Forward Search 2.7

Developer Guide

“Getting started with Forward Search”

Document version 2.7.0.1

Table of contents

Introduction 2
Forward Search System overview 2
Forward Search 3
Installation and configuration 4
Install 5
Configuration 6
Configuration of website source 6
WebService configuration 7
Tagging of content 8
Marking what to index 9
The Robots Exclusion Protocol 10
Robots Meta Tags 10
The fields of the index 11
Predefined field overview 11
Collecting information 12
The TEXT field 12
Dublin Core 12
Extended custom fields 12
Mapping a custom field 13
Language Identification 13
Last modified 13
Categorize webpages based on url 14
Custom metadata mapping (deprecated) 14
Geo Search fields 15
Run Forward Search Crawler 16
Manual trigger 16
Schedule 16
Event trigger 17
Search: Use of the Forward Search WebServices 18
User-control, XSLT or Ajax 18
Search by User-control 18

The other search interfaces ... 19
Engine commands ... 19
Building queries ... 19
The Query language ... 20
Facet counting ... 23
Did-You-Mean ... 23
Type-ahead ... 24
Tuning ... 25
Using a Thesaurus ... 25
Stop words ... 26
Maintenance ... 27
Re-index ... 27
Logs ... 27
Appendix ... 30
Productivity tools in the installation package ... 30
A sample custom thesaurus ... 30
Using and configuring the debug logger ... 32
A sample search query ... 33
Subpage items ... 35
Troubleshooting ... 36

Introduction

This document describes the many aspects of installing, configuring and using the Forward Search web and enterprise search system. Its intended audience is developers and web administrators in charge of constructing and maintaining one or several Forward Search indexes.

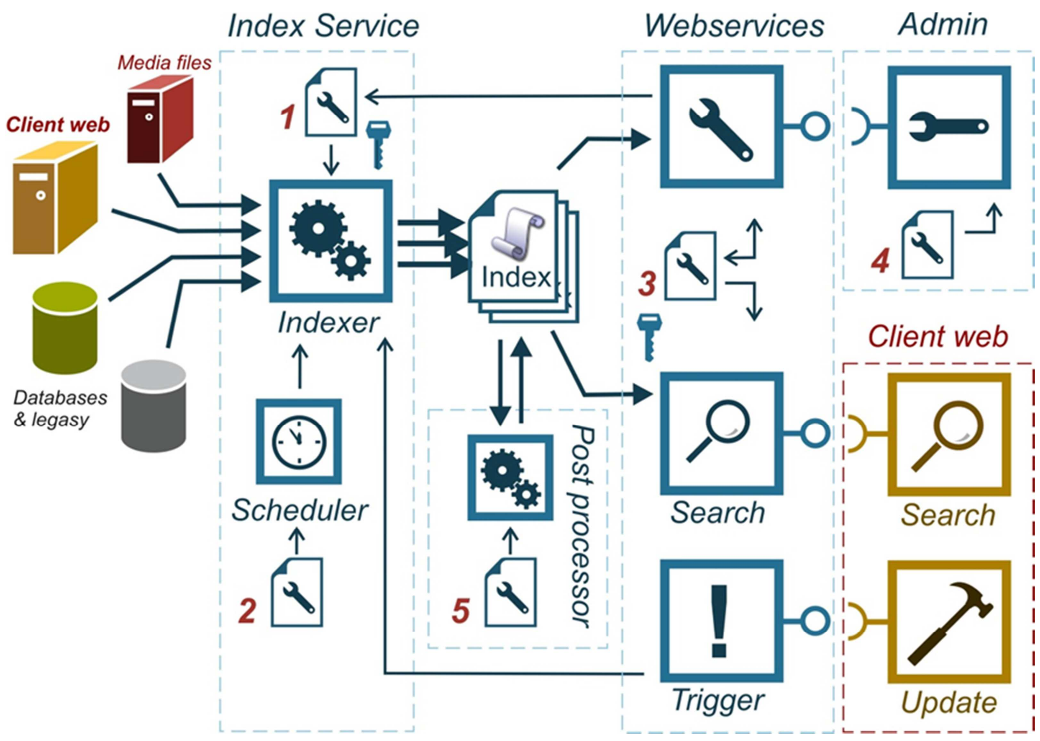
Forward Search System overview

The Forward Search System consists of 6 components:

- the Crawlers and Connectors
- the Index service
- the Index
- the Post Processer service
- the WebServices
- the Admin interface

These elements and their relationship and dependencies with each other and the surrounding system can be seen in the following model overview diagram, and described further below. Blue elements denote Forward Search components.

The configuration entities are marked with red numbers, and are referred to in the text below. The required presence of a valid license file is marked with a key.



Forward Search

Forward Search is a crawler-based search solution that allows for very fast searches into a large set of diverse content.

The first 2 components are “the Crawlers and Content Connectors” and the Index service. The solution is crawler based which means that content is "crawled" and indexed in a separate process not related to the searches. This process Crawls various content sources and constructs an index. This can be done by either event update, scheduled update or a combination of those.
Scheduling is typically done during low-activity hours and controlled by the Forward Search Scheduler. Event update is occurring every time new content is added, altered or deleted and subsequently published.

The third component is the index itself. The index is a large binary file constructed and maintained by the Index Service. The index contains all the discovered words and pointers to the actual content. It also contains a lot of other important and useful data about the content. When a search is performed, the search only needs to look up the search criteria in the index and return the information found herein. It doesn’t have to look into the actual sources at all.

The fourth component is the post processing service. It’s a service that allows for manipulations to the index after it was build. The processes placed in the post processor is basically the same as can be handled in the document finalizer of the index-server, but typically, the post processing service runs independently of the index and searching, in a separate process that can be set at low priority, slowly chewing through the index, processing the documents one by one. It is therefore ideal for slow processes, which would otherwise impede the availability of the index and the speed of the re-indexing. Please refer to separate document on how to configure and use the post processor.

The fifth component is the set of WebServices. There are 5 service endpoints in the Forward Search WebServices.
The first "forwardsearchwebservice.asmx" is the most important service, since it provides the functionality for which Forward Search is known - supplying fast and flexible search.

The Admin service is an extension of the search service, with additional features.

The Trigger service allows for calling the index service whenever new or changed content is published from the source (e.g. CMS system).

The event service makes it possible to trigger index updated from events raised elsewhere.

Injection service makes it possible to fetch an index scheme and update documents in the index directly.

The last Statistic service is administrative services used by the Forward Search Admin client.

The final component of the Forward Search is the Admin interface, the topic of this user-manual. The purpose of the admin interface is to inspect the index and manage the configuration of the system. This management is done by exposing functionality that allows changes to the configuration files of the index server and the Web Services. These files are marked "1" and "3" in the diagram. These two files are referred to as the Indexserver Config.xml and the Webservice Config.xml, since they both have the filename config.xml.

Also the scheduler configuration (sch.xml), marked with a "2" in the diagram allows for maintenance from the admin interface, even though it is not as evident on the diagram. The scheduler controls when the various search-sources are crawled and re-indexed by the index server.

Installation and configuration

Below, the installation and configuration procedures are described in some details. The basic steps are:

Install

1. Download
2. Unblock
3. Unzip
4. WebService setup
5. Indexservice setup
6. Administration client install

Configuration

7. Index & source setup
8. Run the indexer

Install

To Set up Forward Search, complete the following steps and tasks:

1. **Download** the latest available version of Forward Search from http://portal.forwardsearch.dk (Choose the menu item "Technical Resources" and sub-menu "Download". The download is labelled "Release - Forward Search Backend - 2.7.x.x – Full" followed by the release date. The file itself is a zip file named "forwardsearch_version_2.7.x.x.zip" and is a little less than 30 MB large)

2. **Unblock** - Remember to "Unblock" the zip-file first, if applicable.

3. **Unzip** the file to a chosen location on the server file system. This location is the installation "approot". Apart from the main Forward Search modules, a few other things are unzipped as well. Please refer to the appendix for more on these productivity tools.

4. **Web services setup**
   a. Apply the needed rights to the [approot]\WebSite folder.
   b. Change paths in the [approot]\WebSite\Data\web.config file. The 5 paths to change are located under the <appSettings> entry. They instruct the WebServices about where to find the index server files.
   c. Place the received license file to the folder [approot]\WebSite\data. The license file must be renamed to "lic.xml".
   d. Create a website pointing to this folder. This website becomes the "WebService url". Make sure it runs under an account that has write access to the website- and index folders.

5. **Index service setup**
   a. Change paths in the configuration file: [approot]\Forward.IndexServer\data\Config.xml. The paths to change are those 5 paths found in the section "<appDataProcessSettings>", pr. default pointing to "C:\ForwardSearch..."
      Also Change the url in the configuration file, entry <WebServiceNetworkAddress> to: [webservice url]/forwardsearchWebservice.asmx
   b. Place the received license file to the folder [approot]\Forward.IndexServer\data. The license file must be renamed to "lic.xml".
   c. Register the index-server: Run [approot]\Forward.IndexServer\ ServiceInstaller.cmd. This should make the forward index-server appear as "started" and "automatic" in the windows "Services" list.

6. **Administration Client install**
   Chose the version of the administration client to use:
   - Standalone version
   - Sitecore version
   - EPiServer version

   Download it from http://portal.forwardsearch.dk and install it at a chosen location. Please refer to the relevant version of the "Admin package" document for more details on the installation.

Configuration

To setup a target website for search indexing, complete the following steps and tasks

7. Index & Source setup.
   a. Edit the [approot]\Forward.IndexServer\data\Config.xml to include a "websetting" section (also referred to as a "source") relating to the website to be indexed. You can copy the demo site section already present in the file. See "Configuration of website source" below for more details on the settings.
   b. Edit the [approot]\WebSite\data\Config.xml to include a "database" section relating to the index you are about to create. See further details in "WebService configuration" below.

      Notice: We strongly recommend that you use the administration client to create new sources and indexes. That way, the configuration files are kept consistent and all entries are validated.

8. Run the indexer. You can start the indexer from the administration client, or you can run the trigger executable: [approot]\Forward.IndexServer\Forward.IndexServer.Trigger.exe

Configuration of website source

Each website source has its own entry in the index-server configuration file, under the top-entry "webSettings". File-sources are located under the top-entry "fileSettings".

The entries available and thus needs configuration are listed in the table below, with an example of the entry as it appears in the config file, and the name under which it is edited in the Administration Client. We also refer you to the "User Guide Admin Client" page 18 for more on the source configuration fields.

Label	***Description***	***Sample***
Source name	The name, displayed, not actually used.	<webSetting name="SomeName">
Source id	The unique id of the source.	<id>1500</id>
Index name	The name of the target index	<indexname>MyIndex</indexname>
Source enabled	Is this source enabled?	<enable>True</enable>
Website root	The starting url of the crawler.	<webSite>http://www.mysite.com/en/</webSite>
Exclude patterns	Which patterns can NOT appear in the url?	<excludePattern>.js,.css,.gif,.jpg,.png </excludePattern>
Include patterns	Which patterns MUST appear in the url? (leave blank to skip include-pattern test)	<includePattern>/en<includePattern>
Ignore trailing slash	Consider urls with and without trailing slash to be the same page?	<ignoreTrailingSlash>True</ignoreTrailingSlash>
Remove Anchor Text	Don't index text inside anchor tags.	<removeAnchorText>True</removeAnchorText>
Default Language	Use this language if none other found.	<defaultLang>en</defaultLang>
Path to Index	Where is the binary index file?	<pathForwardIndex>C:\forwardsearch\ForwardData\mySite\Index </pathForwardIndex>
Path to dump files	Where should temporary dump files be stored?	<pathDump>C:\forwardsearch\ForwardData\Dump</pathDump>
Event WebService	The url of the event webservice	<pathEvent>http://mywebservice.local/eventservice.asmx</pathEvent>

Use Preview	Store the first page as html in the "PREV" field	`<preview>False</preview>`
Generate thumbnail	generate an image of the first page	`<thumbnail>False</thumbnail>`
Thumbs normal size	Set to True to use 300px thumbnails. False = 100px.	`<thumbnailNormalSize>False</thumbnailNormalSize>`
Use Inherit	Apply inheritance to media documents.	`<useInheritDocumentDetails>False</useInheritDocumentDetails>`
Network credentials	Use these credentials to access protected files.	`<networkCredential>` `<domain />` `<user />` `<password />` `</networkCredential>`
Partial crawl	Only process a part of the website.	`<partialCrawl>False</partialCrawl>`
Partial crawl count	Number of pages to process in a partial crawl.	`<partialCrawlCount>500</partialCrawlCount>`
Partial crawl resume	True: When index is run again, continue the partial crawl. Else start over from the first page.	`<partialCrawlResume>False</partialCrawlResume>`
User agent	The user agent name of the Forward Search crawler.	`<userAgent>ForwardSearchBot</userAgent>`
Forms Authentication	Set up login information for access to protected web pages.	`<formsAuthentication/>`
Stay home	True: Don't leave the current domain when following links.	`<stayHome>True</stayHome>`
Crawler Depth	If "stay home" is false, only move this number of links away from domain.	`<crawlerDepth>1</crawlerDepth>`
Crawler Timeout	Expire a page request after this time (milliseconds)	`<crawlerTimeOut>500000</crawlerTimeOut>`
Fieldmap name	Name of extended custom field map to apply	`<extendedFieldMap>ReferencesMap</extendedFieldMap>`
Post processing script	Path to a script og executable that will be run whenever the indexer completes an indexing.	`<postIndexingScript>RunDocMod.cmd</postIndexingScript>`

WebService configuration

Each index in the Forward Search system has its own "database" entry in the WebService configuration file, under the top-entry "databases".

The entries available and thus needs configuration are listed in the table below, with an example of the entry as it appears in the config file, and the name under which it is edited in the Administration Client.

We also refer you to the "User Guide Admin Client" page 14 for more on editing the indexes.

Label	***Description***	***Sample***
Name		`<name>SampleSite</name>`
Index Path	Where is the binary index file? Same value as the index-server config entry "pathForwardIndex"	`<config> C:\forwardsearch\ForwardData\MySite\Index</config>`
Provider	Integer that indicates which search provider to use. Default is 2; the normal	`<provider>2</provider>`

	Forward Search index. This is the only option for most solutions.	
Multilingual thesaurus	Should the multilingual thesaurus be used for this index? Notice, that the multilingual thesaurus requires a specific license.	<UseMultiLingThesaurus>False</UseMultiLingThesaurus>
Custom thesaurus	Full, local server path to the custom thesaurus file.	<cusths>C:\forwardsearch\WebSite\Data\MySite\cusths.xml</cusths>
Stopword	Full, local server path to the stopword list.	<stopword > C:\forwardsearch\WebSite\stopword_en.lst</stopword >
Fieldmap name	Name of extended custom field map to apply	<extendedFieldMap>ReferencesMap</extendedFieldMap>
Popular terms	Number of days back to maintain popular search words.	<popularTermsDaysBack>2</popularTermsDaysBack>
Popular terms start-weight	A decimal: 0.0 < X <= 1.0. The weight relative to the youngest terms that the oldest terms in the popular list should have. 1.0 = equal weight, 0.5 = half the weight and so forth.	<popularTermsStartweight>0.5</popularTermsStartweight>

Reduce text returned as a search result, particular content of the index field TEXT might be huge.
It is possible to reduce the amount by referring to a section in the WebService configuration, like this "TITL,URL,TEXT@A" as display paragraphs argument for the search service.

```
<reduceTextSections>
  <section name="A">
    <reduceText>...</reduceText>
    <countInWords>true</countInWords>
    <count>25</count>
    <returnFromStart>false</returnFromStart>
  </section>
  <section name="B">
    <reduceText>...</reduceText>
    <countInWords>true</countInWords>
    <count>25</count>
    <returnFromStart>true</returnFromStart>
  </section>
</reduceTextSections>
```

In this case TEXT will only contain 25 word with the query centred within
Section B would return the first 25 words from the field Count in words false would return 25 characters

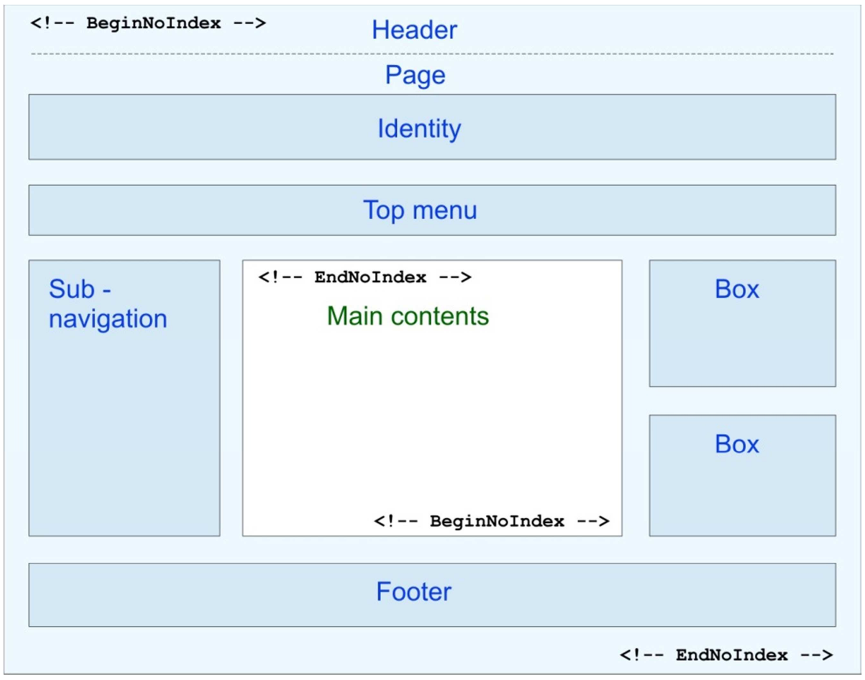
Tagging of content

The process of marking up the content pages with various tags and information in order to enrich the index and thus enhance search capabilities is called "tagging the content". It consists of two tasks: Marking what to index and Emitting extra information.

Marking what to index

You can, and should control what part of the webpages that should be indexed as text - that is, what part of the html should be tokenized and index into the "TEXT" field in the index. You use the de-facto standard comment tags `<!--BeginNoIndex-->` and `<!--EndNoIndex-->` to do this. (No whitespace)

The standard procedure is to exclude everything on a page and then "un-exclude" the main contents only. This approach is illustrated in the figure below. A "BeginNoIndex" starts the page and an "EndNoIndex" terminates the page. In the control or placeholder or other rendering component that emits the main contents html, an "EndNoIndex" starts the emission, and a "BeginNoIndex" terminates it. The result is that only the text within the Main Contents is extracted from the webpage and added to the TEXT field.



You should normally always exclude from being indexed into the TEXT field, the following page elements:

- Headers – identity and similar, that appears unchanged on all pages. (Since the text appears on all pages, it doesn't add any find ability to the documents in the index)
- Navigation elements. (The link text should appear on and be indexed as part of the page to which the link leads. This also ensures that the link text can only be found by authenticated users, if the page is protected)
- Content from teasers, panel-boxes, news-links, banners etc. (These elements should be indexed on their own pages instead. A notable exception is panel-boxes containing trivia or other facts or perspectives on the main content, that does not natively belong to any other page, or is highly relevant for the page where they appear.)

The “No-follow” directive

You can also mark up any area of the webpage that contains links the crawler should not follow. . You use the de-facto standard comment tags `<!-- BeginNoFollow -->` and `<!--EndNoFollow -->` to do this.

Alternately, you can exclude a single anchor from being followed by setting the “rel” attribuite to “nofollow”.

The Robots Exclusion Protocol

You can indicate which parts of the site should not be visited by a robot – including Forward Search - by providing a specially formatted file on the site, in a file called ”robots.txt”. Read more about the robots.txt file here: http://en.wikipedia.org/wiki/Robots_exclusion_standard

Robots Meta Tags

You can also exclude an entire page from being indexed or “followed”, by setting the de facto standard meta tag “robots:

NoIndex:	<meta name="robots" content="noindex">
	<meta name="robots" content="noindex,nofollow">
NoFollow:	<meta name="robots" content="nofollow">
	<meta name="robots" content="noindex,nofollow">

On a more fine-grained level, It is possible to exclude links in blocks (like BeginNoIndex—EndNoIndex) with the comment tags `<!--BeginNoFollow--> <!--EndNoFollow-->`

Or even for single anchors: Rendering the anchor attribute ”rel” with the value ”NoFollow” prevents Forward Search from following the link:

```
<a rel=”NoFollow” href=”/not/searchable/content.aspx” >Don’t follow!</a>
```

The fields of the index

An index consists of a number of fields with a number of values - terms – pointing to the documents that contains these terms. Many of the fields are predefined in Forward Search, and are shared among all indexes. Other fields are defined pr. index and exist only in the index they were designed for.
The field table in the next section lists all the predefined fields, and indicates how Forward Search attempts to collect data for them.

Predefined field overview

The table below lists the predefined index fields, their common name, usage and the possible matching tags from where the data for the field is harvested - mainly as Meta tags. An “X” in the “All” field indicates that this field will be included in “All” searches – searches that have targeted no specific field.

Notice that the predefined field names are with a few exceptions all 3 or 4 letter names, and all in uppercase.

Field name	***Common name***	***All***	***Source***	***Content***
TITL	Title	X	dc.title or <title> tag	Title of document
TEXT	Text	X	All text not marked “NoIndex”	Indexed text of document (normally, the entire text of the page minus header, footer and navigation.)
DOCE	Extension		From url	Document Extension (htm, pdf, docx or similar)
URL	Url		From url	The url of the document
AUTH	Author	X	dc.creator - dc.contributor - author	The Author of the document
CATG	Category	X	Url (via the CategoryRules file) - dc.category	The category of the document. May be auto-generated using the category-rules setup; see the developer guide about this.
KEYW	Keywords	X	dc.keywords - keywords	Any keywords from the metadata of the document.
SUBJ	Subject	X	dc.subject – subject	Any subject from the metadata of the document.
SIZE	Size		The file size	The document size in bytes
MODT	Modified		The modified date	The last modified date of the document, if found.
LANG	Language		See above.	The document language set explicitly or defaulted.
ITID	Item id	X	The CMS item id	A possible document id (typically relevant for CMS)
SIDS	Subitemid		Id of the subitem	Content is fetch from Subitem tag, look at appendix
COMM	Comment	X	dc.description – comments – description	Any comment or description from the metadata of the document.
ACLS	Access control		Emitted access control list, if present.	Access control list values
HASH	Hash		Hashkey of url	The hash key used for storing dump files and uniquely identifying the document.
BIN	Binary		From the document	True or false: Is the document a binary?
PARENT	parent		Determined by crawler	The parent url
SITE	Domain		Url	The main url of the website to which the document belongs
PATH	Local path		-	The local path to the index server copy of the document.

RREF	Referring pages		-	The number of referring pages.
CUSA …CUSO	Custom field A to O	X	Meta tags	The 15 "old" custom fields. Notice, that these 15 custom fields will be made obsolete. Use the Extended Custom fields instead.
ECxx	Extended custom field		Meta tags, html tags.	Extended custom fields with automatic naming based on the field id. The "xx" is replaced with letters "A" thru "J", denoting numbers 0-9. Notice, that Extended custom fields can be labeled individually.
GLOC	Geo locality		Meta Tag	The geo locality, if any.
GLAT	Latitude		Meta tag or html tag	The latitude found for the document.
GLON	Longitude		Meta tag or html tag	The longitude found for the document.
GNxx	Geonet xx		Calculated field.	Geo net fields. For each geo-net (denoted by the xx, just like in the extended custom fields above) contains the actual box that the document is mapped to.

There are more fields stored in the index, but most of those not shown are system-fields where the content is not adequate for viewing.

Collecting information

When Forward Search process content/documents, it collects information regarding the document, based on a number of rules.

- Fetching Meta tags in the order they are listed in above table, Dublin core before normal tags.
- Extended custom field setup
- Enrichments made by custom code ("Categorize webpages based on url", is a plugin sample)
- Part of url or language embedded in query string

Everything else might go into TEXT.

The TEXT field

Any text content on the page not excluded (with BeginNoIndex or anchor-tag exclusion for instance), and not detected as Meta Data, is harvested from the document and placed in the TEXT field, as plain text. (Any html tags are removed)

Dublin Core

Forward Search tries to identify Dublin Core meta-tag fields. It is possible, through the use of extended custom fields, to address even more of the Dublin Core field definition space. In the field table below, Dublin Core fields are defined starting with the prefix "dc." as specified by the standard.

Read more about Dublin Core here: http://dublincore.org/

Extended custom fields

it is possible to can use an arbitrary number of extended custom fields which are manageable from the Administration client.

Mapping a custom field

Thus, a field specification looking like this:

```
<field indexfield="AGENCY" >dc.publisher</field>
```

Will look for data in a field on the page like this:

```
<meta name="dc.publisher" content="John Doe" />
```

Store the data in the index like this:

```
AGENCY:John Doe
```

Tokenized as two individually searchable words:

```
AGENCY:John
AGENCY:Doe
```

For more information on use of extended custom fields consult the Forward Search Extended Custom Fields document available on the Partner Portal.

Language Identification

A dedicated plug-in detect the language (if not already detected) on the document element. The detected value will be set on the predefined document element field “LANG”.
The applied Language detection rules are, in prioritized order:

1. Query string : `?lang=XX` (where XX is the language code)
2. Html-tag: `<html xmlns="http://www.w3.org/1999/xhtml" lang="XX" xml:lang="XX" >` (where XX is the language code)
3. Analyze based on different language characteristics.
4. If all else fails, it uses the default value from configuration file.

Last modified

Forward search look at the timestamp last modified to determine if a document should updates in a series of incremental crawls.

The timestamp can be supplied an http header (normal for binaries) or as a Meta Tag:

- <META HTTP-EQUIV="Last-Modified" CONTENT=“date"/>

These 3 formats are supported:

- Tue, 15 Nov 1994 12:45:26 GMT
- yyyyMMddHHmmss
- yyyyMMddTHHmmss

Read more about Last modified here:
http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.29

Categorize webpages based on url

In order to achieve a simple categorization of your pages – populating the "CATG" field – you can set up a mapping of the urls to a set of page categories. Follow these steps to set up a category mapping:

1. Edit [apppath]\Forward.DataProcess.PlugIns\CategoryRules.xml
2. Specify the domain for which the category rules should apply
3. For each category to include, create a "category" entry with the category name.
4. For each url pattern that should map to this category, add a "pattern" entry with the url pattern (a full or part of the url – like "/news" or "/blog".

When a document is processed by the indexer, it will check the patterns in the file with the document url. If it matches, that category name is stored in the CATG field. The first url pattern to match is used.

This feature is implemented as an example of a finalizer plugin, the project files is located in [apppath] \Tools

Below is a small sample category-rules file for reference:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<rules>
  <www.mysite.com>
    <category name="Product">
      <pattern>/Products</pattern>
    </category>
    <category name="Solutions">
      <pattern>/Solutions</pattern>
    </category>
    <category name="Services">
      <pattern>/Services</pattern>
    </category>
    <category name="NewsEvents">
      <pattern>/Newslist</pattern>
    </category>
    <category name="InvestorRelations">
      <pattern>investor%20relations</pattern>
      <pattern>investor relations</pattern>
    </category>
  </www. mysite.com>
    <excludePattern>http://localhost/test</excludePattern>
</rules>
```

A sample CategoryRules.xml file

Custom metadata mapping (deprecated)

Available for backward compatibility is also the 15 "old" custom fields called "CUSA" to "CUSO". These fields can be mapped to meta-data using a mapping-file. Each mapping converts an html Meta Tag value into a field in the index. This field is indexed as a text, using the standard analyzer, included in the "All" search and hit-highlighted. It will also be possible to sort by the field.

The mapping file is located in the index-server plug-in directory:

[apppath]\Forward.DataProcess.PlugIns\

and is called:

Forward.DataProcess.PlugIns.WebPage.CustomMetadataMapping.xml

These 15 custom fields will be discontinued, but they can be maintained in the index using the extended custom fields feature, and thus, search-code relying on the index field names “CUSA” to “CUSO” can be made to work also after migrating to the next major version.

Geo Search fields

When utilizing the Geo Search capabilities of Forward Search, extra fields are added to the index:

There are as many geo-net fields (GNxx) as there are defined geo nets in the geo-setup file.

Field	**Source**	**Description**	**All?**
GLAT	geo.position or <span id=”latitude”> or <span id=”geo”>	The latitude of the document	
GLON	geo.position or <span id=”longitude”> or <span id=”geo”>	The longitude of the document	
GLOC	geolocality	An associated locality name or closeby locality	*yes*
GUNC	geouncertainty	An indeger in meters that indicates the uncertainty of the location.	
GNxx	Auto	Calculated field with the id of the box for net xx.	

The Geo Search capabilities of Forward Search require a separate license. Please refer to separate documents on how to configure and use the geo-search feature.

Run Forward Search Crawler

There are 3 different ways to activate the crawler and indexer:

- Manual trigger (run [approot]\Forward.IndexServer.Trigger.exe)
- Schedule
- Event based trigger

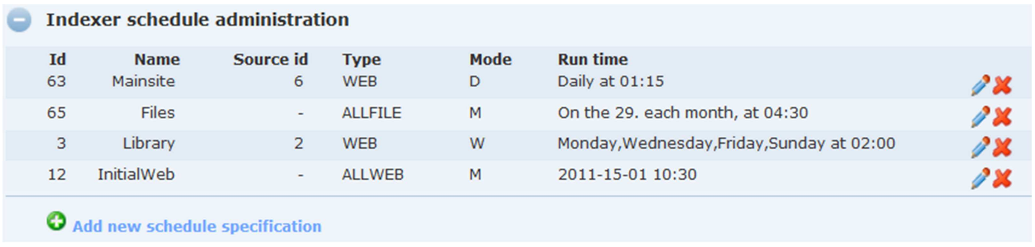
Manual trigger

You can start the indexer manually from the administration client, or you can run the trigger executable: [approot]\Forward.IndexServer\Forward.IndexServer.Trigger.exe

Schedule

You can set up the index server to run on fixed times by using the scheduler build into the index server. (You can also use the windows scheduler if that turns out to be more convenient)

Notice: The schedule file can be edited from the GUI interface the administration client as seen below. This is the recommended method to edit scheduled tasks. Please refer to the administration client documentation for further on scheduled task editing.



Indexer schedule administration

Id	Name	Source id	Type	Mode	Run time
63	Mainsite	6	WEB	D	Daily at 01:15
65	Files	-	ALLFILE	M	On the 29. each month, at 04:30
3	Library	2	WEB	W	Monday,Wednesday,Friday,Sunday at 02:00
12	InitialWeb	-	ALLWEB	M	2011-15-01 10:30

The schedules list in the admin client. There are 4 schedules, 2 for specific sources, 2 for all file/web sources. The top 3 are repeated, the last one only runs once, on the date and time stated.

To manually edit the sch.xml in [approot]\Forward.IndexServer\data, open the file in a simple text editor or xml editor. All scheduled tasks are elements with the tag name “task”. The elements of a single schedule entry – a “task” are:

Id:	unique id
Datetime:	scheduled from first date
Name:	name of task
Modetype:	
	D=dayly
	W=weekly
	m=monthly
Repeatly	only once or reccurring
Daylist	weekday if weekly
Crawltype	which section: WEB,FILE,SPS (WEB = website - FILE = Fileserver - SPS = SharePoint)
Crawlsecid	id from websetting

Below is a sample schedule “task” entry for reference:

```
<task>
    <id>633422088379687500</id>
    <datetime>27-03-2008 12:00</datetime>
    <name>printer</name>
    <modetype>D</modetype>
    <repeatly>True</repeatly>
    <daylist>Thursday;</daylist>
    <crawltype>WEB</crawltype>
    <crawlsecid>1388993805</crawlsecid>
</task>
```

Event trigger

It is often relevant to have the index updated with new or changed content as soon as these changes are published and thus available on the website. This can be accomplished by calling the Event Trigger WebService, included in the Forward Search installation. Then you need to hook the call to this WebService method to the event in the CMS.

When the event is fired, the changed urls are sent to the Forward Search WebService EventService.asmx – the method “EventUpdate”.

The index-server is also triggered, but delays its actions 30 seconds to allow for more urls to be posted. Then it starts a re-indexing of the received urls.

To make sure the index-server is triggered, provide the windows hostname in the entry “ForwardIndexServer_HostName” in web.config of the WebService.

The index-server also needs to be able to retrieve the urls. Therefore, make sure the EventService.asmx url is specified in the index server configuration file, as described in the chapter “Configuration of website source“ earlier in this document.

There are samples with full documentation for use of the event trigger for selected CMS available on the Forward Search Partner Portal.

Search: Use of the Forward Search WebServices

To perform a search into a Forward Search index, you must construct a query and then call a WebService with the query, the name of the index and the fields you wish to have returned in the result set. You may further add pagination information and other parameters to the call, known as “engine commands”.

User-control, XSLT or Ajax

In content management systems like Sitecore and Umbraco XSLT renderings is common. In those cases, you can call the WebService through the use of the XSLT extension delivered with Forward Search. The result is returned as an XML document ready for iterating in the XSLT.

For more complex queries, for queries in In EPiServer and other system, your query and the subsequent result-rendering can be handled by a usercontrol or webpart, utilizing the Forward Search API. The result is returned as a dataset with two or more tables containing the result data, ready for data binding to repeaters and similar.

Finally, to request a search query and process the results client side via Ajax, you can call the handler version of the search interface, and retrieve the results as a Json object ready for JavaScript based iteration and rendering.

Search by User-control

When you decide to create the search query and render the results in a user control or similar, you will need to include the forward search assembly “Forward.Search” in the dll “Forward.Search.dll”. This namespace contains several classes that make the construction of even complex search queries easy. You will also need to make a web-reference to the ForwardSearchWebservice.asmx, a part of the installed Forward Search WebServices.

The Query class allows you to construct queries and sub-queries, and the Term and GeoTerm classes allow you to add search terms and specialized geographical terms to the query.

Once the query is constructed, you can call the WebService method “Search” and pass along the search query by converting it into xml, using its method “ToXml”. For simple text searches, you can skip the query construction and simply call the WebService “Search” method with a text string.

The Search method has these parameters:

`Search( String index,`	The name of the index to query
`string user,`	An optional user (for secure documents)
`string textquery,`	The simple version of the query- a text string
`XmlNode xmlquery,`	The query as an XmlNode
`string format,`	Optional formatting instructions
`int startIndex,`	Pagination: The first document to return from the result
`int endIndex,`	Pagination: The last document
`string displayParagraphs,`	The fields to return, as a comma-separated list [a)]
`string sortParagraphs,`	The field or fields to sort the result by [b)]
`bool ascending,`	Sort ascending?
`string engineCommands )`	Optionally a number of extra commands for the search

a) Fieldnames may be post fixed with “@A” where A refers to a section in the reduct text configuration descripted in the WebService configuration.
b) “WEIGHT” is a special fieldname, telling the search engine to sort results based on relevancy – other fields can be combined but not WEIGHT. There is neither anything as least relevant.

The other search interfaces

The XSLTextension contain among other methods, a “SearchXMLQueryAllArgs” method which gives a very accurate equivalent to the WebService method described above.

It is possible to generate a query in pure xml, see example in “A sample search query” appendix

The search handler “ForwardSearchAsJson.ashx” has the same query-parameter options as the WebService (though not exactly the same names they are shorter), and is therefore also its equivalent.

For more on these interfaces and the other relevant interfaces available in Forward Search, please refer to the other available documents pertaining to the specific needs you have.

Engine commands

As part of the search query, you can pass a semicolon-separated list of engine commands in the “engine commands” field of the search query (all versions of it). Engine commands affects the way the search engine handles the search query and the results. The actual query is not affected by engine commands. Here are some relevant engine commands you can opt to include:

Command	***Effect***
`USEHITHIGHLIGHT` `UHHL`	Fields returned are marked with hit-highlighting, meaning that matching search words are surrounded by a span tag: `<span class="fsHit"> .. </span>` Notice that you can use either the long or shorthand version.
`NOPOP`	The search words are not collected to the popular words system. Use this for all search queries that are not free-text searches entered by visitors.
`NOLOG`	The search query is not collected to the search history system. Use this for all system-generated queries that are not directly related to user input
`#KILL#`	Reset the WebServices, clear caches and reload all configuration files. No search is performed when this command is detected.
`facetlist`	Look a section below about facet counting
`facetlistthreshold`	Look a section below about facet counting

Notice, that there are more engine commands available, some of which are described under the relevant topics.

Building queries

There are basically two ways to construct and submit a search query. The resulting search will be performed the same way disregarding; only the construction, marshalling and parsing of the query is different. The two ways to construct queries are:

1) Constructing a query as a “human text” string. For example:
“`(cauldron OR bubble) AND AUTHOR:Shakespeare`”.

When using this approach, the human text string is parsed into queries and terms by the search query parser.

2) Making a query by adding terms and sub queries to query (either in xml or as .NET user-controls instantiating Search classes). When using this approach, the query parser simply constructs the correct queries and terms directly from the serialized xml - possibly generated from the instantiated .NET classes.

Below is a short introduction to the query language and how to implement it as query string and using the Search classes. After the query language introduction is a short introduction with samples on how to create queries using the Search classes.

The Query language

The query language is constructed of a query made up of one or more a sequence terms.

The Term

One term contains a string of one or more words to search for and an index field to search in – for instance: Find "`Shakespeare`" in field "AUTHOR".

Several words in a term

The string of words is itself parsed by the query parser. Thus, the term: Find "`cauldron OR bubble`" in field TEXT is automatically changed to two terms: Find "`cauldrom`" in field "TEXT" OR'ed with term Find "`bubble`" in field "TEXT". (If the string of words has no logical operator, the AND operator is default used.)

Several terms in a query

Several terms are combined in a query to form a number of parallel terms, that are joined using the logical operator AND, OR or NOT, depending on what is specified. Default is always "AND".

Sub-queries

Several terms can be joined together forming a sub-query that combine with other queries using different boolean operators. For instance the example from above: "`(cauldron OR bubble) AND AUTHOR:Shakespeare`" contains the sub-query (cauldron OR bubble) within the query, that also contains the term `AUTHOR:Shakespeare`. Sub-queries represent sets of parenthesis in the human-text query. There is no technical limit to the level of nesting sub-queries within other queries, but there is an upper limit in number of terms total for the entire query, which is set at 60.000 individual terms. A warning will be issued to the error-log if this limit is passed.

Phrase terms

Searching for an entire phrase in a field is possible using a phrase query. In the human text form, it is indicated by using quotes around the term string: `TEXT:"fire burn and cauldron bubble"`.

"All" search terms

If you don't specify a field to search in, the system automatically assumes "ALL" text fields, which means that the search is done in all those fields that are natural text fields. The table in the section "The fields of the index" above indicates which fields are "ALL" fields.

Wildcard terms

You can use wildcards when specifying the search word. You can substitute single letters (with a question mark) or sequences of letters (using an asterisk) So an example as "`AUTHOR:Shake*`" will find hits in the "AUTHOR" field for "`shakespeare`", as will "`AUTHOR:Shake?peare`".

Range terms

You can search for ranges in special numeric or prepared text fields. By providing a start and end number (double), the term will match all records that fall between these two values, themselves included.

Geo terms

Using the special class Geoterm, you can search for records that fall within a certain distance from a center. You provide a longitude and latitude specification of the center and a maximum distance OR a minimum number of hits. The search engine will match all records within distance OR find the closest records, up to the minimum specified, if available. When using a Geoterm, it must always be added to the outer-most query, and it cannot be nested, nor can you use more than one Geoterm in a query. Geo terms require that the index is prepared for geographical searches. Please reference the separate document on geo-searches with Forward Search. You cannot make geo-searches using human text queries or XML queries, but must use the Search classes in .NET code.

Further options

It is also possible to construct terms with fuzzy logic (the query text doesn't have to match exactly what is found in the index) or with a proximity clause where several words must fall within a certain distance (in characters) from each other in the stored text for the field, for the record to be a match. Read more on this in a separate document.

The Search classes

Getting started with the Search classes in .NET is easy. Just include the Forward.Search.dll in your project and reference the Forward.Search namespace to gain access to these two central classes:

Class: Query

Instances of this class accept the addition of terms and sub-queries, to construct complex queries, and it is responsible for serializing the query object and its terms and sub-queries into XML for the search call to the WebService.

Important methods and properties:

`AddFieldQuery`	Add a term, a geo-term or a sub-query to the query. (4 overloads)
`OperatorBetweenTerms`	Set the Boolean operator to operate on the terms added to this query. AND, OR or NOT.
`ToXml`	Use this to convert the query to an xml document for adding to the Search WebService.

Class: Term
Instances of this class contain query values pertaining to a single field and using a common Boolean operator. Terms must always be added to queries before submitted to the search method of the WebService.

Important methods and properties:

`FieldName`	The field to target. Or “ALL” (or blank for “ALL”)
`FieldValue`	The search string.
`FieldOperator`	If the search string consists of several words, this operator is applied: OR or AND
`FieldType`	Text (default) or Range or DateTime.
`StartRange / EndRange`	If FieldType is Range, add the start and end for the range search, as doubles.
`StartDateTime / EndDateTime`	If FieldType is DateTime, add the start and end for the range search, as date-time instances.
`FactorBoost`	Boost the weight of this field in the calculation of the total score of the result. The value is a percentage of “normal” boost, and thus 100 = no specific boost. Minimum is 1 for a 1/100th boost and maximum is 1000 for a factor 10 boost.
`FactorFuzzy`	Allow some fuzziness in the search (misspelling, different endings etc.) The factor is a digit between 0 and 9, where 9 denote an almost perfect match and 0 indicates a very loose match. Default factor is 5.
`FactorProximity`	Require the words in search string to be in close proximity: A positive integer that indicates the allowed number of words between the search words in the term, for it to be considered a match. If only a single word is in the term search string, this setting is ignored.

Class: GeoTerm
Instances of this class contain query values used for geo-searching. It relates to the Geo Search specific calculated fields of the index, if such exists. Therefore, no field needs to be set manually.

Important methods and properties:

`Latitude`	The latitude as a string, of the center
`Longitude`	The longitude as a string, of the center
`DistanceUnit`	The distance unit entered. Options are Feet, Meter, Kilometer, Mile, NauticMile and Yard
`MaxDistance`	The maximum distance as an integer, in the specified unit. Only records within the circle defined by center and with this distance as maximum are considered hits. (ignored if MinHits > 0)
`MinHits`	The minimum number of hits to look for. If > 0 then the search engine will look for hits in ever increasing distances from the center until MinHits is found or the index is exhausted.

In the appendix you will find a C# snippet of a construction of a query and the subsequent call of a WebService with that query.

Facet counting

If the index to query has been prepared with "facets" – various properties on the documents turned into custom fields for use as dimensions or refinement options on the search result, you can request the Forward Search engine to count and return the selected facet fields as a part of the search query. The facet fields are specified in an engine command like this example, where some product properties are used as facets:

```
"facetlist:BRAND,COLOR,TYPE,OPTIONS;facetlistthreshold:5000;"
```

Facets can be single-value fields (like "Brand", which, for each document will only contain a single value) or multi-valued fields (like "Options" that may contain a comma separated list of various options for the product the document is about).

Read more about setting up and querying facets in the separate document "Facet Counted Search – Quick Guide", available on http://portal.forwardsearch.dk

Did-You-Mean

Forward Search provides another WebService method by which you can request the search engine to return from the index terms that more or less matches the one you propose in the call. This feature is commonly known as the "Did you mean" feature, and can be used for spelling-correction and suggestions for the user to improve the search experience. The method returns a data table with the closest relevant suggestions based on your input. Two similar methods exist for this feature – one has a few extra input parameters – that's the one documented below.

The Did-you-mean method has these parameters:

`DidYouMeanEx( string user,`	An optional user (for secure documents)
`string password,`	An optional password (for secure documents)
`string database,`	The name of the index to query
`string word,`	The word to look for similar terms for
`int suggestions,`	How many suggestions to return (max)
`int minimumdist,`	The minimum "distance" between the word and the suggestions
`int minimumfreq )`	The minimum number of occurrences for a term to be suggested.

The "minimumdist" parameter allows you to specify how different the suggestion must be to be returned as a suggestion. The default is "0" meaning that the word itself, if it exists, is also returned (the distance to itself is "0") A distance of 1 approximately means that a single letter needs to be replaced to get to the suggestion, and so forth.

Notice, that the maximum distance is "8", so that words quite different from the provided word are never returned, and the search engine has a natural "stop" condition for looking further. But short words might never reach 8 in difference, so be sure to consider this when using the results for short words, particularly in small indexes where the risk of alien results is higher, since there will be fewer terms with a close match. The "minimumfreq" parameter allows you to filter the returned suggestions by frequency – a suggestion that only exists one or two times in the entire index, may very well be a misspelling, and this way, you can force a minimum frequency. The default is 4.

Type-ahead

To support “type-ahead” or “auto-completion” functionality of search fields, Forward Search provides a handler function that allows the query of filtered field values from the term lists of the index, and returns the matching hits with frequency counts as a simple Json string, ready for use in various auto-completion JavaScript components. A free-to-use auto-completion component is distributed with the administration client and with our sample site, including source code that demonstrates the usage. Please also refer to the separate document “Type-ahead Quick guide” on more about how to use this interface for auto-completion of search fields.

Tuning

Using a Thesaurus

For free text searches, in order to improve the chance of relevant result hits, you can expand the search query to contain other words that are related to the entered search word. This is done by looking up the entered search words in a thesaurus and adding any found similar terms to the query. If the visitor searches for "Car" we can make sure the engine also searches for results with "Automobile".

You can opt to use a premade multi-lingual thesaurus provided with Forward Search if you desire so. It contains standard word-aliases for 25 languages. A separate license is required for using this option.

Custom Thesaurus

You can also construct a custom thesaurus with words that are tailor-made for the specific application. It could be product-aliases, keywords from recent ads and campaigns, the inclusion of obsolete product-numbers for customers with old catalogues and a number of other practical scenarios. One typical example is to add the correctly spelled version of often-misspelled words. Using the query history feature of Forward Search may suggest relevant candidates for such spell-fixing thesaurus entries.

The custom thesaurus is an xml document located with the index file, the exact location specified in the index setup. The file is pr. default called "cusths.xml" see the section "Forward Search WebService configuration" in the chapter "Installation and configuration" earlier in this document.

The custom thesaurus file consists of a list of words grouped together according to similarity of meaning, synonyms, related words etc.

Notice: You can edit the custom thesaurus through a GUI interface in the administration client, or you can work directly in the xml file. Any changes to this setup is first active after the WebService has been reinitialized (kill application pool or use the admin feature to reload configuration)

Examples

Type	***Word (parent)***	***Alternatives (child)***
Synonyms	Automobile	Car
	Bike	Bicycle
Categories vs. specifics	Toy	Lego
		Doll
	Car	Ford
		Volvo
Singular vs. plural	Video	Videos
	Videos	Video
Alternative writing	AK47	AK-47
	AK/47	AK-47
	AK 47	AK-47

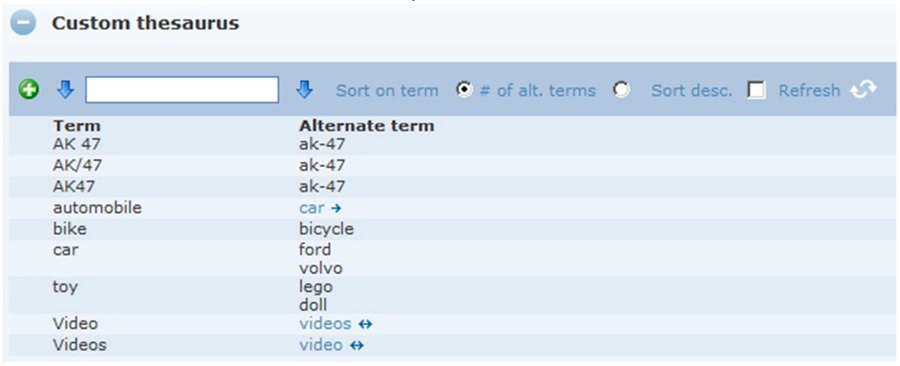
Notice, that for the count, we want to have both singular pointing to plural and plural pointing to singular. This could also be the case for synonyms and categories, but less likely for alternative writing of a product id.

The structure of the thesaurus is as shown here – for the examples above, the "Car" word. (Full sample is in appendix)

```
<Parent>
  <id>5</id>
  <word>car</word>
</Parent>
<Child>
  <id>5</id>
  <word>ford</word>
</Child>
<Child>
  <id>5</id>
  <word>volvo</word>
</Child>
```

Each “word” or “term” is a “parent” node with a unique id, and each alternative is a “child” node pointing to that id. The order of entries in the custom thesaurus is based on the order of creation, and all parents are listed first. See the appendix for the full example custom thesaurus.

In the administration client editor, the same thesaurus would look like this:



Term	Alternate term
AK 47	ak-47
AK/47	ak-47
AK47	ak-47
automobile	car →
bike	bicycle
car	ford volvo
toy	lego doll
Video	videos ↔
Videos	video ↔

In the administration client, the custom thesaurus terms and alternates are grouped together. Color and arrows indicates if an alternate is itself also a term in the thesaurus, and a double-arrow indicates that it, as a term, relates to the term it itself relates to as an alternate. The list is alphabetically sorted by term, or by count of alternates.

Stop words

Some words can be considered as noise, especially when results are returned based on relevance ranking. To avoid these words from being indexed and being searched for, you can create and maintain a “stop-word” list with the words to ignore. Stop words are persisted in normal text files, one stop word pr. line, in the relevant language.

Forward Search uses the stop word list specified in the WebService configuration file, as mentioned in the section “Forward Search WebService configuration” on page 8 above. Alternately, the system looks for a common stop word list called "stopword_common.lst" in index parent dir. If several languages are present as sources for a single index, it is recommended to edit the common stop word list, including all stop-words for all relevant languages.

If no stop word list is provided, an internal list is used with these English stop-words:

```
I, a, about, an, are, as, at, be, by, com, de, en, for, from, how, in, is, it, la,
of, on, or, that, the, this, to, was, what, when, where, who, will, with, the, www
```

To turn off using stop words, including the internal list, create a new, empty file called "stopword_common.lst" and place it in the parent directory to the index.

Maintenance

Re-index

The indexes are normally maintained and updated through the setup of scheduled tasks and through updates occurring as a result of event updates in the CMS. If you need to re-index an index you can do it in one of several ways:

- Using the Administration Client. For the relevant index, click the “Recycle” button and confirm the re-index in the subsequent dialog-box.
- Manual file-deletion. Delete the files in the dump, storage and index folders of the relevant source and index. Then run forward.indexserver.trigger.exe
- Automatic file-deletionService trigger. Run forward.indexserver.trigger. Exe with ”reindex” console argument.
- Direct call to Service. Telnet to forward.indexserver service on port 12051 write command ”CLEANALL” and ”CRAWL ALL”

Logs

The index server and the WebService write various status and error messages to several logs. Below is a very brief description of these logs, to assist you in finding the relevant information when monitoring and trouble-shooting the system.

The “I’m alive” logging

The Index-service writes ”I’m alive” to the Windows “Application Event log” once every hour. If the index-service is not running properly, this entry should be missing. Use this to establish an approximate “time of death” if the index-server has stopped working.
For a few severe errors, the Index-service writes the error to the Windows “Application Event log”. These log-entries may therefore contain extra information pertaining to the failure.

The crawler logs

The index-service crawler process writes a status-log to [apppath]\data\logging\[yyyymmdd].il files. These files contain information about the crawling and indexing process, hereunder warnings and error messages and how long time indexing tasks and steps take to complete. The log is, as the indexing-process, divided into two major tasks; crawling and data processing. The log reflects this using these 3 abbreviations:

IS	Index server. Typically start and stop messages.
CW	Crawler process. The crawler crawls the content and gathers all the documents to be indexed. The crawler therefore also finds broken links.
DP	Data processor. The data processor performs the actual indexing. It also performs a number of other pre- and post-processing of the found documents.

Looking at these log files, it gives a picture of how the website performs, we would except between 3 and 15 entries per second. The “il” logs can be viewed using the administration client, and here filtered for various types of entries. A number of specialized reports on these log files can also be applied, and 3 user-definable custom reports are available.

Entries for each visited page is logged with both “WOP:” and “ResponseTime for:”, where WOP is pre page fetch and ResponseTime is post. Response time entries are aggregated into the admin client to collect info about average response time, slowest pages (blacksheeps).
Be careful about assumptions about response time, because a normal visitor will never visit all pages, and might have a better cache hit ratios.

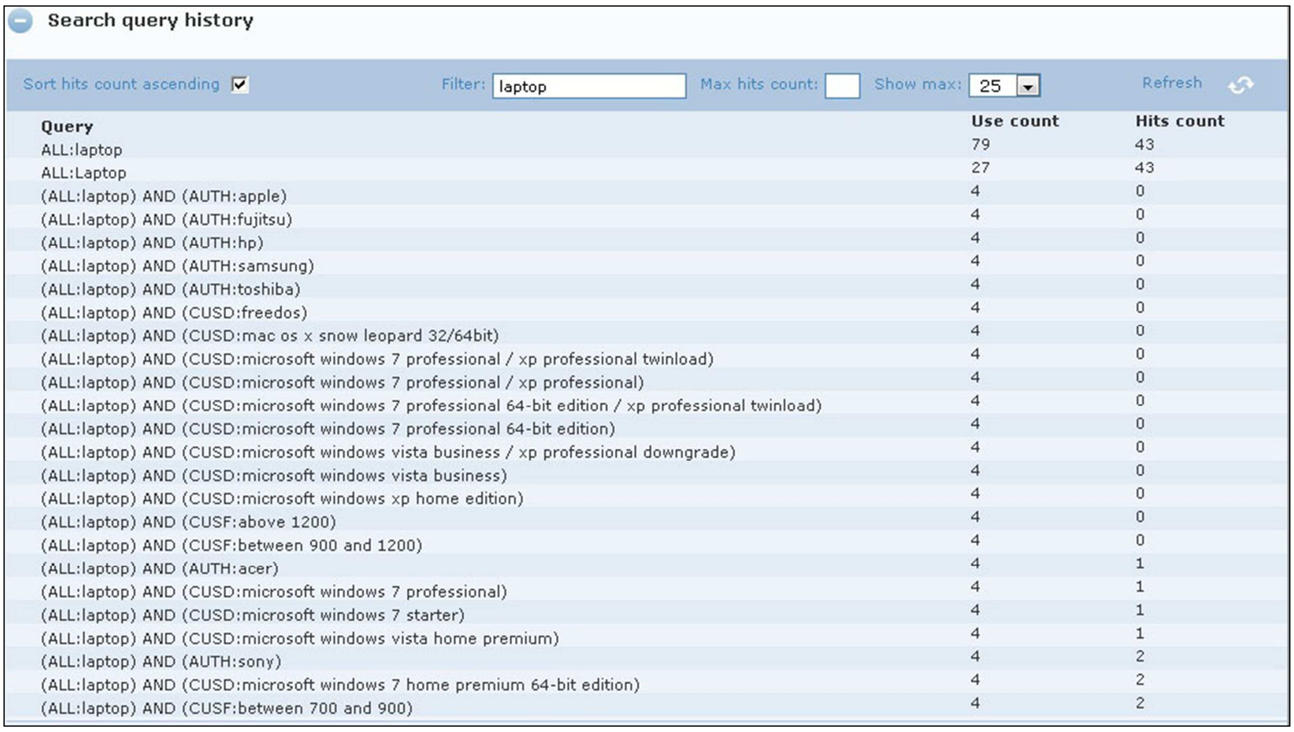
From version 2.7 Debug mode - Setting in config.xml is ignored; it is changed into 2 log4net loggers:

Normal logger: forwardsearchmainlogger
Verbose logger: forwardsearchmainloggerlevel2

The Query log

This log captures all queries made against the indexes, unless “NOLOG” was specified in the search call engine command field. The log files are date-stamped and located here: [apppath]\WebSite\ForwardSearchWebService\Data\logging\[yyyymmdd].qlf

The administration client allows viewing and filtering the query log.



Query	Use count	Hits count
ALL:laptop	79	43
ALL:Laptop	27	43
(ALL:laptop) AND (AUTH:apple)	4	0
(ALL:laptop) AND (AUTH:fujitsu)	4	0
(ALL:laptop) AND (AUTH:hp)	4	0
(ALL:laptop) AND (AUTH:samsung)	4	0
(ALL:laptop) AND (AUTH:toshiba)	4	0
(ALL:laptop) AND (CUSD:freedos)	4	0
(ALL:laptop) AND (CUSD:mac os x snow leopard 32/64bit)	4	0
(ALL:laptop) AND (CUSD:microsoft windows 7 professional / xp professional twinload)	4	0
(ALL:laptop) AND (CUSD:microsoft windows 7 professional / xp professional)	4	0
(ALL:laptop) AND (CUSD:microsoft windows 7 professional 64-bit edition / xp professional twinload)	4	0
(ALL:laptop) AND (CUSD:microsoft windows 7 professional 64-bit edition)	4	0
(ALL:laptop) AND (CUSD:microsoft windows vista business / xp professional downgrade)	4	0
(ALL:laptop) AND (CUSD:microsoft windows vista business)	4	0
(ALL:laptop) AND (CUSD:microsoft windows xp home edition)	4	0
(ALL:laptop) AND (CUSF:above 1200)	4	0
(ALL:laptop) AND (CUSF:between 900 and 1200)	4	0
(ALL:laptop) AND (AUTH:acer)	4	1
(ALL:laptop) AND (CUSD:microsoft windows 7 professional)	4	1
(ALL:laptop) AND (CUSD:microsoft windows 7 starter)	4	1
(ALL:laptop) AND (CUSD:microsoft windows vista home premium)	4	1
(ALL:laptop) AND (AUTH:sony)	4	2
(ALL:laptop) AND (CUSD:microsoft windows 7 home premium 64-bit edition)	4	2
(ALL:laptop) AND (CUSF:between 700 and 900)	4	2

The Search Query history list shown here is filtered by the term “laptop”. There have been 79 queries for that term, which returned 43 hits. Also notice, that the same search but with a capital “L” was performed 27 times. Since the search engine always searches case-insensitive, obviously the two versions of the query return the same number of hits.

The Popular search words log

This log captures the search terms used against the indexes, unless “NOPOP” or “#NOLOG#” was specified in the search call engine command field. The terms included are those queried against text fields using AND or OR operators. Sub queries and terms following “NOT” operators will not be added the popular search word log. The days back popular search words are maintained, and the relative weight each term should have in ranking the popularity, is configurable in the index-server config file.

The popular-search word files are date-stamped and located here:

[apppath]\WebSite\ForwardSearchWebService\Data\logging\[yyyymmdd].plf
The administration client allows viewing and filtering the query log.

The Debug logs

The index-server and the WebServices may be configured to write extensive debug-logs and error logs to a subdirectory under [apppath]\website\data\debuglogs\ and [apppath]\Forward.Indexserver\data\debuglogs\. These logs, that use Log4Net, contain a large number of information that can assist pinpointing problems in the indexing process as well as the search process. This logging is per default disabled, but can be enabled by creating the mentioned above, and setting the error-level to "DEBUG". See the appendix for more on configuring the debug logging.

Appendix

Productivity tools in the installation package

When unzipping the Forward Search package, the main modules are unzipped to three sub-folders of the installation root. Two other folders are also created, containing various additions and productivity tools. These are described in brief below.

PPS

The PPS subfolder contains the binaries and configuration files for the Forward Search Post processing services, used for altering and enhancing the index after re-indexing. For more on the post processor, please refer to a separate document.

Tools

The Tools folder contains three sub directories: the “Finalizer”, the “XSLT Helper” and the “Tester”. They are described below.

- The Finalizer is a separate Visual Studio C# project containing the basic “Document Finalizer” code that can be used to create a modified version of the document finalizer assembly, and through this modify the prepared documents before they are written to the index.

- The XSLT helper is an extension assembly to be used when rendering search results using XSLT transformation. It contains the logic needed to query the Forward Search WebService and to iterate and render the result.

- The Tester is an executable windows application that allows you to test the Forward search WebService and make queries against a selected index. It is therefore useful both for testing the configuration of the WebService and for testing the actual contents of an index.

A sample custom thesaurus

The custom thesaurus seen below matches the one described in the section about the custom thesaurus, with all entries present.

```
<?xml version="1.0" standalone="yes"?>
<Thesaurus>
  <xs:schema id="Thesaurus" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-
microsoft-com:xml-msdata">
    <xs:element name="Thesaurus" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
      <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="Parent">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="id" msdata:AutoIncrement="true" type="xs:int" />
                <xs:element name="word" type="xs:string" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="Child">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="id" type="xs:int" />
                <xs:element name="word" type="xs:string" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
      <xs:unique name="Constraint1">
        <xs:selector xpath=".//Parent" />
        <xs:field xpath="id" />
      </xs:unique>
      <xs:keyref name="ParentToChild" refer="Constraint1">
        <xs:selector xpath=".//Child" />
        <xs:field xpath="id" />
      </xs:keyref>
    </xs:element>
  </xs:schema>
  <Parent>
    <id>0</id>
    <word>bike</word>
  </Parent>
  <Parent>
    <id>1</id>
    <word>ak47</word>
  </Parent>
  <Parent>
    <id>2</id>
    <word>automobile</word>
  </Parent>
  <Parent>
    <id>3</id>
    <word>videos</word>
  </Parent>
  <Parent>
    <id>4</id>
    <word>video</word>
  </Parent>
  <Parent>
    <id>5</id>
    <word>car</word>
  </Parent>
  <Parent>
    <id>6</id>
    <word>ak/47</word>
  </Parent>
  <Parent>
    <id>7</id>
    <word>ak 47</word>
  </Parent>
  <Child>
    <id>0</id>
    <word>bicycle</word>
  </Child>
  <Child>
    <id>1</id>
    <word>ak-47</word>
  </Child>
  <Child>
    <id>2</id>
    <word>car</word>
  </Child>
  <Child>
    <id>3</id>
    <word>video</word>
  </Child>
  <Child>
    <id>4</id>
    <word>videos</word>
  </Child>
  <Child>
    <id>5</id>
    <word>ford</word>
  </Child>
  <Child>
    <id>5</id>
    <word>volvo</word>
  </Child>
  <Child>
    <id>6</id>
    <word>ak-47</word>
  </Child>
  <Child>
    <id>7</id>
    <word>ak-47</word>
  </Child>
</Thesaurus>
```

Using and configuring the debug logger

The Forward Search operates a logger utilizing the renowned Apache Log4net logging system.

At install time, these components are installed as part of the admin client:

log4net.dll	The latest log4net.dll.
log4net.config	The configuration file for the logging system.*

Enable debug logging

At install time, the debug logging is disabled. In the configuration, it is possible to turn on the debug logging, by changing the line <level value=”INFO” /> in the <root> section into <level value=”DEBUG” />, and thus allowing DEBUG level messages to be sent to the debug log file. The <root> section will then look like this:

```
<root>
     <level value="DEBUG" />
     <!-- appender-ref ref="SmtpAppender" -->
     <appender-ref ref="RollingLogFileAppender" />
     <appender-ref ref="RollingErrorFileAppender" />
</root>
```

For more information about how to configure and use log4net used in the Forward Search Admin client, please refer to the Apache Log4Net documentation here:

http://logging.apache.org/log4net/index.html

A sample search query

Below you will find a snipped of C# code, that implements a search call by first constructing a query with geo-search, range and a nested “OR” query, and then calls the WebService with it. All the search parameters are “hardcoded” except the index name, which obviously is not what is normally done.

In the search call, we ask for the first 100 hits, and we want the result sorted by the geographical distance (GDST). For each hit returned, we ask for the title and text, as well as the color, price and distance to be included.

```
private DataSet Search(string index)
{
    Query mainquery = new Query(Operator.AND);

    // Add the geo-term - with a fixed center and distance
    GeoTerm geoterm = new GeoTerm();
    geoterm.Latitude = "55.7904";
    geoterm.Longitude = "6.4370";
    DistanceUnits u = DistanceUnits.Meter;
    geoterm.MaxDistance = 2000;
    mainquery.AddFieldQuery(geoterm);

    // Add a subquery with 3 OR'ed values
    Query subquery = new Query();
    subquery.OperatorBetweenTerms = Operator.OR;
    Term tA = new Term("COLOR", "YELLOW");
    Term tB = new Term("COLOR", "ORANGE");
    Term tC = new Term("COLOR", "BEIGE");
    subquery.AddFieldQuery(tA);
    subquery.AddFieldQuery(tB);
    subquery.AddFieldQuery(tC);
    mainquery.AddFieldQuery(subquery);

    // Add a range query for prices
    Term range = new Term("PRICE", 2000.0, 5000.0);
    range.FieldType = FieldType.Range;
    mainquery.AddFieldQuery(range);

    // Create the webservice proxy and call it. Return the dataset if success.
    ForwardSearchWebService ws = new ForwardSearchWebService();
    try
    {
        DataSet dataset = ws.Search(index, "EveryOne", string.Empty, mainquery.ToXml(), "", 1, 100,
                                     "TITL,TEXT@A,COLOR,GDST,PRICE", "GDST", false, "");
        if (dataset != null && dataset.Tables.Count > 0)
        {
            return dataset;
        }
    }
    catch (Exception e)
    {
        errormessages.Text = "Could not call Search on webservice. Failed with error: " + e.Message;
    }
    return null;
}
```

This will generate an xml query like this:

```
<Root>
  <OperatorBetweenTerms>AND</OperatorBetweenTerms>
  <Terms>
    <GeoTerm Latitude="55.7904" Longitude="6.4370" MaxDistance="2000" MinHits="0"
             DistanceUnit="Meter" FactorBoost="100" NegativeSignLetters="WS" />

    <SubQuery>
      <OperatorBetweenTerms>OR</OperatorBetweenTerms>
      <Terms>
             <Term FieldName="COLOR" FieldOperator="AND" SearchExpand="None"
             FieldValue="YELLOW" FieldType="Text" FactorFuzzy="5" FactorProximity="1"
             FactorBoost="100" />
             <Term FieldName="COLOR" FieldOperator="AND" SearchExpand="None"
             FieldValue="ORANGE" FieldType="Text" FactorFuzzy="5" FactorProximity="1"
             FactorBoost="100" />
             <Term FieldName="COLOR" FieldOperator="AND" SearchExpand="None" FieldValue="BEIGE"
             FieldType="Text" FactorFuzzy="5" FactorProximity="1" FactorBoost="100" />
      </Terms>
    </SubQuery>

    <Term FieldName="PRICE" FieldOperator="AND" StartRange="2000" EndRange="5000"
             SearchExpand="None" FieldType="Range" FactorFuzzy="5" FactorProximity="1"
             FactorBoost="100" />
  </Terms>
</Root>
```

Subpage items

Tagging content with sub item comments make the crawler index each block as separate documents and with a common header. The Url field will refer to the full of the page with the itemid added as bookmark.

```
<html>
            <head>
                        <title>Common header</title>
            </head>

            <body>

                        <!--SubItemBegin {EDE96BE5-6716-4ABA-8D78-8417B37EF527}-->
                        <div>
                                    <a href="#{EDE96BE5-6716-4ABA-8D78-8417B37EF527}" />
                                    Item 1
                        </div>
                        <!--SubItemEnd-->

                        <!--SubItemBegin {EDE96BE5-6716-4ABA-8D78-8417B37EF528}-->
                        <div>
                                    <a href="#{EDE96BE5-6716-4ABA-8D78-8417B37EF528}" />
                                    Item 2
                        </div>
                        <!--SubItemEnd-->

                        <!--SubItemBegin {EDE96BE5-6716-4ABA-8D78-8417B37EF529}-->
                        <div>
                                    <a href="#{EDE96BE5-6716-4ABA-8D78-8417B37EF529}" />
                                    Item 3
                        </div>
                        <!--SubItemEnd-->

                        <!--SubItemBegin {EDE96BE5-6716-4ABA-8D78-8417B37EF530}-->
                        <div>
                                    <a href="#{EDE96BE5-6716-4ABA-8D78-8417B37EF530}" />
                                    Item 4
                        </div>
                        <!--SubItemEnd-->

            </body>
<html>
```

Troubleshooting

Why does it return a license error, when I'm using the WebService?

- Is it at standard license, with limited frontend webserver access?

My search won't work on certain navigation

- Is the selected page set? (start and end cursors for which part of the query to return)

Why do my documents show up with strange titles?

- How is your metadata controlled? Dublin core title tag? Or do the binary documents contain any document properties?

Why won't my crawler run again?

- Look in the log for "|E|Storage table lock: doc" it is a locking mechanism that prevents the crawler from updating the index inconsistent. If the process previous was interrupted (terminated) it might need attention (if the interrupted step was data processing, a reindex might be necessary)

Why does my title get cut off?

- It could be a matter of encoding if the title contains apostrophes "somebody's car" the correct rendering would be "somebody&apos;s car"

How do I combine thesaurus hits with expanded search?
I did search on keywords this way:
Term keywordsMatch = new Term ("ALL", keywords, TermOperator.AND, ExpandSearch.TruncateRight);
When I change the ExpandSearch to "None", the custom thesaurus works.
Is there any possibility to have the benefit of the thesaurus together with the benefit of "truncate right"?

- Sure just add both terms, with and without truncation

How do I control my content?

- Beside simple crawler based indexing.
- You might consider:
    - The Document.Finalizer a plugin for the indexing pipeline right before documents are written into the index.
    - Injection of content into an index is possible through WebService api

Is it safe to delete files from dump?

- Yes, if the indexing process is idle.

Why is my index not updated?

- Is the service running? (write to application event log once every hour)
- Is the schedule set? (sch.xml in "data" folder or windows schedule task)
- Is the storage tables locked? (normally "forwarddata\storage\...*.lck")