

Apache Solr for TYPO3 - Fluid Templating

version 0.0

Timo Schmidt, Markus Friedrich, Frans Saris and Daniel Siepmann

2016-06-04 22:36

Contents

Introduction	1
Thanks	1
Before you start	1
How to get it	1
What does it do?	1
Basic Setup of EXT:solrfluid	1
Install solrfluid	1
Include TypoScript Setup	1
Use plugin instances from EXT:solrfluid	2
Check the Frontend	3
Use custom Fluid Templates	4
Frontend	5
Fluid Template Structure	5
Result List	6
Facets	6
Facet Types	6
Option	7
Query Group	7
Hierarchical	8
Date Range	10
Numeric Range	11
Rendering with fluid	12
Facet Grouping	12
Default Partial	13
Autosuggest	13
Sorting	14
Results per Page	15
Ajaxified Results	16
How it works?	16
Backend	17
Plugins	17
Results Plugin	17
Flexform Configuration	17
Development	21
Development Environment	21
Testing and Continuous Integration	21
Unit Tests	21
Integration Tests	21
Bootstrapping the Test Environment	21
Running the ci Suite	22

Development Workflow	22
Code Structure	23
Domain Layer & Domain Model	23
ResultSet	23
ViewHelpers	24

Introduction

Welcome to the manual of EXT:solrfluid. In this document we want to document the features of solrfluid and help to configure, use and adapt it to your needs.

Thanks

Thanks to all partners and contributors who support the development around Apache Solr & TYPO3.

Before you start

Make sure your solr extension is configured to index everything you need

- EXT:solr is installed
- TypoScript template is included and solr endpoint is configured
- TYPO3 domain record exists
- Solr sites are initialized through "Initialize Solr connections"
- Solr checks in the reports module are green

If you run into any issues with setting up the base EXT:solr extension, please consult the [documentation](#). Also please don't hesitate to ask for help on the [TYPO3 Solr Slack channel](#)

How to get it

EXT:solrfluid is available for dkd partners only. If you want to get it go to <http://www.typo3-solr.com> or call dkd +49 (0)69 - 247 52 18-0

What does it do?

The solrfluid addon allows you to use the well known template engine fluid, together with EXT:solr. To achieve this, solrfluid ships the needed domain model classes that can be used during the rendering to access the data

Basic Setup of EXT:solrfluid

EXT:solrfluid is an addon for EXT:solr and requires that EXT:solr is installed and configured. Starting with version 6.0 the fluid rendering in EXT:solrfluid will be moved to EXT:solr and the old templating will be dropped.

To allow a smooth migration it is possible to use EXT:solr 5.0 and EXT:solrfluid 1.0 side by side and use the old templating and the new templating side by side, just be using a different plugin instance.

Technically EXT:solrfluid ships an extbase controller and some domain classes and view helpers to implement the new rendering.

Install solrfluid

You can import our shipped version of EXT:solrfluid and install it with the TYPO3 extension manager.

Include TypoScript Setup

Now you need to include the TypoScript template **"Search - Fluid rendering (include after Default Configuration) (solrfluid)"**, right after the normal EXT:solr TypoScript setup:

General

Appearance


Access

Extended

Categories

Content Element

Type

 Insert Plugin ▼

Column

Normal ▼

Header

Header

Search with Fluid


Subheader

Type

H1 ▼

Link


Selected Plugin

 Search: Form, Result, Additional Components (SolrFluid) ▼

Check the Frontend

When everything is configured correctly you can open the page in the frontend and do a search. The example below shows a search for "cms" with an indexed TYPO3 introduction package:

Use custom Fluid Templates

 **TYPO3 CMS**
INTRODUCTION PACKAGE

SEARCH

SEARCH WITH FLUID

GET STARTED

FEATURES

CUSTOMIZING

CONTENT EXAMPLES

ABOUT

RESOURCES

Sort by

Relevance

Title

Type

Author

Creation Date

Narrow Search

Content Type

+ pages (6)

Last searches

• cms

• *

Frequent searches

•

Parsed Query:
(+DisjunctionMaxQuery((tagsInline:cms | content:cms^40.0 | tagsH1:cms^5.0 | title:cms^5.0 | keywords:cms^2.0 | tagsH4H5H6:cms^2.0 | tagsH2H3:cms^3.0)) (/no_coord

Searched for "cms". Found 6 results in 4 milliseconds. Displaying results 1 to 6 of 6.
Results per page:

Congratulations
Relevance: 100%
Test the **CMS** Explore TYPO3 **CMS** backend and the limitless possibilities of TYPO3 **CMS** by using one of [...] To help get you started with TYPO3 **CMS**, we've included usage examples of the standard content elements that have made TYPO3 **CMS** so popular. Examples [...] an easy entry into TYPO3 **CMS**. It can be used as an example to play around or to kickstart your own projects. Included features of the Introduction Package TYPO3 **CMS** custom theme for Twitter

Score	Field	Boost
+ 0.43868986	content	40.0
= 0.43868986 (Inaccurate analysis! Not all parts of the score have been taken into account.)		

History
Relevance: 71%
newer, less sophisticated **CMS** solutions, TYPO3 is a full-grown, enterprise **CMS** that has been widely adopted by companies of all sizes across the world, and is an established standard throughout many universities [...] A Mature, Reliable **CMS** TYPO3 is a mature, stable, and secure platform that has been actively developed and improved for over ten years. Unlike many newer [...] term Content Management was still widely unheard of. Today there are many open source and proprietary **CMS** solutions on the market, but none come close to TYPO3 in terms of sheer functionality and maturity

Use custom Fluid Templates

After these steps solrfluid is usable and using the default Templates, Layouts and Partials. If you want to overwrite them, you can change the TypoScript configuration:

```
plugin.tx_solrfluid {
    view {
        layoutRootPaths.10 = EXT:yourpath/Layouts/
        partialRootPaths.10 = EXT:yourpath/Partials/
        templateRootPaths.10 = EXT:yourpath/Templates/
    }
}
```

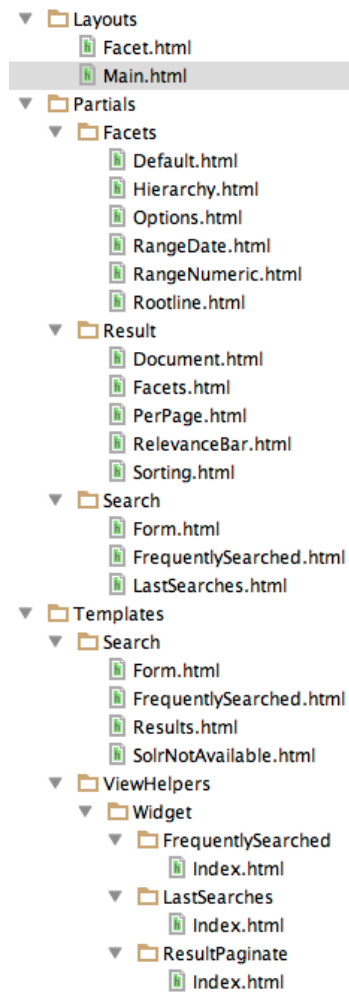
Now you can copy the default partials from the extension to you project path and adapt them to your needs.

Frontend

After the setup and setting up the own custom templates, it is time to explore the template structure and discover the possibilities that solrfluid offers to render your search results, facets and other frontend related elements.

Fluid Template Structure

First we start with a short overview of the template structure. This is just to get an rough overview. The templates will be explained in detail in the template where they belong to:



- **Layouts:** Layouts that are used in the search and the faceting.
- **Partials:**
 - **Facets:** Partials that are use to render the specific facet types.
 - **Result:** Partials that are used during the result rendering (e.g. to render the result document, sorting or perPage selector)
 - **Search:** Partials that are used for the search also when no search was executed.
- **Templates:**
 - **Search:** All templates that are used to render the actions in the SearchController
 - **ViewHelper:** All templates that are use in the widgets (FrequentSearched, LastSearches, ResultPaginate)

Result List

The most important part of a search are the results. The rendering of the results is done in the "Results.html" template (Located in Templates/Search/Results.html)

The following part of the default template iterates over the results and renders every document with the Document.html partial (Partials/Result/Document.html)

```
<s:widget.resultPaginate resultSet="{resultSet}">
  <ol start="{pagination.displayRangeStart}" class="results-list">
    <f:for each="{documents}" as="document">
      <f:render partial="Result/Document" section="Document"
        arguments="{resultSet:resultSet, document:document}" />
    </f:for>
  </ol>
</s:widget.resultPaginate>
```

This structure allows you to use e.g. the fluid if ViewHelper to render a result with a different partial, based on a field value. But as you see in the template above, by default the partial "Result/Document" is used.

The "document" partial is getting the document object. In our case this is an instance of "ApacheSolrForTypo3SolrfluidDomainSearchResultSetSearchResult" the api of this object allows to get the solr field content with "Document->getFieldName()" that can be used as "document.fieldName" in fluid.

Facets

The goal of a good search is, that the user will find what he is looking for as fast as possible. To support this goal you can give information from the results to the user to "drill down" or "filter" the results up to a point where he exactly finds what he was looking for. This concept is called "faceting".

Imagine a user in an online shoe shop is searching for the term "shoe", wouldn't it be useful to allow the user to filter by "gender", "color" and "brand" to find exactly the model where he is looking for?

In the following paragraphs we will get an overview about the different facet types that can be created on a solr field just by adding a few lines of configuration.

Facet Types

A solr field can contain different type of data, where different facets make sence. The simplest facet is an option "facet". The "options facet" just contains a list of values and the user can choose one or many of them. A more complex type could be a "range facet" on a price field. A facet like this needs to allow to filter on a range of a minimum and a maximum value.

The "type" of a facet can be controlled with the "type" property. When nothing is configured there, the facet will be threatened as option facet.

```
plugin.tx_solr.search.faceting.facets.[faceName].type = [typeName]
```

Valid types could be: options | queryGroup | hierarchy | dateRange | numericRange

In the following paragraphs we will introduce the available facet types in EXT:solrfluid and show how to configure them.

Option

The simplest and most often used facet type is the options facet. It renders the items that could be filtered as a simple list.

To setup an simple options facet you can use the following TypeScript snippet:

```
plugin.tx_solr.search {
    faceting = 1
    faceting {
        facets {
            contentType {
                label = Content Type
                field = type
            }
        }
    }
}
```

By using this configuration you create an options facet on the solr field "type" with the name "contentType". This field represents the record type, that was indexed into solr. Shown in the frontend it will look like this:

Narrow Search

Content Type

+ pages (31)

+ tx_solr_file (18)

Summary:

Type	options
DefaultPartial	Partials\Facets\Options.html
Domain Classes	Domain\Search\ResultSet\Facets\OptionBased\Options*

Query Group

The query group facet renders an option list, comparable to the options facet, but the single options are not created from plain solr field values. They are created from dynamic queries.

A typical usecase could be, when you want to offer the possibility to filter on the creation date and want to offer options like "yesterday", "last year" or "more then five years".

With the following example you can configure a query facet:

```
plugin.tx_solr.search {
    faceting = 1
    faceting {
        facets {
            age {
                label = Age
                field = created
                type = queryGroup
            }
        }
    }
}
```

```

        queryGroup {
            week.query = [NOW/DAY-7DAYS TO *]
            old.query = [* TO NOW/DAY-7DAYS]
        }
    }
}

```

The example above will generate an options facet with the output "week" (for items from the last week) and "old" (for items older then one week).

The output in the frontend will look like this:

```

Age
+ week (14)
+ old (35)

```

An more complex example is shipped with this extension and can be enabled by including the template "**Search - (Example) Fluid queryGroup facet on the field created**", this example makes also use of renderingInstructions to render nice labels for the facet.

Summary:

Type	queryGroup
DefaultPartial	Partials\Facets\Options.html
Domain Classes	Domain\Search\ResultSet\Facets\OptionBased\QueryGroup*

Hierarchical

With the hierarchical facets you can render a tree view in the frontend. A common usecase is to render a category tree where a document belongs to.

With the following example you render a very simple rootline tree in TYPO3:

```

plugin.tx_solr.search {
    faceting = 1
    faceting {
        facets {
            pageHierarchy {
                field = rootline
                label = Rootline
                type = hierarchy
            }
        }
    }
}

```

The example above just shows a simple example tree that is just rendering the uid's of the rootline as a tree:

Rootline
+ 1 (31)
+ 14 (18)
+ 16 (4)
+ 17 (1)
+ 18 (1)
+ 19 (1)
+ 15 (1)
+ 21 (1)
+ 22 (1)
+ 23 (1)
+ 24 (1)
+ 25 (1)

A more complex example, that is rendering the pagetree with titles is shipped in the extension. You can use it by including the example TypoScript "**Search - (Example) Fluid hierarchy facet on the rootline field**":

Rootline
+ Congratulations (31)
+ Content Examples (18)
+ News (4)
+ TYPO3 - An idea is born (1)
+ T3UXW09 - The first TYPO3 User eXperience Week (1)
+ The TYPO3 Association is founded (1)
+ Site map (1)

Summary:

Type	hierarchy
DefaultPartial	Partials\Facets\Hierarchy.html
Domain Classes	Domain\Search\ResultSet\Facets\OptionBased\Hierarchy*

Technical solr background:

Technically the hierarchical facet for solr is the same as a flat options facet. The support of hierarchies is implemented, by writing and reading the facet options by a convention:

```
[depth] - /Level1Label/Level2Label
```

When you follow this convention by writing date into a solr field you can render it as hierarchical facet. As example you can check indexing configuration in EXT:solr (EXT:solr/Configuration/TypoScript/Solr/setup.txt)

```
plugin.tx_solr {
    index {
        fieldProcessingInstructions {
            rootline = pageUidToHierarchy
        }
    }
}
```

In this case the "fieldProcessingInstruction" "pageUidToHierarchy" is used to create the rootline for solr in the conventional way.

Date Range

When you want to provide a range filter on a date field in EXT:solr, you can use the type "**dateRange**".

The default partial generates a markup with all needed values in data attributes. Together with the provided jQuery ui implementation you can create an out-of-the-box date range facet.

With the following typoscript you create a date range facet:

```
plugin.tx_solr.search {
    faceting = 1
    faceting {
        creationDateRange {
            label = Created Between
            field = created
            type = dateRange
        }
    }
}
```

In the extension we ship the TypoScript example "**Search - (Example) Fluid dateRange facet with jquery ui datepicker on created field**" that shows how to configure a dateRange facet and load all required javascript files.

When you include this template a date range facet will be shown in the frontend that we look like this:

The screenshot shows a search interface. At the top, there's a 'Created Between' facet with two input boxes. Below it is a calendar for June 2016. The calendar shows dates from 1 to 30. The date 8 is highlighted. To the right of the calendar, there's a list of search results. The first result is 'T3UXW09 - The first TYPO3 User e>'. It has a relevance of 100% and a score of 0. The second result is 'T3UXW09 - The first TYPO3 User e> Behringen (Germany) to work on TY'. It also has a relevance of 100% and a score of 0.

As described before for the date range facet markup and javascript code is required, looking at the example template "**Search - (Example) Fluid dateRange facet with jquery ui datepicker on created field**" in "Configuration/TypoScript/Examples/DateRange" you see that for the jQueryUI implementation the following files are included:

```
page.includeJSFooterlibs {
    solr-jquery = EXT:solr/Resources/JavaScript/JQuery/jquery.min.js
    solr-ui = EXT:solr/Resources/JavaScript/JQuery/jquery-ui.core.min.js
    solr-datepicker = EXT:solr/Resources/JavaScript/JQuery/jquery-ui.datepicker.min.js
    solr-daterange = EXT:solrfluid/Resources/Public/JavaScript/facet_daterange.js
}

page.includeCSS {
    solr-ui = EXT:solr/Resources/Css/JQueryUi/jquery-ui.custom.css
}
```

Numeric Range

Beside dates ranges are also usefull for numeric values. A typical usecase could be a price slider for a products page. With the user interface you should be able to filter the documents for a certain price range.

In the default partial, we also ship a partial with data attributes here to support any custom implementation. By default we will use the current implementation from EXT:solr based on jQueryUi.

The following example configures a **numericRange** facet for the field "**pid**":

```
plugin.tx_solr.search {
    faceting = 1
    faceting {
        pidRangeRange {
            field = pid
            label = Pid Range
            type = numericRange
            numericRange {
                start = 0
                end = 100
            }
        }
    }
}
```

```

    gap = 1
  }
}
}

```

The numeric range facet requires beside the template also a javascript library to render the slider. The example typescript template "**Search - (Example) Fluid numericRange facet with jquery ui slider on pid field**" can be used to see the range slider with jQuery ui for the solr field pid by example.

When you configure a facet on the pid field like this, the frontend will output the following facet:



Beside the implementation with jQueryUi you are free to implement a range slider with any other javascript framework.

Rendering with fluid

Rendering facets with fluid is very flexible, because you can use existing ViewHelpers and implement your own logic in ViewHelpers to support your custom rendering logic.

In the default template the main faceting area on the left side, is done in the following file:

Resources/Private/Partials/Result/Facets.html

This template is used to render only the area for a few facets. The following part is the relevant part where we iterate over the facets:

```

<s:facet.area.group groupName="main" facets="{resultSet.facets.available}">
  <div class="facet-area-main">
    <div class="solr-facets-available secondaryContentSection">
      <div class="csc-header">
        <h3 class="csc-firstHeader">Narrow Search</h3>
      </div>
      <ul class="facets">
        <f:for each="{areaFacets}" as="facet">
          <li class="facet facet-type facet-type-{facet.type}">
            <f:render partial="Facets/{facet.partialName}"
              arguments="{resultSet:resultSet, facet:facet}" />
          </li>
        </f:for>
      </ul>
    </div>
  </div>
</s:facet.area.group>

```

Looking at the code above we see to important details that are important for solrfluid.

Facet Grouping

Autosuggest

The first important part is the **facet.area.group** ViewHelper. By default all facets in the group **main** will be rendered. This value is the default value.

When you now want to render the facet at another place you can change the group with the following TypeScript configuration:

```
plugin.tx_solr.search {
    faceting = 1
    faceting {
        contentType {
            field = type
            label = Content Type
            groupName = bottom
        }
    }
}
```

Now the facet belongs to another group and will not be rendered in the "main" area anymore.

Default Partial

Another important fact is that *Facet->getPartialName()* is used to render the detail partial. The default implementation of a facet will return the default partial, that is able to render this facet.

If you need another rendering for one facet you can overwrite the used partial within the configuration:

```
plugin.tx_solr.search {
    faceting = 1
    faceting {
        contentType {
            field = type
            label = Content Type
            partialName = mySpecialFacet
        }
    }
}
```

Combining all of these concepts together with the flexibility of fluid you are able to render facets in a very flexible way.

Autosuggest

A user of the search typically wants to find the results as fast as possible. To support the user and avoid too much typing solr can create a drop down list of common suggested search terms right after the search input box.

This feature can be easily configured with the following typescript setting:

```
plugin.tx_solr.search {
    suggest = 1
    suggest {
        numberOfSuggestions = 10
        suggestField = spell
    }
}
```

```
}  
}
```

Beside the server related part solrfluid ships the jQueryUi autocomplete implementation to show the suggest results. If you want to configure an the autosuggest by example, you can include the typoscript example template "**(Example) Fluid suggest/autocomplete with jquery ui**".

When everything is configured the frontend will show you a drop down of suggestions when you are typing in the search field:



Sorting

When no sorting is selected the search will order the results by "**relevance**". This relevance is calculated by many factors and has the goal to deliver the best result for the query on the first position. That's what you expect from a search :)

For some usecases you want to change the sorting of the results by a certain field. In an onlineshop a user might want to order the results by the price to find the cheapest product that is matching his query.

A simple sorting can be configured with the following typoscript snippet:

```
plugin.tx_solr.search.sorting >  
plugin.tx_solr.search {  
    sorting = 1  
    sorting {  
        defaultOrder = asc  
  
        options {  
            relevance {  
                field = relevance  
                label = Relevance  
            }  
  
            title {  
                field = sortTitle  
                label = Title  
            }  
        }  
    }  
}
```

With the configuration above the possibility to sort by title is introduced. At the same time the sort by relevance link can be used to reset the sorting to sort by the natural solr relevance.

Sort by

Relevance
▲ [Title]

Used facets

- Content Type: pages (31)
Remove all filters

Narrow Search

Content Type
+ pages (31)
Rootline
+ Congratulations (31)
+ Content Examples (18)
+ News (4)

Parsed Query:
(+MatchAllDocsQuery("*:*")())/no_coord

Searched for "***". Found 31 results in 12 milliseconds. Displayir
Results per page:

1234»

Any language, any character
Relevance: 100%
TYPO3 Speaks Your Language TYPO3 supports all langu
frontend, backend and database interactions run U

Score	Field
= 0	(Inaccurate analysis! Not all parts of the score have been taken into account.)

Community

Templating

The rendering of the sorting is done on "Resources/Private/Partials/Results/Sorting.html" this partial is using the configuration and the view helpers to generate sorting links with the same behaviour as in ext:solr. For sure you can modify this template and use the ViewHelpers in the way how you want to implement your custom sorting.

Results per Page

EXT:solr allows you to configure how many result per page will be shown and at the same time the user can also change this value to an allowed value.

The following configuration can be used to configured the results per page:

```
plugin.tx_solr {
    search {
        results {
            resultsPerPage = 6
            resultsPerPageSwitchOptions = 12, 18, 24
        }
    }
}
```

When you apply the configuration above, the frontend will show 6 search results by default and show the options 12, 18 and 24 to the user to change the amount of visible results

*

Parsed Query:

(+MatchAllDocsQuery(*:*)())/no_coord

Searched for "*". Found 49 results in 11 milliseconds. Displaying

Results per page

✓ 12
18
24

1

2

3

4

5

6

7

8

9

»

Any language, any character

Relevance: 100%

TYPO3 Speaks Your Language TYPO3 supports all languages

Templating

The rendering of the "perPage selector" is done on "Resources/Private/Partials/Results/PerPage.html". This partial is build in a way that the behaviour of the perPage selector is the same as in EXT:solr. If you want to do your custom rendering for example with links instead of a for, you can customize the rendering there.

Ajaxified Results

To improve the use experience and the performance it is possible to load most of the sub requests with ajax.

To activate the **"ajaxification"** you need to include the typoscript template **"Search - Fluid: ajaxify the searchresults with jQuery"**

Search - (Example) Fluid dateRange facet with jquery ui datepicker on created f

Search - (Example) Fluid numericRange facet with jquery ui slider on pid field (s

Search - (Example) Fluid suggest/autocomplete with jquery ui (solrfluid)

Search - Fluid: ajaxify the search results with jQuery (solrfluid)

Include Basis Template:

How it works?

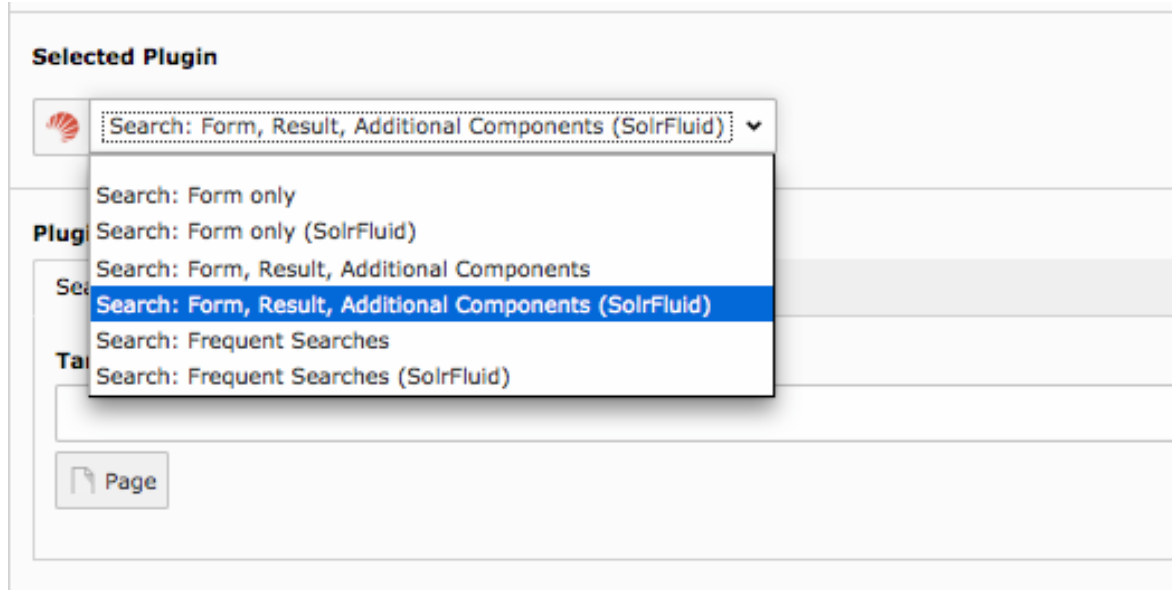
For all links with the css class **"solr-ajaxified"** the javascript search controller triggerst the request against the same search page but with the eID **"tx_solrfluid_search"** which is just rendering the search request. The response is replacing everything in the container **"div.tx_solr"** with the content of the response.

Backend

Plugins

Solrfluid provides the following plugin instances that can be configured in the backend:

- Results plugin: **"Search: Form, Result, Additional Components (SolrFluid)"**
- Form plugin: **"Search: Form only (SolrFluid)"**
- Frequent Searches plugin: **"Search: Frequent Searches (SolrFluid)"**



Results Plugin

The results plugin is the most important plugin of the extension. It is responsible to render a search form and the results.

Flexform Configuration

All configuration can be done with TypoScript and the settings from EXT:solr are used. For some settings it makes sense to overwrite them with the flexform in the plugin settings.

The following settings can be overwritten by instance with the flexform:

"Target Page":

Target page that should be used when a search is submitted. This can be usefull when you want to show the results on another page.

When nothing is configured the current page will be used.

Overwritten TypoScript Path	plugin.tx_solr.search.targetPage
Type:	Integer

"Initialize search with empty query":

If enabled, the results plugin issues a "get everything" query during initialization. This is useful, if you want to create a page that shows all available facets although no search has been issued by the user yet.

Note: Enabling this option alone will not show results of the get everything query. To also show the results of the query, see option *Show results of initial empty query* below.

Overwritten TypoScript Path	plugin.tx_solr.search.initializeWithEmptyQuery
Type:	Boolean

"Show results of initial empty query":

Requires **"Initialize search with empty query"** (above) to be enabled to have any effect. If enabled together with **"Initialize search with empty query"** the results of the initial "get everything" query are shown. This way, in combination with a filter you can easily list a predefined set of results.

Overwritten TypoScript Path	plugin.tx_solr.search.showResultsOfInitialEmptyQuery
Type:	Boolean

"Initialize with query":

This configuration can be used to configure an initial query string that is triggered when the plugin is rendered.

Overwritten TypoScript Path	plugin.tx_solr.search.initializeWithQuery
Type:	String

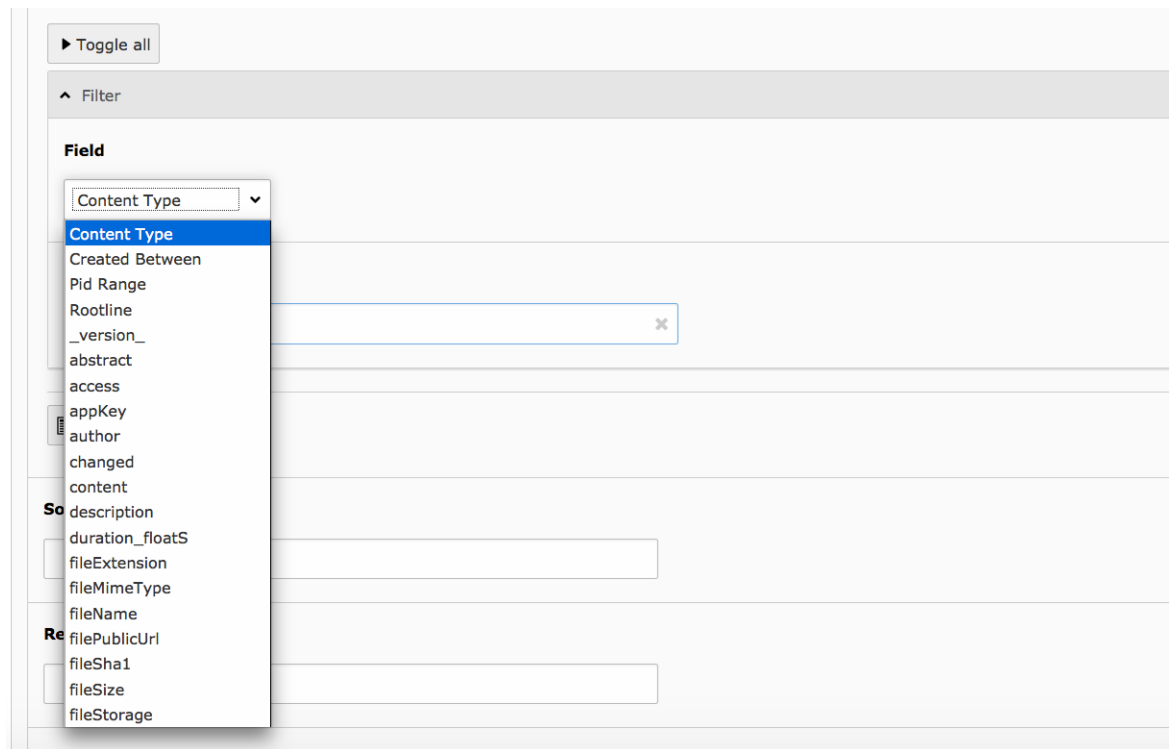
"Show results of initial query":

This option is used to configure if the results of an initial query should be shown.

Overwritten TypoScript Path	plugin.tx_solr.search.showResultsOfInitialQuery
Type:	Boolean

"Filters":

This flexform element allows you to define custom filters by selecting a solr field and a value:



Overwritten TypeScript Path	plugin.tx_solr.search.query.filter.
Type:	Array

"Sorting":

When you want to sort initially by a field value and not by relevance this can be configured here.

Overwritten TypeScript Path	plugin.tx_solr.search.query.sortBy
Type:	String
Example:	title desc

"Boost Function":

A boost function can be useful to influence the relevance calculation and boost some documents to appear more at the beginning of the result list. Technically the parameter will be mapped to the **"bf"** parameter in the solr query.

Use cases for example could be:

- "Give newer documents a higher priority":

This could be done with a recip function:

```
recip(ms(NOW,created),3.16e-11,1,1)
```

- "Give documents with a certain field value a higher priority":

This could be done with:

```
termfreq(type,'tx_solr_file')
```

Overwritten TypeScript Path	plugin.tx_solr.search.query.boostFunction
Type:	String
Example:	recip(ms(NOW,created),3.16e-11,1,1)

See also:

<https://cwiki.apache.org/confluence/display/solr/The+DisMax+Query+Parser#TheDisMaxQueryParser-Thebf%28BoostFunctions%29Parameter>

<https://cwiki.apache.org/confluence/display/solr/Function+Queries>

"Boost Query":

The boostQuery is a query that can be used for boosting. Technically it is mapped to the "**bq**" parameter of the solr query. Compared to boost a function a boost query provides less use cases.

An example could be to boost documents based on a certain field value:

type:tx_solr_file

Overwritten TypeScript Path	plugin.tx_solr.search.query.boostQuery
Type:	String
Example:	type:tx_solr_file

See also:

<https://cwiki.apache.org/confluence/display/solr/The+DisMax+Query+Parser#TheDisMaxQueryParser-Thebq%28BoostQuery%29Parameter>

Development

In this section we want to describe what you need to develop for or based on EXT:solrfluid.

Development Environment

To simplify the development for TYPO3 and solr related components we provide a development environment based on vagrant and the Homestead box of the TYPO3 core.

You can find the development box in the following git repository:

<https://github.com/TYPO3-Solr/solr-typo3-devbox>

When you start the box, you will find a pre-configured environment for the support TYPO3 LTS version and the solr installation that is needed for the installed EXT:solr version.

Testing and Continues Integration

The goal during the development of EXT:solrfluid was, to test most of the components with unit and integration tests.

Unit Tests

For the single classes we've added unit tests whenever we though it is usefull. The unit tests should run very quickly, to git a quick response.

Integration Tests

As in EXT:solr the integration tests are more complex and test the integration of the different components. Since a database server and a solr server is required this is needed to run the integration test suite. During the bootstrap of the test environment, we use the TYPO3 core functionality for database tests and we install a local solr server with out install script.

To simplify the local usage of the unit and integration tests, we ship a few bash script that support you to get everything started.

Bootstrapping the Test Environment

When you want to start the testrunner in your shell you need to bootstrap it once:

```
source ./Build/Test/bootstrap.sh --local
```

The bootstrapper will prompt for some values:

```
/v/w/7/t/e/solrfluid git:feature/add-documentation $ source ./Build/Test/bootstrap.sh --local
Choose a TYPO3 Version (e.g. dev-master,~6.2.17,~7.6.5): ~7.6.9
Choose a EXT:solr Version (e.g. dev-master,~3.1.1): dev-master
Choose a database hostname: localhost
Choose a database name: test
Choose a database user: root
Choose a database password: supersecret
You are running composer with xdebug enabled. This has a major impact on runtime performance. See https://getcomposer.org/xdebug
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
```

When the bootstrapper was finished successful the following was done:

Development Workflow

- Environment variables for the TYPO3 testing framework have been exported
- Test database was created
- Test solr instance was created

Afterwards you can run the ci suite in your shell

Running the ci Suite

When the test environment was bootstrapped correctly you can start the test runner:

```
./Build/Test/cibuild.sh
```

When everything is configured correctly all tests should run through and you should get a green bar:

```
/v/w/7/t/e/solrfluid git:feature/add-documentation $ ./Build/Test/cibuild.sh
PWD: /var/www/7.6.local.typo3.org/typo3conf/ext/solrfluid
Run unit tests
PHPUnit 5.3.4 by Sebastian Bergmann and contributors.

..... 65 / 70 ( 92%)
..... 70 / 70 (100%)

Time: 3.96 seconds, Memory: 29.50MB

OK (70 tests, 145 assertions)

Generating code coverage report in HTML format ... done
Run integration tests
PHPUnit 5.3.4 by Sebastian Bergmann and contributors.

..... 23 / 23 (100%)

Time: 3.42 minutes, Memory: 14.25MB

OK (23 tests, 71 assertions)

Generating code coverage report in HTML format ... done
```

Development Workflow

For the development of EXT:solrfluid we use our internal git repository. For the git structure we are using "**git flow**". Phabricator & Arcanist can be used for code reviews.

The following steps are required to work on a task in solrfluid:

- **Install git flow**
 - See <https://github.com/nvie/gitflow> and <https://github.com/nvie/gitflow/wiki/Installation>
- Install [arcanist](<https://secure.phabricator.com/book/phabricator/article/arcanist/>)
- Checkout origin develop branch (git checkout --track -b develop origin/develop)

- Git flow initialize `git flow init -d`
- Create new feature branch (`git flow feature start my-new-feature`)
- Run tests (See CI Chapter of this document)
- Commit your changes (`git commit -am 'Add some feature'`)
- Send changes to code review (`arc diff`)
- Once the review is complete, you will run (`arc land [branch]`) in the review branch, which will merge its contents into the deploy branch you branched off of, and then delete the review branch. for help run (`arc help land`)

Code Structure

The components of EXT:solrfluid have been developed with the domain driven design (DDD) approach (https://de.wikipedia.org/wiki/Domain-driven_Design) for our extension we tried to separate the code by the following layers:

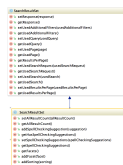
- **Domain:** Everything that is related to the "search" domain should be implemented here.
- **System:** Everything that is related to the "system" (e.g. TYPO3 specific) should be implemented here.

Domain Layer & Domain Model

The classes of the domain layer are located in "Classes/Domain" and should contain everything that is related to the "search domain".

ResultSet

The "SearchResultSet" is the main entity that you get passed to the view. It can be used to access all search related objects on your result page.



The `SearchResultSet` can be used e.g. to get facets and spelling suggestions. A focus for the first release was a new domain model for facets, that can be rendered with fluid or any other template engine.

Facets

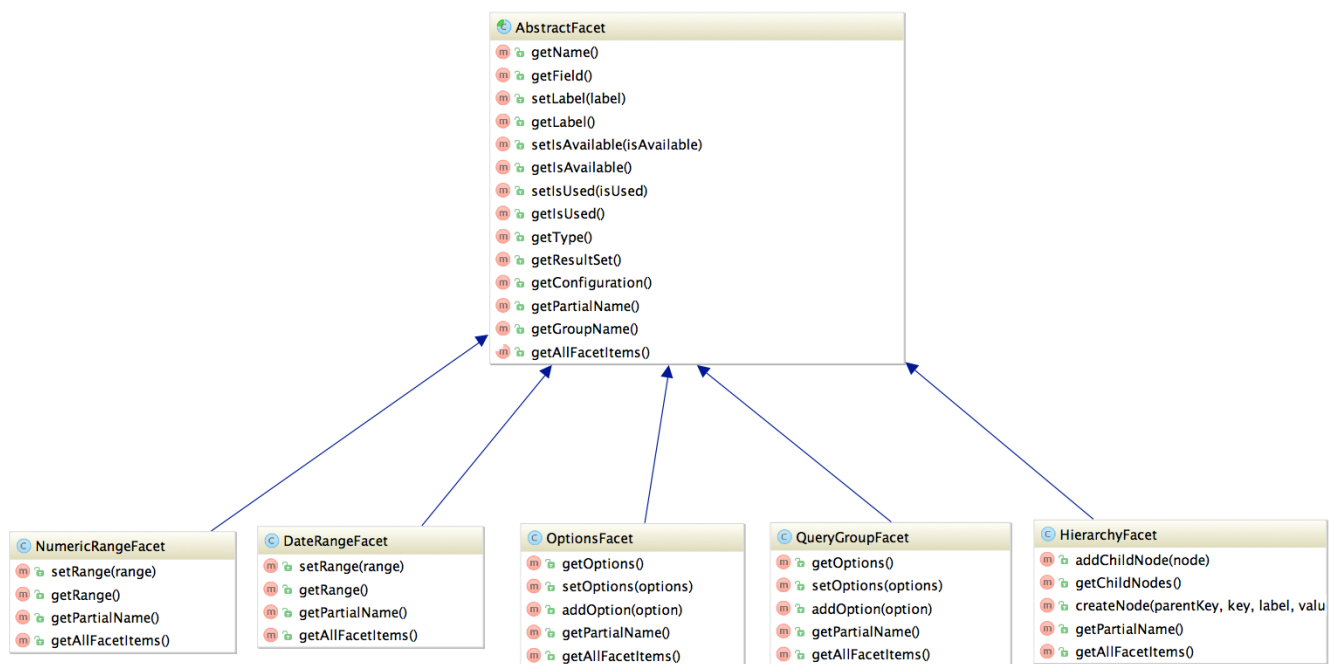
The following UML diagram shows the implemented facets in EXT:solrfluid. Every facet has one or more facet items attached. For the **OptionsFacet** the FacetItem is an **Option**, for the **NumericRangeFacet** a **NumericRange**.

Rendering of a facet:

Based on the **"type"** TypeScript configuration the **"FacetParserRegistry"** chooses the responsible facet parser class that is used to create the object structure from the solr response. Each facet type is shipped with a default fluid partial, that is able to render such a facet.

The typoscript configuration "**partialName**" can be used to force the rendering with another fluid partial.

For advanced use cases you can use the **"FacetParserRegistry"** to register your own facet type or overwrite the facet parser for a certain facet type.



As you see in the diagram above solrfluid ships a clean object structure of the facets, that you can render in your custom templates as you need them.

ViewHelpers

Beside the controllers, the domain objects and the templates we ship a few useful view helpers. To avoid a strong coupling between the extension and fluid as template engine we tried to keep all ViewHelpers as "slim" as possible. Whenever it was possible we moved the logic into custom service classes and just use them in the ViewHelper.

Since everything belongs to the **"SearchResultSet"** and we wanted to avoid the need of passing this object around from "template to template" and "partial to partial" we decided to provide an own "ControllerContext" that referenced the "SearchResultSet". With this approach, it is possible to access the **"SearchResultSet"** in every ViewHelper.

With the current release we ship the following concrete ViewHelpers:

Path	Description
s:debug.documentScoreAnalyzer	Used to render the score analysis.
s:debug.query	Shows the solr query debug information.
s:document.highlightResult	Performs the highlighting on a document.
s:document.relevance	Shows the relevance information for a document.
s:facet.area.group	Filters the facets in the rendering scope to one group.
s:uri.facet.addFacetItem	Add's a facet item to the current url.
s:uri.facet.removeAllFacets	Removes all facet items from the current url.
s:uri.facet.removeFacetItem	Removes a single facet item from the url.

ViewHelpers

s:uri.facet.setFacetItem	Sets one single item for a facet (and removes other setted)
s:uri.paginate.resultPage	Creates a link to a result page of the current search.
s:uri.search.currentSearch	Creates a link to the current search (with facets, sorting...)
s:uri.search.startNewSearch	Creates a link for a new search by a term.
s:uri.sorting.removeSorting	Creates a link to the current search and removes the sorting.
s:uri.sorting.setSorting	Creates a link to the current search and sets a new sorting.
s:pageBrowserRange	Provides the range data for the pagination.
s:searchForm	Renders the searchForm.
s:translate	Custom translate ViewHelper (uses translations from ext:solr)