Full Text Search

Project Description

Full Text Search is a fast, powerful and easy to set up search solution for Umbraco web sites.

Yes, but why?
In working for clients, we found that XLST Search didn't handle content replacement and wasn't flexible enough. The main alternative, Examine, required a fair amount of work and know-how to integrate into our sites. So, we decided to write something that can index an entire web page, in the same way that Google does, and allow searching that index with minimal setup and no additional code.

OK, tell me more
Full Text Search indexes the full content of published Umbraco pages, including any macros or shared content.
Out of the box, it allows searching of this index using fuzzy matching, quoted queries, context highlighting and more - all without you having to write a single line of code. It's intended to fill the gap between the ease of use of XSLT Search and the relative complexity of writing your own solution using Examine.

What are the specifics?

- Indexing is approached in a fashion similar to a search engine crawler: the full HTML produced by a page is read after publishing, tag stripped and added to the index. There are options to allow common content (menus, footers etc) to be excluded from the index.
- A Razor macro is used to format and display search results. This allows for easy modification of the display format of search results, again without having to touch any of the code.
- For more advanced users, there are options to customise search through various configuration options, as well as to modify indexing and result retrieval behaviour from your own code.

As a result of using Full Text Search, search should be lightning quick, even on sites with large amounts of content. This is because the package is essentially just an intelligent front end to Umbraco's Examine/Lucene.

Anything you're hiding from me that is going to screw-up my day?
Happily, no. But because we're honest people and like doing things properly, we will tell you that Full Text Search is technically beta quality. We love it and are confident about what it can do, but it hasn't been tested on enough sites to be considered completely bug free or stable yet.

Installation Instructions

Requirements

Umbraco 4.7 or higher and .NET 4

Pre Installation

Make sure Umbraco is properly set up and configured, and that your database user has rights to create tables (this should certainly be the case if you've gone through the default install procedure). The package creates a table called fullTextCache, you may want to check that doesn't already exist, in the unlikely event that some other package uses that name. Backup your database and files.

Install Package

Download the correct package version for your Umbraco installation. The package version number indicates the minimum compatible Umbraco version number for which the package should be installed. Currently, 3 versions are available:

- **Full_Text_Search_4.7.x.zip** for Umbraco 4.7.0 to 4.7.2
- **Full_Text_Search_4.8.x.zip** for Umbraco 4.8.0 to 4.11.x
- **Full_Text_Search_6.0.x.zip** for Umbraco 6.0.0 to 6.1.x

Once downloaded, login to Umbraco, go to the developer tab -> packages ->Install local package, upload the file and go through the installation prompts.

Indexing Configuration

There's a few things you'll have to set up for indexing to work.

You'll need to customise the config file at ~/config/FullTextSearch.config . There's a lot of options here, most of which can be ignored for now.

There are a couple of options which need to be correctly set for basic indexing to function.
Firstly, you need to set the correct URL/Hostname for your site in the config file. IIS will need to be able to resolve this URL/Hostname combination to your site. The easiest way to do this is to set the <HttpUrl> to point at default.aspx in your umbraco installations document root. e.g.

```
<HttpUrl>http://mysite.com/default.aspx</HttpUrl>
or
<HttpUrl>http://localhost:8080/default.aspx</HttpUrl>
```

Alternatively, If you're having problems with name resolution you can set the <HttpHost> field, which sets the host header indpendently of the URL called. e.g. set

```
<HttpUrl>http://127.0.0.1/default.aspx</HttpUrl>
<HttpHost>mysite.com</HttpHost>
```

If required, HttpUrl and/or HttpHost can also be overridden in the web.config by adding the following app settings respectively: FullTextSearchHttpUrl and FullTextSearchHttpHost.

After setting the URL, set <Enabled> to true. There's a lot of other options here, the file is reasonably well commented, and some are described in more detail in the settings and customisation part of this guide.
The package will also automatically add the necessary Umbraco Examine configurations in the ~/config/ExamineSettings.config and ~/config/ExamineIndex.config files. The default settings are basic - if you're familiar with examine feel free to customise to your taste...

These are the settings that will be added:

ExamineSettings.config
Added the following under <providers> in the <ExamineIndexProviders> section

```
<add name="FullTextIndexer" type="FullTextSearch.Providers.FullTextContentIndexer,
FullTextSearch"
runAsync="true"
supportUnpublished="false"
supportProtected="false"
interval="10"
analyzer="Lucene.Net.Analysis.Standard.StandardAnalyzer, Lucene.Net"
enableDefaultEventHandler="true"
indexSet="FullTextIndexSet"/>
```

Note that Full Text Search does not currently support indexing unpublished or protected pages. So leave these settings at false.
Added the following under <providers> in the <ExamineSearchProviders> section

```
<add name="FullTextSearcher" type="UmbracoExamine.UmbracoExamineSearcher, UmbracoExamine"
analyzer="Lucene.Net.Analysis.Standard.StandardAnalyzer, Lucene.Net"
indexSet="FullTextIndexSet"/>
```

ExamineIndex.config
Added the following under <ExamineLuceneIndexSets>

```
<IndexSet SetName="FullTextIndexSet" IndexPath="~/App_Data/TEMP/ExamineIndexes/FullText/">
          <IndexAttributeFields>
                    <add Name="id" />
                    <add Name="nodeName"/>
                    <add Name="nodeTypeAlias"/>
          </IndexAttributeFields>
          <IndexUserFields>
                    <add Name="bodyText" />
          </IndexUserFields>
          <IncludeNodeTypes/>
          <ExcludeNodeTypes />
</IndexSet>
```

Feel free to customise these settings to suit your site. Consult the Umbraco examine documentation for the meaning of the various settings. Briefly, "IndexUserFields" tells examine which user fields from the node to add to the index. "ExcludeNodeTypes" allows you to set a list of node type aliases to exclude.
Once you've done that indexing should basically work. You'll need to publish a page on the site to actually add your content to the index, as indexing is triggered on publish events.

There's a lot more settings that control indexing, most of these are detailed in the settings and basic customisation guide. One thing we'd definitely recommend you do is to change all your common

macros (stuff that appears on every page, main navigation, search box, etc.) so that they don't output on search indexing. That prevents searches for common terms turning up irrelevant results. See the linked guide for more details
If you're having problems with indexing, the following ideas could help with debugging...

1. Re-create the index by using the full text search tab in the developer section of the back office
2. Delete the index at ~/App_Data/TEMP/ExamineIndexes/FullText/ and republish to re-create
3. Use a tool like Luke to examine the index directly
4. Consult the Umbraco error log in the database. Indexing errors should show up there

Setting up your search form

An actual search page is not added by the package. But a template and a document type (both alias "FullTextSearchPage") are added to your site.

In order to get a basic search up and running on your site, all you need to do is add a page of type "Full Text Search Page" somewhere underneath your home page. The search form is automatically added to that page by the FullTextSearch macro present in the template. The template is not a complete HTML page, so you'll probably want to put it under whatever master page you're using. In order to have the search results correctly formatted you will also need to reference the FullTextSearch.css stylesheet.

The package adds a FullTextSearch macro and Razor file, this uses helper functions from the FullTextSearch search extenstions to query the Lucene index and return results/errors. The results and errors are then formatted for display by the Razor code.

The only thing that really needs to be present in your template for search to function is the macro FullTextSearch. You may also want to reference the FullTextSearch stylesheet, or copy its contents into your own.

There's a number of macro parameters that control how the search works and what gets output. These are documented in the XSLT file as well as the Settings and basic Customisation section.

If you're supplying search terms from a search box that's not on the search page, supply the search terms as a GET or POST parameter named "Search" (this is editable in the Razor file).

The default HTML for the form should be OK for most purposes, if it doesn't work for you it's entirely editable in the FullTextSearch.cshtml without having to touch any code.

The actual text displayed on the search page("Your search did not match any documents" etc.) is controlled by several dictionary entries, placed under "FullTextSearch" in the dictionary. Currently only English is included. Feel free to edit dictionary entries to your taste. Note that some of them use formatted strings in the c# string.Format style. What this means is that the numbers in curly brackets ({1} etc.) are replaced with a variable piece of information. It should be pretty obvious from the text what that info is. Remove these if you don't want/need that info.

Settings and Basic Customisation

Indexing Settings

Preventing Common Content from Appearing in the Index

On most sites you won't want menus, common titles etc. appearing in the search results. As we index the full page HTML that can happen here, so we've provided a few methods to prevent that.

For Macros:
There's a GeneralExtensions helper function that allows you to determine whether or not the page is being called by the search indexer. Simply call GeneralExtensions.IsIndexingActive() from your Razor. This doesn't do anything particularly sophisticated. It simply checks for a parameter that can be passed by query string or cookie, by default that parameter is called FullTextActive, this can be changed in the FullTextSearch.config file. To prevent your macros from outputting if search is active simply wrap them in the following if block...

```
if (GeneralExtensions.IsIndexingActive())
{
          //macro
}
```

More generally...
The indexer can also be configured to strip out defined parts of the HTML from the page output. This is controlled by the <TagsToRemove> and <IdsToRemove> settings in the FullTextSearch.config file. So, for example, by default any HTML element with an ID of "mainNavigation" and all it's children are stripped from the output and will not appear in the index. This is a quick and simple way to prevent certain content from appearing in search results.

Preventing Certain Document Types from Appearing in the Index

This can be accomplished in a couple of ways. Adding the node type alias to ExcludeNodeTypes in the ExamineIndex.config file disables all indexing for a specific node type.
Adding the node type alias to the <NoFullTextNodeTypes> list in FullTextSearch.config will prevent the page HTML from being rendered and included in the index, but not prevent any other node properties from being included.

Preventing Certain Pages from Appearing in the Index

It's common practice in Umbraco to have a Property "umbracoNaviHide" that prevents pages from showing up in the main navigation. We've adapted this, and any page with a property of "umbracoSearchHide" won't show up in the index or search results. So this is a useful way to prevent individual pages from being searched. The name(s) of the property(s) that disable indexing for a page are specified in FullTextSearch.config under <DisableSearchPropertyNames>

Changing the indexing method

FullTextSearch has two ways in which it indexes pages.

1. HTTP Renderer(default)
   This fires off web requests to an address and hostname specified in FullTextSearch.config.
2. Programmatic Renderer
   This uses a .net Server.Execute command to render page content. This means it can only be run with an active HTTP Context (i.e. synchronously while publishing is taking place). It's theoretically neater, and was initially intended to be the only renderer, but the requirement that it run synchronously can make publishing painfully slow on large sites.

Generally the default HTTP Renderer should work well enough, if you're having problems with it switch it for the Programmatic renderer in the FullTextSearch.config file, remembering to also set PublishEventRendering to true to ensure there is an active HTTP Context for it to work with.

Further Customisation

It's possible to override the default renderers from you own code, as well as hook into several indexing events, in order to customise the indexing process to your needs. Have a look at the advanced customisation instructions.

Search Settings

Macro Parameters

There's a lot of macro parameters to the FullTextSearch.cshtml file, they control how the index is searched, and how the results are output. A list of them, and their use follows...

queryType
Type of search to perform. Possible values are:

MultiRelevance ->
The default. The index is searched for, in order of decreasing relevance

1. the exact phrase entered in any of the title properties
2. any of the terms entered in any of the title properties
3. a fuzzy match for any of the terms entered in any of the title properties
4. the exact phrase entered in any of the body properties
5. any of the terms entered in any of the body properties
6. a fuzzy match for any of the terms entered in any of the body properties

MultiAnd ->
Similar to MultiRelevance, but requires all terms be present

SimpleOr->
Similar to MultiRelevance again, but the exact phrase does not
get boosted, we just search for any term

AsEntered->
Search for the exact phrase entered, if more than one term is present

Note that quoted queries are correctly processed in all search modes except AsEntered.

Other special query types (Boolean, wildcard, etc), are not supported as yet.

titleProperties
A comma separated list of properties that are part of the page title, these will have their relevance boosted by a factor of 10 defaults to nodeName. Set to "ignore" not to search titles.

bodyProperties
A comma separated list of properties that are part of the page body. These properties and the titleProperties will be searched.
defaults to using the full text index (FullTextSearch by default) only

summaryProperties
The list of properties, comma separated, in order of preference, that you wish to use to create the summary to appear under the title. All properties selected must be in the index, cos that's where we pull the data from.
Defaults to Full Text

titleLinkProperties
The list of properties, comma separated, in order of preference,that you wish to use to create the title link for each search result.
Defaults to titleProperties, or if that isn't set nodeName

rootNodes
Comma separated list of root node ids Only nodes which have one of these nodes as a parent will be returned.
Default is to search all nodes

contextHighlighting
Set this to false to disable context highlighting in the summary/title. You may wish to do this if you are having performance issues as context highlighting is (relatively) slow.
Defaults to on.

summaryLength
The maximum number of characters to show in the summary.
Defaults to 300

pageLength
Number of results on a page. Defaults to 20. Set to zero to disable pagination.

fuzzyness
Lucene Queries can be "fuzzy" or exact.
A fuzzy query will match close variations of the search terms, such as plurals etc. This sets how close the search term must be to a term in the index. Values from zero to one. 1.0 = exact matching. Note

that fuzzy matching is slow compared to exact or even wildcard matching, if you're having performance issues this is the first thing to switch off.
Defaults to 0.8

useWildcards
Add a wildcard "*" to the end of every search term to make it match anything starting with the search term. This is a slightly faster, but less accurate way of achieving the same ends as fuzzy matching. Note that fuzzyness is automatically set to 1.0 if a wildcards are enabled.
Defaults to off

Further customisation

If the existing macro isn't flexible enough for you, have a look at the advanced customisation file, which contains some information on how to control output. There's an event that's called just before output which allows you to modify search results from your own code, or you can call the search functions from your own user controls rather than rely on the Razor Helper.

Advanced Customisation

For projects where the default configuration options are not sufficient several methods have been provided to allow you to customise the package from your own code.

Events

Examine Indexing Events

You can hook into any of the Umbraco examine events from your own code in the usual fashion. This allows control over the indexing process. For example you can edit the rendered HTML before it gets indexed, or cancel indexing on certain pages.

Firstly you'll need to subscribe to the events. This means running some code at Umbraco start-up, the usual way to accomplish this is in the constructor of a class that extends umbraco.BusinessLogic.ApplicationBase.
So, very very briefly, start an empty visual studio web app project, reference the relevant Umbraco dlls, examine dlls and the Governor.Umbraco.FullTextSearch dll, then create something like the below, build it, and stick the dll in your Umbraco bin directory. Consult the Umbraco site for better documentation on this that took more than 5 minutes to write...

```
namespace MyAddon
{
          public class MyEvents : umbraco.BusinessLogic.ApplicationBase
          {
                    public MyEvents()
                    {
                              BaseIndexProvider indexer =
ExamineManager.Instance.IndexProviderCollection["FullTextIndexer"];
                              indexer.NodeIndexing += new EventHandler(indexer_NodeIndexing);
```

```
                    }

                    void indexer_NodeIndexing(object sender, IndexingNodeEventArgs e)
                    {
                              // do stuff
                    }
          }
}
```

Depending on your version of Umbraco, umbraco.BusinessLogic.ApplicationBase might be deprecated and you'll need to use the appropriate method.

There are a number of events built into examine, documentation doesn't seem to be readily available, but you can look through the code, or just use intellisense to pick them out. If you want to edit the rendered HTML or cancel indexing then NodeIndexing is the one to use.

Search Results Event

There's an event in the search helper that's fired once for every search result.
You can access this at SearchExtension.ResultOutput . This will allow you to change/add/delete any of the content that's about to be pushed out to the Razor, giving you more flexibility in what to display.

Swapping Indexers and Renderers

It's also possible to override the way FullTextSearch renders, and then indexes pages completely, or just for a certain node type.

To get a sense of the way it works you'll probably need to look over the code-base for FullTextSearch. But some basic information is included here that should be enough to get you started.

To clarify, the renderer reads the page's full HTML and returns it.
The indexer controls which pages get indexed and how the HTML is stripped.
When the renderer runs is determined by the PublishEventRendering setting in FullTextSearch.config, if this is set to true it will run immediately after publishing, and a page publish won't return until the renderer does. If PublishEventRendering is set to false (this is the default setting) then the default indexer will call the renderer during the indexing phase, and page publishing doesn't have to wait for the renderer to return.

Renderers

FullTextSearch can use any class that implements the interface IDocumentRenderer to render documents. You can replace the default renderer entirely, inherit and extend/override it, and do either of these conditional on the type of node being rendered. An example will probably make this easier to understand...

Override Renderer for node type "umbHomepage" to add the text "Dummy Homepage Text" onto the end of the full text index

```
public class MyEvents : umbraco.BusinessLogic.ApplicationBase
{
          public MyEvents()
          {

          FullTextSearch.Manager.Instance.DocumentRendererFactory.Register<MyRenderer>("umbHom
epage");
          }
}

public class MyRenderer : DefaultHttpRenderer
{
          public override bool Render(int nodeId, out string fullHtml)
          {
                    if (base.Render(nodeId, out fullHtml))
                    {
                              fullHtml += "Dummy Homepage Text";
                              return true;
                    }

                    return false;
          }
}
```

Override renderer for all node types
`FullTextSearch.Manager.Instance.DocumentRendererFactory.RegisterDefault<MyR enderer>();`
Have a look at the code to get a better idea of how it works. One thing to bear in mind is that you can only use the Umbraco node factory API if PublishEventRendering is set to true in the FullTextSearch.config file. Otherwise you'll need to use the Document API.

Indexers

Indexers can be overridden in much the same way as renderers. e.g.
`FullTextSearch.Manager.Instance.FullTextIndexerFactory.Register<MyIndexer>( "someNodeTypeAlias");`
The class "MyIndexer" will need to implement IFullTextIndexer.
Bear in mind that if PublishEventRendering is not active your indexer will need to handle calling the appropriate renderer, or just inherit the existing DefaultIndexer class.

Scheduled Indexing

In the event that content on your site changes regularly without any nodes being published a web service is provided which allows you to

1. Trigger a full re-creation of the site index
2. Re-index a list of nodes

It is envisaged that this service be called by means of a scheduled task of some kind. No mechanism for doing this is currently provided, though a console application may be added in future.

The web service is provided in a separate package to the main FullTextSearch. You can download it from the downloads page.

It requires the file umbraco.webservices.dll, which is not present by default on some umbraco installations. If your install lacks this file you can build it yourself from the umbraco source, or download it from the downloads page for Umbraco 4.7.0

The web service is located at /umbraco/webservices/FullTextService.asmx . There are four methods of interest:

RebuildFullTextIndex(string username, string password) : Trigger a full re-indexing of the site. Be warned that this deletes the current index completely. So search will be unavailable until re-indexing is completed, which happens at a rate of about 5 pages/sec on my development box.

ReindexFullTextNodes(string username, string password, int[] nodes) : Re-index the supplied list of nodes

ReindexFullTextNodesAndChildren(string username, string password, int[] nodes) : Re-index the supplied list of nodes and all their children.

ReindexAllFullTextNodes(string username, string password) : Re-index all nodes one-by-one.

You can also trigger these actions from your own code using the FullTextSearch.Admin.AdminActions class