index class there. When using this approach, you must add the appropriate references to both the assembly and the main Kentico project.

  OR

- Define the custom index directly in the Kentico web project and load the class via the API. This ensures that the system automatically compiles the index class and you do not need to use a separate assembly. The example below demonstrates this approach.

Writing the custom index code

The following example shows how to create a custom index that searches the content of text files:

1. Open your web project in Visual Studio.
2. Add a new class into the **App_Code** folder (or **Old_App_Code** on web application installations). For example, name the class **TextFileIndex.cs**.
3. Edit the class and make sure that the following **using** statements are present at the top of the code:

```
using System;

using CMS.Search;
using CMS.DataEngine;
using CMS.IO;
using CMS.Helpers;
using CMS.EventLog;
using CMS.Base;
using CMS;
```

4. Make the class implement the **ICustomSearchIndex** interface.

```
public class TextFileIndex : ICustomSearchIndex
```

5. Define the **Rebuild** method inside the class:

> You must always include the *Rebuild* method when writing custom indexed. The method fills the index with data, which determines what kind of searches the index provides. The system calls the method when building the index for the first time and on each subsequent rebuild.

```
/// <summary>
/// Fills the index with content.
/// </summary>
/// <param name="srchInfo">Info object representing the search index</param>
public void Rebuild(SearchIndexInfo srchInfo)
{
        // Checks whether the index info object is defined
        if (srchInfo != null)
        {
                // Gets an index writer object for the current index
```

```
                IIndexWriter iw = srchInfo.Provider.GetWriter(true);

                // Checks whether the writer is defined
                if (iw != null)
                {
                        try
                        {
                                // Gets an info object of the index settings
                                SearchIndexSettingsInfo sisi = srchInfo.
IndexSettings.Items[SearchHelper.CUSTOM_INDEX_DATA];

                                // Gets the search path from the Index data field
                                string path = Convert.ToString(sisi.GetValue
("CustomData"));

                                // Checks whether the path is defined
                                if (!String.IsNullOrEmpty(path))
                                {
                                        // Gets all text files from the specified
directory
                                        string[] files = Directory.GetFiles(path,
"*.txt");

                                        // Loops through all files
                                        foreach (string file in files)
                                        {
                                                // Gets the current file info
                                                FileInfo fi = FileInfo.New(file);

                                                // Gets the text content of the
current file
                                                string text = fi.OpenText().
ReadToEnd();

                                                // Checks that the file is not
empty
                                                if (!String.IsNullOrEmpty(text))
                                                {
                                                        // Converts the text to
lower case
                                                        text = text.
ToLowerCSafe();

                                                        // Removes diacritics
                                                        text = TextHelper.
RemoveDiacritics(text);

                                                        // Creates a new Lucene.
Net search document for the current text file
                                                        SearchDocumentParameters
documentParameters = new SearchDocumentParameters()
                                                        {
                                                                Index = srchInfo,
                                                                Type =
SearchHelper.CUSTOM_SEARCH_INDEX,
                                                                Id = Guid.
NewGuid().ToString(),
                                                                Created = fi.
CreationTime
                                                        };
```

```
                                                        ILuceneSearchDocument doc
= LuceneSearchDocumentHelper.ToLuceneSearchDocument(SearchHelper.CreateDocument
(documentParameters));

                                                        // Adds a content field.
This field is processed when the search looks for matching results.
                                                        doc.AddGeneralField
(SearchFieldsConstants.CONTENT, text, SearchHelper.StoreContentField, true);

                                                        // Adds a title field.
The value of this field is used for the search result title.
                                                        doc.AddGeneralField
(SearchFieldsConstants.CUSTOM_TITLE, fi.Name, true, false);

                                                        // Adds a content field.
The value of this field is used for the search result excerpt.
                                                        doc.AddGeneralField
(SearchFieldsConstants.CUSTOM_CONTENT, TextHelper.LimitLength(text, 200), true,
false);

                                                        // Adds a date field. The
value of this field is used for the date in the search results.
                                                        doc.AddGeneralField
(SearchFieldsConstants.CUSTOM_DATE, fi.CreationTime, true, false);

                                                        // Adds a url field. The
value of this field is used for link urls in the search results.
                                                        doc.AddGeneralField
(SearchFieldsConstants.CUSTOM_URL, file, true, false);

                                                        // Adds an image field.
The value of this field is used for the images in the search results.
                                                        // Commented out, since
the image file does not exist by default
                                                        // doc.AddGeneralField
(SearchFieldsConstants.CUSTOM_IMAGEURL, "textfile.jpg", true, false);

                                                        // Adds the document to
the index
                                                        iw.AddDocument(doc);
                                                }
                                        }

                                        // Flushes the index buffer
                                        iw.Flush();

                                        // Optimizes the index
                                        iw.Optimize();
                                }
                        }

                        // Logs any potential exceptions
                        catch (Exception ex)
                        {
                                EventLogProvider.LogException
("CustomTextFileIndex", "Rebuild", ex);
                        }

                        // Always close the index writer
                        finally
```

```
                        {
                                iw.Close();
                        }
                }
        }
}
```

You need to write the code of the *Rebuild* method according to the specific purpose of the index. Use the following general steps for all indexes:

a. Get an **Index writer** instance for the search index.
   - The index writer object must implement the **CMS.Search.IIndexWriter** interface.
   - Get the index writer by calling the *Provider.GetWriter(true)* method of the *SearchIndexInfo* object.

b. Define search **Documents** and their fields for the items that you wish to add to the index.
   - Custom search document objects must implement the **CMS.Search.ILuceneSearchDocument** interface.
   - Create search documents by calling the *CMS.Search.SearchHelper.CreateDocument* method, and then convert the result via the *LuceneSearchDocumentHelper.ToLuceneSearchDocument* method.

c. Call the **AddDocument** method of the *Index writer* for every search document.
d. After you have added all required search documents, call the **Flush** and **Optimize** methods of the *Index writer*.
e. Call the **Close** method of the *Index writer*.

Using data parameters for custom indexes

The **SearchIndexInfo** parameter of the *Rebuild* method allows you to access the data fields of the corresponding search index object. The sample code loads the content of the **Index data** field and uses it to define the path to the searched text files.

When writing your own custom indexes, you can use the *Index data* field as a string parameter for any required purpose. The parameter allows you to modify the behavior of the index directly from the administration interface without having to edit the index code.

Updating the content of custom indexes

By default, the only way to update the content of a custom index is to rebuild the whole index (i.e. using the implementation of the *Rebuild* method). You cannot use the Kentico search indexing tasks to update custom indexes.

With additional custom development, you can update indexes by calling the **Update** or **Delete** methods of the **CMS.Search.SearchHelper** class. Typically, you need to update the index from custom code outside of the index class whenever the indexed content changes. However, in this case you need to ensure that the API is never called concurrently – problems can occur if multiple processes attempt to update the same index at the same time.

Loading App_Code index classes

For indexes defined in the *App_Code* folder, you need to ensure that the system loads the appropriate class when building the index. You can find additional information related to this topic in Loading custom classes from App_Code.

Add the **RegisterCustomClass** assembly attribute above the *TextFileIndex* class declaration (requires *using* statement for the **CMS** namespace).

```
[assembly: RegisterCustomClass("TextFileIndex", typeof(TextFileIndex))]
```

This attribute registers custom classes and makes them available in the system. The attribute accepts two parameters:

- The first parameter is a string identifier representing the name of the class. The name must match the value of the **Index provider class** field specified for the given search index on its **Indexed content** tab (*TextFileIndex* in this example).
- The second parameter specifies the type of the class as a **System.Type** object. When the system builds the custom search index, the attribute ensures that an instance of the given class is provided.

Registering custom search indexes

1. Log in to the Kentico administration interface and open the **Smart search** application.
2. Click **New Index**. Fill in the following properties:
    - **Display name**: Text file index
    - **Index type**: Custom Index
3. Click **Save**.
4. Switch to the **Indexed content** tab and enter the names of the assembly and class where the custom index is implemented:
    - **Assembly name**: (custom classes)
    - **Class**: TextFileIndex
5. Type any required parameters into the **Index data** field. In this example, you need to specify the file system path of the folder containing the text files that the index will search. You can create a new folder for this purpose, e.g. *C:\SearchExample* and add some text files into it.
6. Click **Save**.
7. Go to the **General** tab and **Rebuild** the index.

Result

The index is now fully functional. To test the index, switch to the **Search preview** tab and try searching for any words from the text files created in the *SearchExample* folder.