Umbraco research

1. Usefull packeges

* Umbraco Bookshelf

Allows users to create, display and share Umbraco learning resources - <https://our.umbraco.org/projects/backoffice-extensions/bookshelf> or install via nuget [www.nuget.org/packages/UmbracoBookshelf/](http://www.nuget.org/packages/UmbracoBookshelf/)

Note : After installing the package go to Users section, find your user and check checkbox for Bookshelf – know you will see package icon in the main tree.

Umbraco presentation

1. What is Examine?

1.1 This is a provider based Indexer/Searcher API and wraps the Lucene.Net indexing/searching engine. Working with Examine is very easy and allows you to query or index almost any content in web site.

Umbraco Examine is the Umbraco implementation of Examine. It is not exclusive to Umbraco and can be used as a completely stand alone component on any project that needs a fast Index.

* 1. License

Examine is under Microsoft Public License (Ms-PL), this is mean, we are free of charge to use it, but can not use it examine name, logo or trademark. More info - <https://examine.codeplex.com/license?LicenseHistoryId=37500>.

1. Implementation
   1. Basic configuration – all configuration comes whit main umbraco installation.

You can start using examine search immediately.

Little Examine Terminology - there is 3 basic parts:

* + - **Indexer** - This is the object that performs the storing data into the index.
    - **Searcher** – The searcher is the object that performs the searching of data that is stored in the index.
    - **Index Set** - An index set is what defines an index, where the index is saved and how the information is stored in the index.

Naming conventions - Our Indexer, Searcher and associated Index Set must all be named according to convention so that they match.

Conventions:

* + - {name}Indexer
    - {name}Searcher
    - {name}IndexSet

Examples:

* + - ExternalIndexer
    - ExternalSearcher
    - ExternalIndexSet

After installing Umbraco site look in the config folder. There is all config files that umbraco needs.

* /Config/ExamineIndex.config

By default is configured 3 IndexSets – InternalIndexSet, InternalMemberIndexSet and ExternalIndexSet.

All index sets that starts with ‘Internal’ prefix are internal umbraco and we don not have work there. The same is valid also for searchers and indexers.

Umbraco provides for us ExternalIndexSet:

<IndexSet SetName="ExternalIndexSet" IndexPath="~/App\_Data/TEMP/ExamineIndexes/External/" />

We can extend this configuration, but this will be seen in next sliders.

For now we can search in all site content(fields and document types).

* /Config/ExamineSettings.config

There is ExamineIndexProviders and ExamineSearchProviders. Also by three of kind – internal and external.

* + ExternalIndexer

<add name="ExternalIndexer" type="UmbracoExamine.UmbracoContentIndexer, UmbracoExamine"/>

We can extend it with this additional properties:

* + - dataService - the type that this provider will instantiate in order to query Umbraco for the data that it requires. Generally this shouldn't need to change unless you want to use test data from a non-umbraco source or you have very custom requirements.
    - indexSet - explicitly specifies the index set to use. Generally this wired up based on naming convensions.
    - supportUnpublished - if you want the indexer to index content that is not published.
    - supportProtected - if you want the indexer to index content that is protected.
    - runAsync = will process the queue files into the index asynchronously, unless you are testing, this should always be true.
    - interval = how often the async service will process the file queue in seconds.
    - analyzer = the Lucene.Net analyzer to use when storing data.

See: <http://www.aaron-powell.com/lucene-analyzer>.

* + - enableDefaultEventHandler = will automatically listen for Umbraco events and index when required.
    - logLevel="Info" or "Verbose". Info is the default, Verbose will show more detailed logs
  + ExternalSearcher

<add name="ExternalSearcher" type="UmbracoExamine.UmbracoExamineSearcher, UmbracoExamine" />

We can extend it with this additional property:

* + - indexSet - explicitly specifies the index set to use. Generally this wired up based on naming convensions.
  + Note: In ExamineSearchProviders section, defaultProvider must be the name of our search provider - e.g. – ExternalSearcher. This is by default.
  1. Custom configuration:

In main cases only ExternalIndexSet required custom settings.

As we said by defaut our index set looks like this:With

<IndexSet SetName="ExternalIndexSet" IndexPath="~/App\_Data/TEMP/ExamineIndexes/External/" />

This will use all site content for searching.

2.2.1 Example customization:

<IndexSet SetName="ExternalIndexSet" IndexPath="~/App\_Data/TEMP/ExamineIndexes/External/">

<IndexAttributeFields>

<!-- Set here all page properties that we want to be indexed. -->

<add Name="id" />

<add Name="version" />

<add Name="parentID" />

<add Name="writerID" />

<add Name="creatorID" />

</IndexAttributeFields>

<IndexUserFields>

<!-- Set here all site custom properties that we want to be indexed. -->

<add Name="testTitle" EnableSorting="true" />

<add Name="testDescription" EnableSorting="true" />

</IndexUserFields>

<IncludeNodeTypes>

<!-- Set here all site document types that we want to be indexed. -->

<add Name="Test"/>

</IncludeNodeTypes>

<ExcludeNodeTypes>

<!-- Set here all site document types that we want to NOT be indexed. -->

</ExcludeNodeTypes>

</IndexSet>

2.3 Code behind customization:

//// Very basic - get all results

//var results = Umbraco.TypedSearch(model.SearchText);

// creating custom searcher

// var defaultSearcher = ExamineManager.Instance.DefaultSearchProvider;

var searcher = ExamineManager.Instance.SearchProviderCollection["ExternalSearcher"]; // the same as defaultSearcher

// Create search criteria to build the search query

var searchCriteria = searcher.CreateSearchCriteria();

// We have some grouping methods heplping us to build the query - Or(), And(), Not(), Equals(), GroupedOr(), GroupedAnd() and e.g.

var query = searchCriteria.Field("testTitle", "hello")

.Or()

.Field("testDescription", "hello")

.Compile();

var results = searcher.Search(query);

// Important note!!!

// Here is the tin moment how Examine reads the query and feeds to Lucene.

// +testTitle:hello testDescription:hello - this means find me all pages that testTitle MUST contain "hello" or all pages that testDescription SHOULD contain "hello"

// The + specifies that it MUST meet this rule.

**Difference between Lucene boolean clauses – MUST and SHOULD**

Assume that there are two clauses: Clause A and Clause B.

* + Clause A have SHOULD, Clause B - SHOULD

This will imply that even if one of the clause is satisfied (A or B), then the document will be a hit.

* + Clause A have MUST, Clause B - SHOULD

In this case, a document will be a hit when it "will" satisfy clause A whether this document satisfies clause B or not.

But if the document does not satisfies clause A, then no matter whether it satisfies clause B or not, it will not be a hit.

* + Clause A have MUST, Clause B - MUST

In this case, a document will be a hit, only when it will satisfy "both" the clauses. If it will fail to satisfy even one of the clause, then it will not be a hit.

There is third clause - MUST\_NOT. Use this operator for clauses that must not appear in the searching documents.

// To fix this we can use BooleanOperation.Or in our search criteria

var searchCriteria = searcher.CreateSearchCriteria(BooleanOperation.Or);

// Now the query looks like

// testTitle:hello testDescription:hello - which means give me anything where testTitle or testDescription contain "hello"

Assume that in the all other examples we will use search criteria object with Boolean operator Or:

searchCriteria = searcher.CreateSearchCriteria(BooleanOperation.Or);

Another examples:

// LuceneQuery: {nodeTypeAlias:home +content:hello}

// Means - give me only this pages that is from "Home" doc type and have word "hello" in the "content" property

var query = searchCriteria

.Field("nodeTypeAlias", "Home")

.And()

.Field("content", "hello")

.Compile();

// LuceneQuery: {nodeTypeAlias:home content:hello}

// Means - give me only this pages that is from "Home" doc type or any other pages that have word "hello" in the "content" property

var queryTwo = searchCriteria

.Field("nodeTypeAlias", "Home")

.Or()

.Field("content", "hello")

.Compile();

// LuceneQuery: nodeTypeAlias:test +(+testTitle:hello +testDescription:hello)

// Means give me this pages from doc type "Home" and they fields "testTitle" and "testDescription" must contains word "hello"

var query = searchCriteria

.Field("nodeTypeAlias", "Test")

.And()

.GroupedAnd(new string[] { "testTitle", "testDescription"}, new string[] { "hello" })

.Compile();

// LuceneQuery: nodeTypeAlias:test (testTitle:hello testDescription:hello)

// Means give me this pages from doc type "Home" or some of their fields "testTitle" and "testDescription", must contains word "hello"

var query = searchCriteria

.Field("nodeTypeAlias", "Test")

.Or()

.GroupedOr(new string[] { "testTitle", "testDescription"}, new string[] { "hello" })

.Compile();

Fuzzy

// Sometimes users will query our site looking for a term that they could have misspelled or is very close. Fuzzy gives you the ability to get Lucene to look for terms that look like your term. Eg mound could actually be sound.

var query = searchCriteria.Fields("testTitle", "hello".Fuzzy(0.8f)).Compile();

//The optional value you pass into Fuzzy between 0 and 1 specifies how Fuzzy or how close the match is to the original. For instance a match of 0.5 will not return when a threshold of 0.8 is specified.

Boosting

// Sometime you want to give a field more relevance than others. There is Boosting came up. It gives a particular query a higher relevance.

var query = searchCriteria.Fields("testTitle", "hello".Boost(8)).Or().Field("testDescription", "hello".Boost(5)).Compile();

Raw Lucene Queries

We may build our custom Lucene string and put in a query like this:

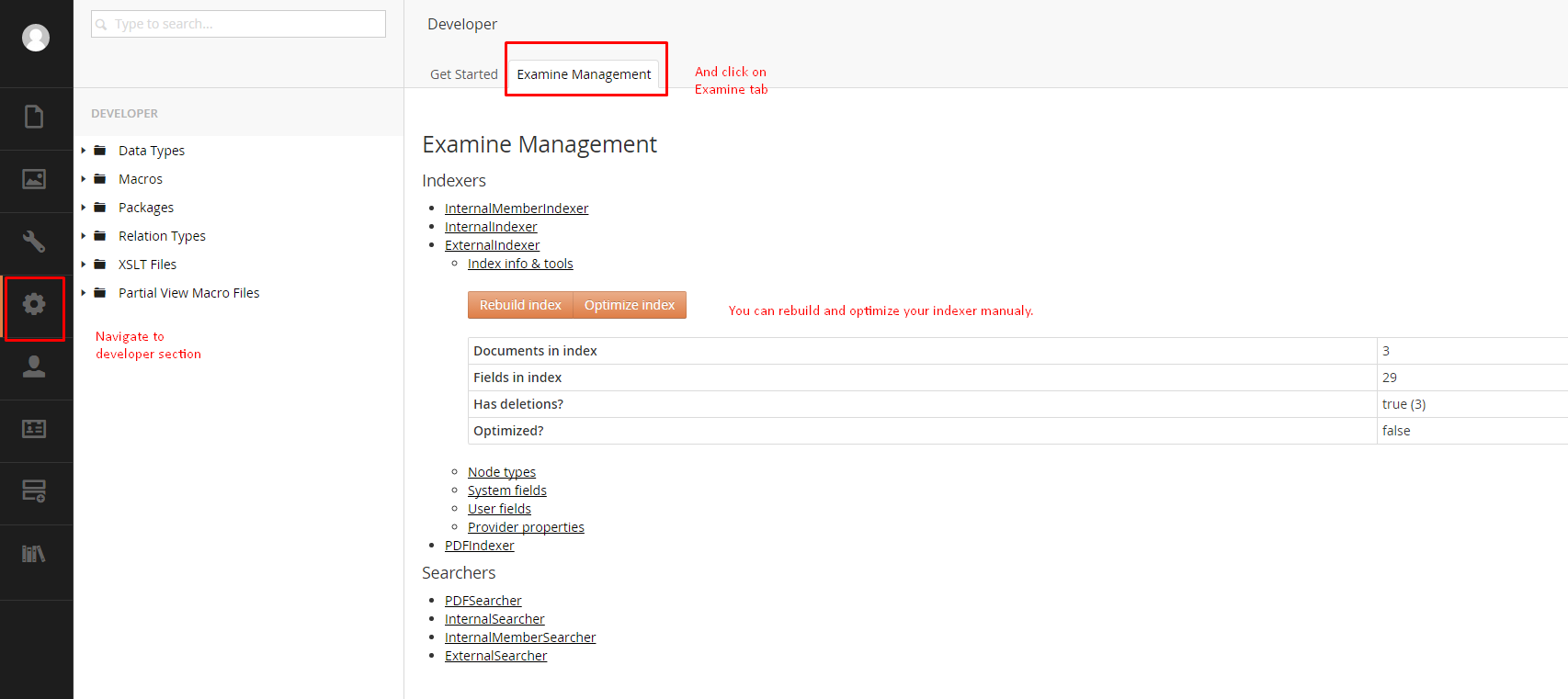
var luceneString = "+testTitle:" + term;

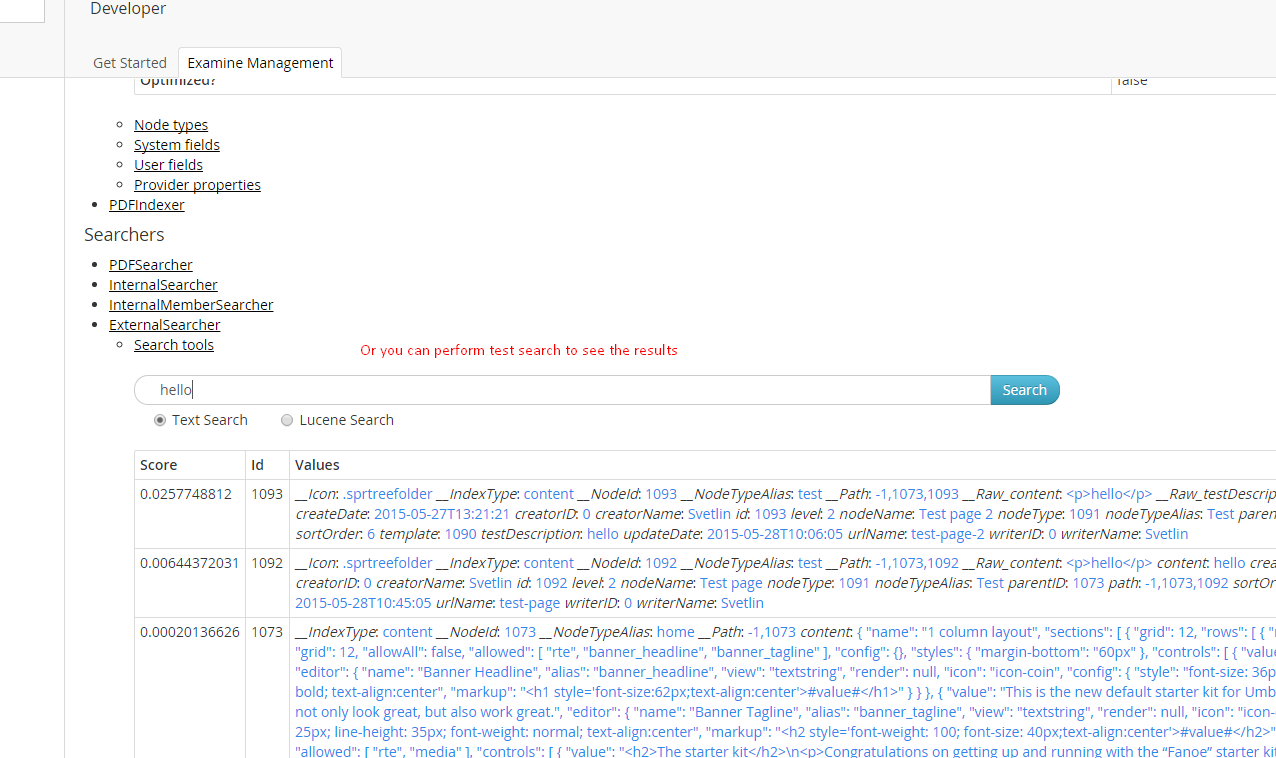
var rawQuery = searchCriteria.RawQuery(luceneString);

var searchResults = Searcher.Search(rawQuery);

2.4 Using Umbraco back office interface:

* 1. Examine dashboard in Umbraco back office





1. Additional modules
   1. Umbraco indexing for PDF files.

You can install this module via Nuget – use user interface or type this command in Packege manager console in VS:

Install-Package UmbracoCms.UmbracoExamine.PDF

The installer has added a new Indexer & Searcher to ~/Config/ExamineSettings.config

called: "PDFIndexer" & "PDFSearcher"

and added a new Index Set to ~/Config/ExamineIndex.config called "PDFIndexSet"

With this installer you can search in selected pdf files, using the Examine Search API like examples below. But you need to specify your searcher:

var searcher = ExamineManager.Instance.SearchProviderCollection["PDFSearcher "];

1. Negatives

Lucene is super fast search engine and after the Umbraco Examine steps of this platform I can not see any negatives.