Article
Browse Code
Stats
Revisions (3)
Alternatives
Comments
Add your own alternative version

Tagged as
ASP.NET-Core
visual-studio-2017
Stats
2.3K views
3 bookmarked
Posted 10 Jul 2018
Licenced CPOL

Working with the dtSearch® ASP.NET Core WebDemo Sample Application

F. Scott Barker, 10 Jul 2018

Some features of the ASP.NET Core WebDemo application dtSearch provides in the dtSearch Engine product include a scrolling word list, faceted search navigation, and multicolor hit highlighting.

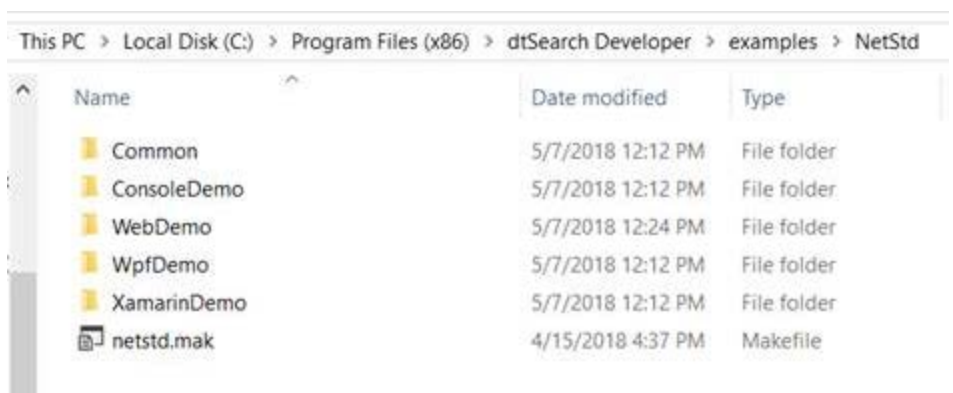
Editorial Note

This article is in the Product Showcase section for our sponsors at CodeProject. These articles are intended to provide you with information on products and services that we consider useful and of value to developers.

Introduction

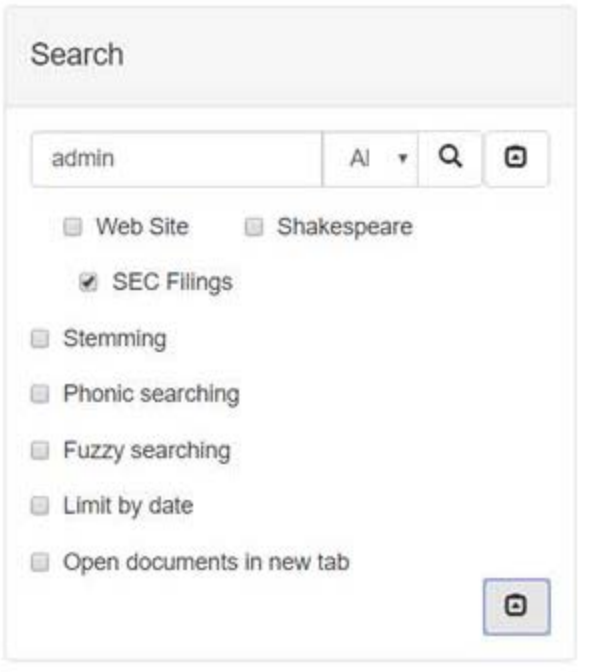
In this article, I'm going to show you some of the features of the ASP.NET Core WebDemo application dtSearch provides in the dtSearch Engine product. Features include a scrolling word list, faceted search navigation, and multicolor hit highlighting. The features outlined in this article are available in the dtSearch Engine version 7.91 build 8546 or later. Working with the demo also requires Visual Studio 2017 updated to 15.5 or later.

One of the things developers have appreciated over the many releases of the dtSearch developer products are the code samples dtSearch provides to show how to use various features. Unlike some product companies that give short snippets of code to show how to use a feature, dtSearch shows how to use many features in a real world way, often allowing developers to use the code with only minor changes in their own applications. The ASP.NET Core WebDemo sample application is a great example of this.



Example folder for NetStd

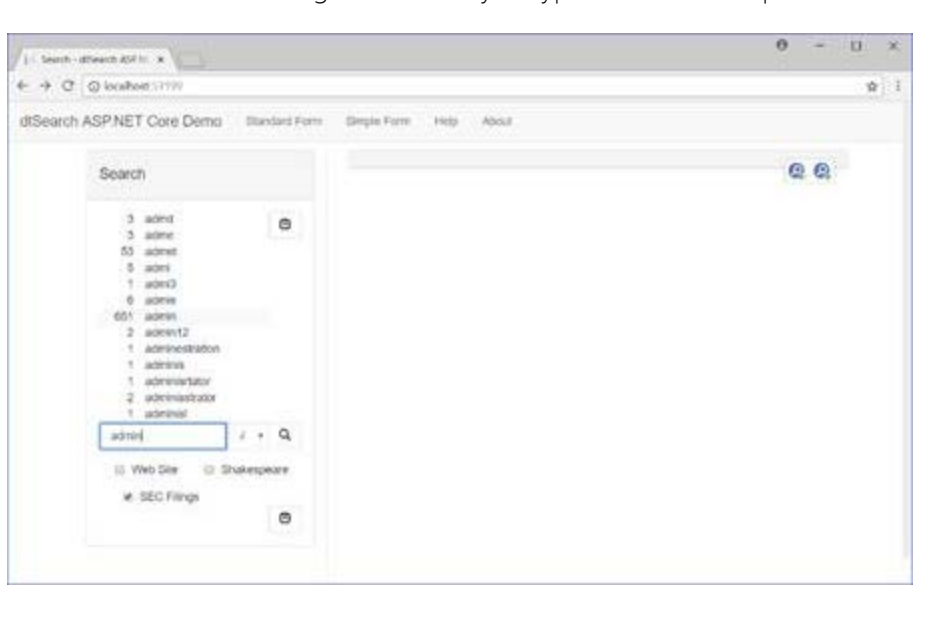
The .NET Core implementation is also a good starting point for developing cross-platform applications. You can read more on this from Microsoft here. dtSearch highlights this by including sample code for not only the ASP.NET Core WebDemo, but also other platforms using the .NET Standard library as shown here in the example folder for NetStd.



ASP.NET Core demo showing basic initial search form

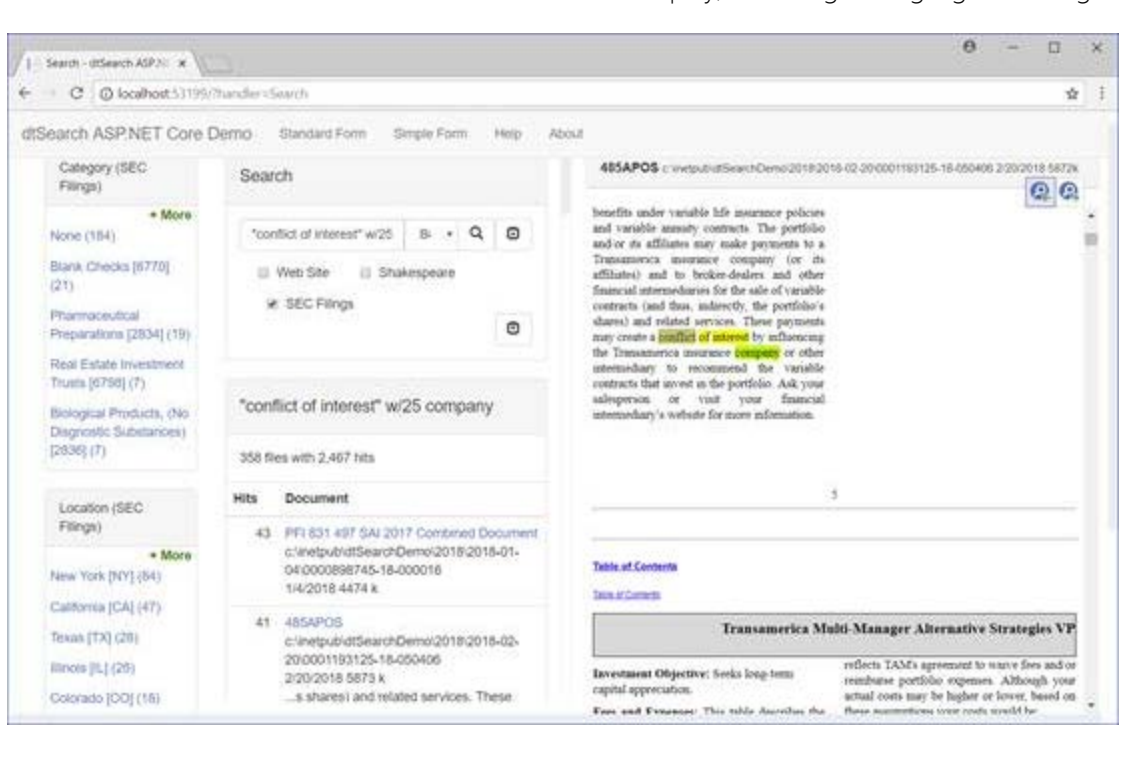
You can try the web demo on the dtSearch website here.

The demo offers a scrolling word list as you type in a search request.



ASP.NET Core demo showing scrolling word list

This screenshot shows faceted search and document display, including hit-highlighted navigation options.



ASP.NET Core demo showing search results

Setting Up the Sample Data

For sample data, the demo relies on a collection of SEC filings called EDGAR. The EDGAR data comes from the SEC website here.

The actual files to download are **.gz* files, which are here.

The **.gz* files contain a series of **.nc* files, each representing one filing. The **.nc* files in turn contain metadata and documents in a unique SEC format. dtSearch has a utility, *EdgarExtract.exe*, that will unpack a single **.gz* file to a folder tree, like this:

```
EdgarExtract 20180102.nc.tar.gz k:\edgar\2018
EdgarExtract 20180103.nc.tar.gz k:\edgar\2018
***
```

For each extracted file, EdgarExtract will create a ***props.xml*** file with the metadata associated with the file, including the facets.

NOTE: EdgarExtract is just a quick utility to generate sample data from the SEC filings, so it does not necessarily get everything that may be stored in a **.nc* file.

Once you have the data downloaded, it is time to index the data.

Indexing the Sample Data

To index the sample data along the lines of the `WebDemo`:

- Designate as stored fields: `Company`, `CompanyState`, `FilingType`, `CompanySic`
- Designate as enumerable fields (for use in faceted search): `CompanyState FilingType CompanySic`
- Tell dtSearch to look for the **.props.xml* files that EdgarExtract creates. To do this, the undocumented internal flag `dtsFlagUsePropsXml` (16) must be set in `Options.OtherFlags`
- Enable caching of documents and text so the index will be all you need to show retrieved documents and highlight hits

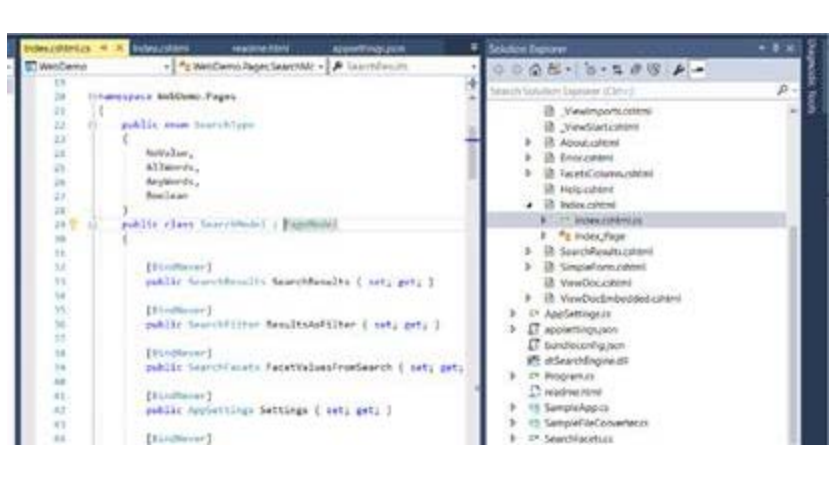
To do this with dtSearch Desktop:

1. Start dtSearch Desktop
2. Press F12 to enable "Developer Mode"
3. In Options > Preferences > Developer Options, set OtherFlags to 16. This tells dtSearch to look for the **.props.xml* files that EdgarExtract creates to annotate each document with the SEC's metadata
4. Run the dtSearch Desktop indexer and click Create (Advanced)
5. Under "Fields to display in search results" list: `Company CompanyState FilingType CompanySic`
6. Click the "Facets" button to access faceted search options (enabling Developer Mode makes this visible). Under Faceted Search Fields, enter: `CompanyState FilingType CompanySic`
7. Check the boxes to enable caching of documents and caching of text
8. Create the index
9. Add documents to the index as usual using these recommended filters:
   a. Include: **.pdf *.htm* *.rtf *.doc* *.xls* *.ppt* *.xml *.txt*
   b. Exclude: **.props.xml *abs-ee*.xml*

Key Demo Features

Because of the depth of the `WebDemo`, instead of going line by line, I will point out various objects and the features they encapsulate. The first of these objects is the *Index.cshtml* object and its accompanying C# file. This file is well worth reading on your own.

Index.cshtml and Index.cshtml.cs

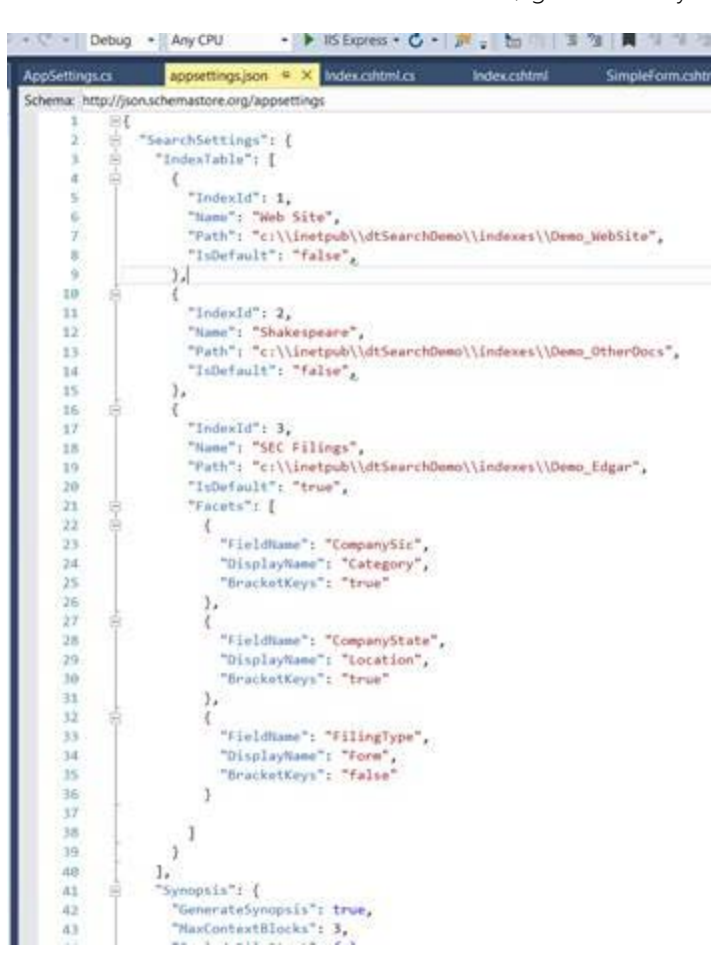


The SearchModel class handles setting up and executing a SearchJob.

The *Index.chtml* file is made up mainly of the `SearchModel` class.

It allows both `POST` and `GET` requests. `POST` requests will come from the search form. `GET` requests will come from links created in search results pages to initiate additional searches. Two types of links are created:

1. searches to go to other pages of search results, generated by the `SearchPager` class, and
2. searches to handle faceted searches, generated by the `SearchFacets` class



Displaying various options in AppSettings.json/AppSettings.cs.

Indexes are identified by `IndexId` rather than by path to avoid exposing index paths over the web. A table in *AppSettings.json*/*AppSettings.cs* provides the mapping between index ids and index paths.

Scrolling Word List

The scrolling word list is implemented on the server side in *WordsController.cs*, which uses the `dtSearch WordListBuilder` control to enumerate words. The client-side JavaScript is in *wwwroot\js\for_WordList.js* and uses AJAX to get words to display from the server.

Use the `IndexCache` object and the `WordListBuilder` together for optimal performance. (Otherwise, the `WordListBuilder` would need to open the index for every incoming request.) The `IndexCache` is set up as a singleton in *Startup.cs* and passed to *WordsController.cs* and *SearchModel.cs* using dependency injection.

Document Display

The demo includes options to display retrieved documents in an `IFRAME` (*ViewDoc.cshtml*) or in a separate window (*ViewDocEmbedded.cshtml*).

The demo does not maintain session state so the links to open each document must contain all of the information needed to highlight hits in that document. The `GetSearchResultsItem` method in `SearchModel` handles this, using the `SearchResults.UrlEncode*` methods to generate a `UrlEncoded` expression to use in the link `HREF` attribute. A set of corresponding `UrlDecode*` methods are used to re-generate the `SearchResults` item when a link is clicked.

There are "`WithIndexId`" versions of these methods that allow a supplied `IndexId` to be used instead of the index path. This is done to avoid exposing index paths to web users.

Multicolor hit highlighting (using a different color for each search term) can be enabled through an option in *appSettings.cs*. Multicolor highlighting requires more information to be generated at search time because the dtSearch Engine needs to know which hit matches each search term. Three flags at search time request this information: `dtsSearchWantHitsByWord`, `dtsSearchWantHitsArray`, and `dtsSearchWantHitsByWordOrdinals`.

For more information on multicolor hit highlighting, check out this article.

Because this information is generated by a search, the link for each search results item needs to contain enough information to run a search that just retrieves that item. The `GetSearchResultsItem` method in `SearchModel` handles this, using the `SearchResults`. `UrlEncodeItemAsSearchWithIndexId` method in the API.

SearchPager.cs

The `SearchPager` class handles paging of search results. It does this by URL-encoding the search (using `SearchUrlEncoder`) that generates each page of results. A Bootstrap pagination control in *SearchResults.cshtml* is used to display the links to generate each page of results.

SearchFacets.cs and FacetsColumn.cshtml

`SearchFacets` handles implementing the faceted search interface, and *FacetsColumn.cshtml* displays the faceted search information. For further information about implementing a faceted search interface with the dtSearch Engine, please see:

- dtSearch faceted search FAQ
- CodeProject dtSearch Engine faceted search introductory article
- CodeProject dtSearch Engine faceted search advanced article

For each index, the facets applicable to the index are listed in the index table in *AppSettings.json*.

The facets in the demo come from the SEC Edgar database. The SEC Edgar database is a public collection of securities filings. The three facets in the demo are:

`CompanySic`: Numeric "SIC" code used in Edgar to identify the type of business. EdgarExtract adds descriptive text so the field has the descriptive text with the numeric code in brackets. For example, SIC Code 1311 is represented as "Crude Petroleum & Natural Gas [1311]".

`CompanyState`: A two letter code that the SEC uses to identify the state or country where the company is located. For states, this is just the usual abbreviation (MD=Maryland), but for countries the codes are pretty random. Again, EdgarExtract adds descriptive text with the two-letter code following in brackets.

`FilingType`: The SEC form that the filing corresponds to.

The facet tables in *AppSettings.json*, `CompanySic` and `CompanyState` are marked with the `BracketKeys` property, which tells the demo that the key value in a facet is in brackets at the end of the facet. For example, if someone clicks on the facet "Crude Petroleum & Natural Gas [1311]", this will generate a search that includes the condition "SicCode contains 1311".

Summary

dtSearch by far supplies the most complete code I have ever worked with for their various demonstrations. The dtSearch ASP.NET Core WebDemo is no exception. Whether you are looking for examples of using the faceted searches, the `WordListBuilder`, or many other features using the ASP.NET Core code, this demo is a good starting point.

More on dtSearch

- dtSearch.com
- A Search Engine in Your Pocket – Introducing dtSearch on Android
- Blazing Fast Source Code Search in the Cloud
- Using Azure Files, RemoteApp and dtSearch for Secure Instant Search Across Terabytes of A
- Windows Azure SQL Database Development with the dtSearch Engine
- Faceted Search with dtSearch – Not Your Average Search Filter
- Turbo Charge your Search Experience with dtSearch and Telerik UI for ASP.NET
- Put a Search Engine in Your Windows 10 Universal (UWP) Applications
- Indexing SharePoint Site Collections Using the dtSearch Engine DataSource API

License

Share

About the Author

F. Scott Barker
Software Developer (Senior) AppsPlus
United States

Scott Barker, a former member of the Microsoft team, has been developing with dtSearch products for use in both web and desktop based applications for many years now. This includes a wide range of applications from managing documents on unknown network setups for government agencies, including indexing IRS forms using dtSearch Spider product against the IRS Website, to using the dtSearch engine for indexing email messages for use with a Deep Learning Engine. You can see many of these projects documented in cases studies on the dtSearch Website on the Custom dtSearch Development — Independent Provider Listings page.

Comments and Discussions