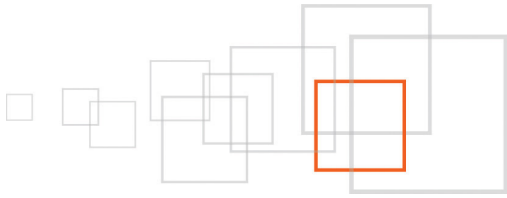
Advanced development with eZ Find

part 3 : Leveraging the Solr syntax

By Gilles Guirand
http://www.gandbox.fr



Index

1 Goal description

Here is the third part of a series of tutorials about eZ Find. At the end of this series, you will have been exposed all details on how it works, and be able to make an advanced usage, in various context, of this enterprise-grade search plug-in. This series will serve as a base for a talk on the subject at the eZ Conference 2010, in Berlin. As well as the Libre Software meeting.

2 Introduction

The previous and second post of this series described how to index additionnal fields in Solr, in order to leverage them using eZ Find's native syntaxe, of the form : **'mycontentclass/mycontentattribute/mycontentsubattribute'**. This eZ Find-specific syntax is very comfortable, but not exclusive. It is indeed possible to mix the eZ Find-specific syntax and the Solr-specific syntax, like for example the field names (**'attr_myfield_type'**) or logic operators (**AND**, **NOT**, etc.) .

" - ***Yes, this is bad practice***. *An "interface" syntax is not made to be worked-around, this potentially endangering the lower layers' evolutivity, namely Solr.*

*- **Yes, this can make development easier** in some cases, or even be a life-saver in some complex situations"*

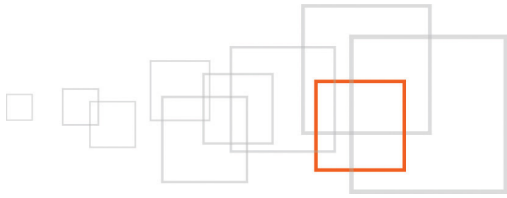
This post dives into concrete examples of how and when one can leverage Solr's syntax. The examples are simplified on purpose, for obvious educational reasons.

3 Pre-requisites and target population

This tutorial requires to know how to set up eZ Find. The online documentation describes the required operation in details, there : http://ez.no/doc/extensions/ez_find/2_2.

You should also read and understand the first and second part of this tutorial :

1. http://share.ez.no/tutorials/ez-publish/advanced-development-with-ez-find-part-1-datatypes-in-solr-and-ez-find
2. http://share.ez.no/tutorials/ez-publish/advanced-development-with-ez-find-part-2-indexing-additional-fields-in-solr



4 Step 1 : How to sort on an attribute present in several content classes

4.1 The issue

It is one of eZ Publish's timeless problematics :

- Two distinct content classes are created, for some reason : " **Post**" and "**Article**"
- Identical attributes are added to both classes, for they are useful in both, like a " **Date**" attribute for instance.

Result :
It is impossible to have both "**Post**" and "**Article**" objects in a fetch result, sorted by decreasing date (unless a terrifying template operator is developed for this specific purpose). Generally, developers try to use one single content class, a more generic one, to work around the issue, or rather relocate it (mutualization of content classes has its drawbacks).

4.2 The solution, using eZ Find

This first post in this series details the naming conventions of Solr fields. One positive side-effect of this convention (related to Solr's dynamicfields concept) is the fortunate absence of the content class identifier in the field name. This means we can leverage this homonymy as we wish, through searches, filters or sorts depending on the use-case.

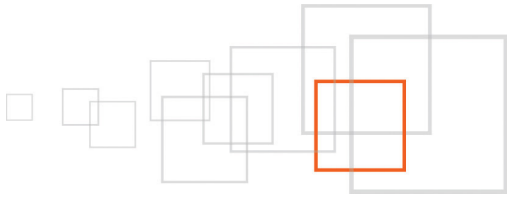
eZ Publish template code example when filtering on the “Post” content class only :

```
{def $search_result = fetch( 'content', 'list', hash( 'parent_node_id', 2,
    'class_filter_type',  'include',
    'class_filter_array', array(24),
    'sort_by', array( array( 'attribute', false(), 'post/date' ) ),
    'limit', 10,
    'depth', 3
))}
```

Equivalent eZ Find template code example, solving our cross-content-class sort, applied to “Post” & “Article” :

```
{def $search=fetch( ezfind, search,
     hash( query , '',
           'class_id', array('post', 'article'),
           'limit', 10,
           'sort_by', hash('attr_date_dt', 'desc')
))}
```

Note :
A desirable evolution of eZ Find would be to give the possibility to use a '**//date**' type of syntax, in order to make optional the currently automatically added content class filter in the query sent to Solr.



5 Step 2 : How to work with keywords

Unlike the previous example of dates, keywords in eZ Publish are stored in an external table **ezkeyword_attribute_link** (additional storage location, on top of the standard content storage location, for an extended logic), allowing to link a given keyword to various pieces of content, of various content classes. However, the per-keyword fetch is not as equipped as a standard content/list fetch for instance, in terms of available filters (**class_filter_type, class_filter_array, extended_attribute_filter**, etc.). This limitation is understandable since allowing for a cross-content-class filter reduces freedom when it comes to filtering on class-specific attributes.

Following the same idea as for the per-date sorting, it is possible to leverage eZ Find to realize all necessary operations around keywords. Here are examples :

```
'filter', array('attr_tags_lk:"ez publish"', 'NOT attr_title_t:"RSS"')
```

Result :
Only returns the results associated with the "**eZ Publish**" or "ez publish**" keywords** (mind the usage of **_lk**, meaning lowercase), and the title of which do not contain "**RSS**".

```
'filter', array('attr_tags_lk:"ez publish"', 'attr_tags_lk:"mootools"')
```

Result :
Only returns the results associated with both the "**eZ Publish**" and "**ez publish**" keywords, and the "**Mootools**" and "**mootools**" keywords.

6 Step 3 : How to create complex search filters

Here are a few illustrations of what it is possible to achieve using the vast set of Lucene operators. The set of available operators depends on the deployed Solr version (Solr 1.4, shipped with eZ Find 2.2 at the time of writing this post).

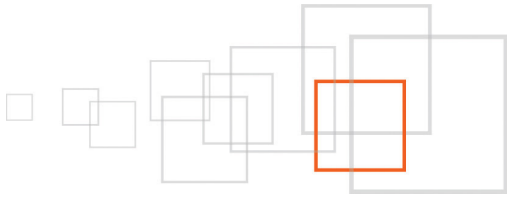
```
'filter', array('NOT ( attr_title_t:(ez+find) OR attr_intro_t:(ez+find) )')
```

Result :
Only returns results which contain the '**ez find**' or '**eZ Find**' expression in the '**title**' or '**Intro**' attributes. Note the usage of the '**text**' (**_t**) of the '**title**' attribute, bringing case-insensitivity, unlike the '**string**' type.

```
'filter', array('attr_title_s:[A TO G] AND ezf_df_text:google~0.7')
```

Result :
Only returns results of which the '**title**' starts by A,B,C,D, E or F (G excluded), and the content of which approximately contains the '**google**' expression (means it may also contain : Google, iGoogle, etc.).

- **Note** : the '0.7' ratio can be adjusted to better suit a given situation
- **Note bis** : the **'ezf_df_text'** field is built dynamically, by copying the content of all of the document's **'string'**, **'text'** ou **'keyword'** fields. One could also use the **'ezf_sp_words'** field if the spelcheck feature is enabled. See the schema.xml file, and the definition of these “copyField” fields for more details.



7 Conclusion

This last post presents how eZ Find helps working around and/or extending legacy eZ Publish fetches, by for instance using a cross-content-class query, or by relying on Apache Solr's native filters (Lucene syntax).

eZ Find constitutes one of the major breakthroughs of eZ Publish, proposing a first step towards the next CMS generations, namely :

- An advanced indexing and querying system. The current integration level of Solr in eZ Find is close to exhaustive, placing eZ Publish a step ahead its Open Source concurrents
- A dynamic storage system : currently handled through am obsolete SQL / Filesystem layer, should evolve towards a dynamic storage system as MongoDB or CouchDB. It probably is eZ Systems' next challenge
- Both an exhaustive and well-performing API and Framework : a key project for eZ Publish, which would deserve a better performing template engine, and most important, a thorough low-level workflow layer, proposing hooks in many, key places in all available operations : what is the future of Zeta Components in this regard ? Should a wide-spread framework be used instead (Zend) ?

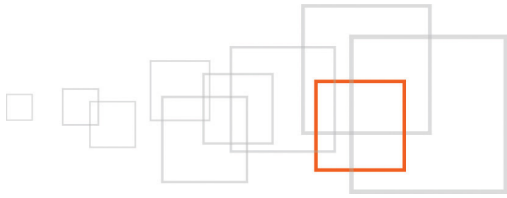
We can also raise the the subject of convergence between professional CMSes and DMSes (Document Management Systems, like Alfresco). Both universes tend to come closer functionally, when it comes to achieving the three points mentioned right above (indexing, storage, API).

As many questions and challenges eZ Systems will have to address within the forthcoming months or years, relying on a major asset : eZ Find is already functioning, widely used, extensively field-tested, extensible and highly competitive when it comes to complex and professional deployments.

I would like to thank Nicolas Pastorino for translating this tutorial to english, and Paul Borgermans for his availability.

8 Resources

- eZ Find 2.2 official documentation : http://ez.no/doc/extensions/ez_find/2_2
- eZ Find source code : here
- Apache Solr Wiki : http://wiki.apache.org/solr/
- Tutorial on eZPedia.org : how to create a template operator : http://ezpedia.org/ez/template_operators
- Keyword fetching documentation : http://ez.no/doc/ez_publish/technical_manual/4_x/reference/modules/content/fetch_functions/keyword
- Lucene query parser syntax : http://lucene.apache.org/java/2_9_1/queryparsersyntax.html
- eZ Find's Solr schema definition file : here



- Solr Copy Fields documentation : http://wiki.apache.org/solr/SchemaXml#Copy_Fields

9 About the author : Gilles Guirand

Gilles Guirand is a certified eZ Publish Developer. He is widely acknowledged by the community to be one of the national experts on highly technical and complex eZ Publish issues. With over 12 years experience in designing complex web architectures, he has been the driving force behind some of the most ambitious eZ Publish Projects: Web Site Generators, HighAvailability, Widgets, SOA, eZ Find, SSO, Web Accessibility and IT systems Integrations.

10 License

This work is licensed under the Creative Commons – Share Alike license (http://creativecommons.org/licenses/by-sa/3.0).