EXT:solrfluid Fluid Rendering for ApacheSolrForTypo3

Release 2.0.0

Timo Hund, Markus Friedrich, Frans Saris and Daniel Siepmann

2017-04-05 13:28

CONTENTS

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.

inspiring people to *share*.   TYPO3

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

inspiring people to *share*. TYPO3

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

inspiring people to *share*. TYPO3

WHAT DOES IT DO?

The solrfluid addon allows you to use the well known template engine fluid, together with EXT:solr. To achieve this, solfluid 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:
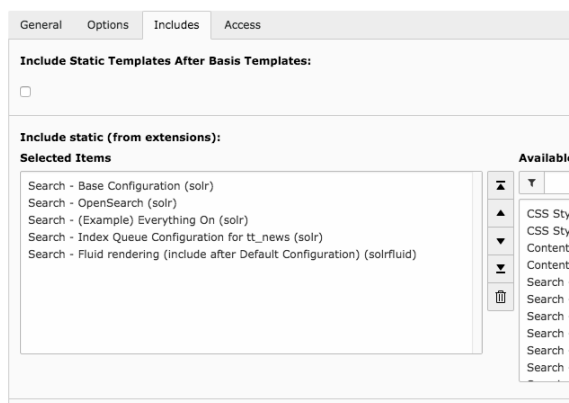
Use plugin instances from EXT:solrfluid

You can used the fluid rendering instead the normal rendering by using the plugins that are postfixed with "solrfluid" instead the normal pi based plugins.

Example:

Instead using **"Search: Form, Result, Additional Components"** use **"Search: Form, Result, Additional Components (SolrFluid)"**. After these steps solrfluid is usable and using the default templates. If you want to use your own once, you can change the template location.

Check the Frontend

When everything is configured correctly you can open the page in the frontend a do a search.

The example below shows a search for "cms" with an indexed TYPO3 introduction package:

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.

inspiring people to **share**. TYPO3

FRONTEND

After the setup and setting up the own custom templates, it is time to explore the template structure and discover the possbilities 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 rought overview. The templates will be explained in detail in the template where they belong to:

```
▼ ☐ Layouts
      ⓘ Facet.html
      ⓘ Main.html
▼ ☐ Partials
   ▼ ☐ Facets
         ⓘ Default.html
         ⓘ Hierarchy.html
         ⓘ Options.html
         ⓘ RangeDate.html
         ⓘ RangeNumeric.html
         ⓘ Rootline.html
   ▼ ☐ Result
         ⓘ Document.html
         ⓘ Facets.html
         ⓘ PerPage.html
         ⓘ RelevanceBar.html
         ⓘ Sorting.html
   ▼ ☐ Search
         ⓘ Form.html
         ⓘ FrequentlySearched.html
         ⓘ LastSearches.html
▼ ☐ Templates
   ▼ ☐ Search
         ⓘ Form.html
         ⓘ FrequentlySearched.html
         ⓘ Results.html
         ⓘ SolrNotAvailable.html
   ▼ ☐ ViewHelpers
      ▼ ☐ Widget
         ▼ ☐ FrequentlySearched
               ⓘ Index.html
         ▼ ☐ LastSearches
               ⓘ Index.html
         ▼ ☐ ResultPaginate
               ⓘ Index.html
```

- 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, Result-Paginate)

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 threated 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 TypoScript snipped:

```
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, compareable 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 possiblity 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:

inspiring people to **share**. 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
        }
    }
}
```

inspiring people to *share*.   TYPO3

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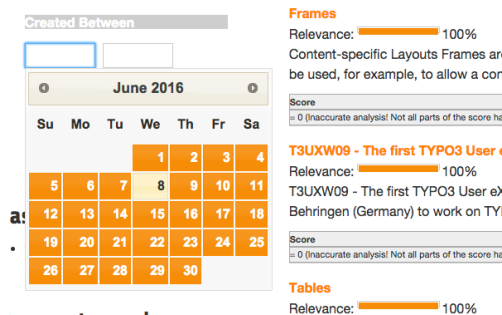
As described before for the date range facet markup and javascript code is required, looking at the example template **"Search - (Example) Fluid dateRange facet with jquery ui datepicker on created field"** in "Configuration/TypoScript/Examples/DateRange" you see that for the jQueryUi implementation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

```html
<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

The first important part if 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 TypoScript 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 Partials

Another important fact is that *Facet->getPartianName()* 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 want to find the results a fast as possible. To support the user and avoid to 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 typoscript 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:

inspiring people to *share*.   TYPO3
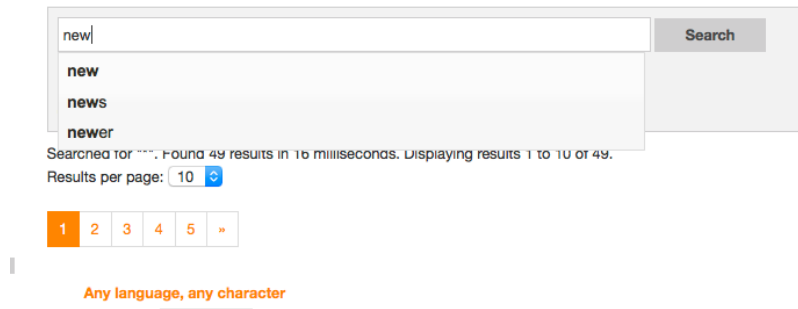
Sorting

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 snipped:

```
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.

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 modifiy 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
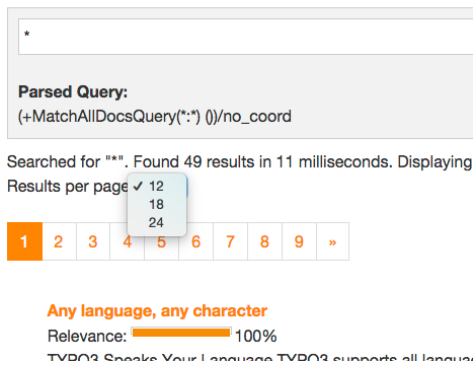
Templating

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"**
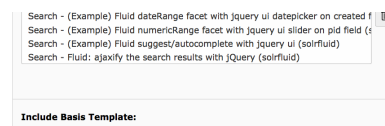
How it works?

For all links with the css class **"solr-ajaxified"** the javascript search controller triggers the request against the same search page but with the type *"7383"* which is just rendering the search request. The response is replacing everything in the container "div.tx_solr" with the content of the response.

inspiring people to *share*. TYPO3

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)"**
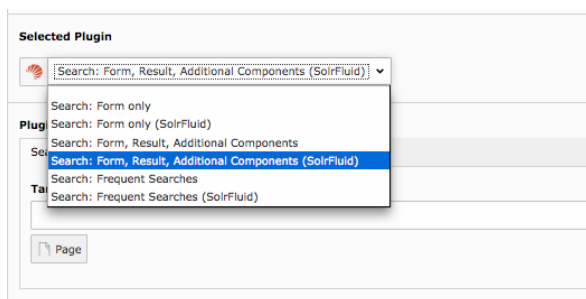
Results Plugin

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 sence 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.

inspiring people to *share*. TYPO3

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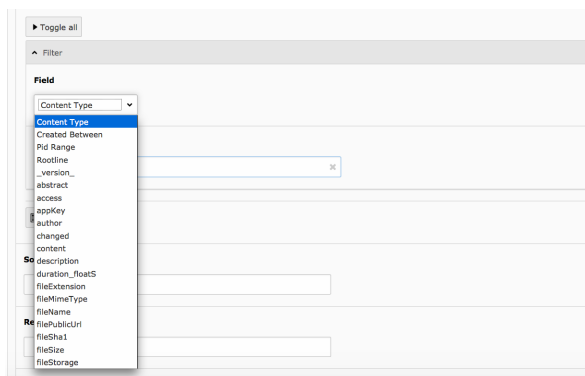
Overwritten TypoScript Path	plugin.tx_solr.search.query.filter.
Type:	Array

"Sorting":

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

Overwritten TypoScript 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 never 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 TypoScript 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 TypoScript 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 boostrapper was finished successful the following was done:

- 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 boostrapped 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:

inspiring people to **share**. TYPO3

```
/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 initialze `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".

```
/v/w/7/t/e/solrfluid git:feature/add-documentation $ ./Build/Test/cibuild.sh
```

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"** TypoScript configuration the **"FacetParserRegistry"** chooses the responsible facet parser class that is used to create the object structure from the solr repsonse. 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.
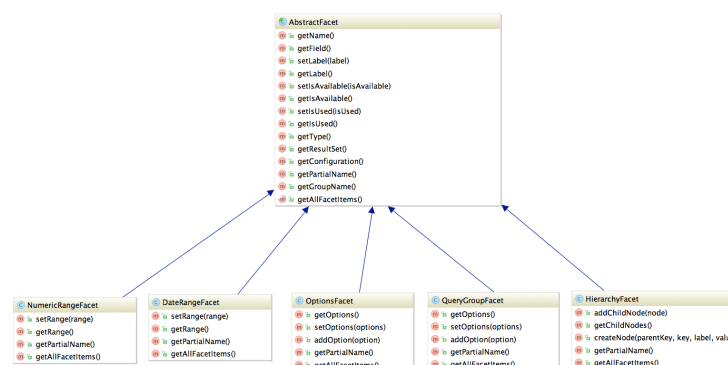
As you see in the diagram above solrfluid ships a clean object structure of the facets, that you can render in your custom templates as you need them.

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 ob 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.removeFacet	Removes all options from one facet.
s:uri.facet.removeFacetItem	Removes a single facet item from the url.
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)

RELEASES

EXT:solrfluid Releasenotes 2.0.0

This release is a follow up release of EXT:solr 6.1 and the last release for TYPO3 CMS LTS 7.

1. Add implementation of facetOptions reverseOrder

Added tests Refactored code structure to have AbstractOptionsFacet Allow to use manualSortOrder and reverseOrder for Options and QueryGroup facets

2. [BUGFIX] Default value for cHash on Paginator

3. [BUGFIX] Use gap from facet and data from configuration

In the template no gap was rendered, because it was read from the wrong object.

This change:

- Adjusts the template to call getGap from the range object
- Uses the gap from the settings instead from the solr response
- Adds some assertions to the NumericRangeParserTest

4. Increase version of jQuery to v3.1.1

5. [BUGFIX] Fix hierarchy facet

- Fix parsing of hierarchy facet
- Add testcase to reproduce bug

6. [FEATURE] Add parentNode and childNodeSelected getters to Node

- The following new Methods to the Node class: * getParentNode * getHasParentNode * getHasChildNodeSelected

7. [BUGFIX] Fatal in backend when 'flexParentDatabaseRow' is not set

8. [BUGFIX] Fix numeric range support

Negative numeric ranges were not supported since the parser fails to split active filter values correctly if there are negative values in it. This commit adapts the parsing of the active filter values and fixes this issue.

9. [BUGFIX] Don't escape document values (breaking change)

As we use fluid to render all data and fluid has build-in escaping of all values we don't need to escaped all values during "building" of the document object.

As this could be a breaking change when you depend on the escaping in your templates of you already "fixed" it your self by using *{document.title -> f:format.raw()}* this behaviour is only enabled when you set:

plugin.tx_solrfluid.features.useRawDocuments = 1

This is now part of the default typoscript setup

Outlook

In the next releases of EXT:solr this extension will be a part of EXT:solr.

Thanks

Thanks to all contributors:

- Andreas Allacher
- Markus Friedrich
- Sascha Nowak
- Sascha Egerer
- Timo Hund
- Frans Saris
- Rafael Kähm

In addition we also want to say "thank you" to all of our EB Partners that subscribed a new partnership for 2016:

- Arrabiata Solutions GmbH & Co. KG
- avonis
- Bank CIC AG
- Bitmotion GmbH
- Citkomm services GmbH
- cron IT
- CS2 AG
- Cosmoblonde GmbH
- Daniz online markting
- datenwerk innovationsagentur gmbh
- Die Medialen GmbH
- die_schnittsteller GmbH

- E-magineurs

- Fernando Hernáez Lopez

- Future Connection AG

- Gernot Leitgab

- .hausformat

- Hirsch & Wölfl GmbH

- hs-digital GmbH

- IHK Neubrandenburg

- internezzo AG

- jweiland.net

- L.N. Schaffrath DigitalMedien GmbH

- mehrwert intermediale kommunikation GmbH

- netlogix GmbH & Co. KG

- Pixel Ink

- Pixelpark AG

- pixolith GmbH & Co. KG

- polargold GmbH

- portrino GmbH

- Q3i GmbH & Co. KG

- raphael gmbh

- RUAG Corporate Services AG

- sitegeist media solutions GmbH

- ST3 Elkartea

- Star Finanz-Software Entwicklung und Vertriebs GmbH

- Stefan Galinski Interndienstleistungen

- Speedpartner GmbH

- sunzinet AG

- Systime A/S

- SYZYGY Deutschland GmbH

- tecsis GmbH

- web-vision GmbH

- websedit AG - Internetagentur

- Webstobe GmbH

- werkraum GmbH

- WIND Internet

- wow! solution

- zdreicon AG

Thanks also to our partners who already singed up for a 2017 partnership (EB2017):

- Amedick & Sommer Neue Medien GmbH

- cron IT GmbH

- b:dreizehn GmbH

- Die Medialen GmbH

- Leibniz Universität IT Services, Hannover

- LOUIS INTERNET

- polargold GmbH

- Mercedes-AMG GmbH

- Triplesense Reply GmbH

- zdreicom AG

Thanks to everyone who helped in creating this release!

Support us in 2017 by becoming an EB partner:

http://www.typo3-solr.com/en/contact/

EXT:solrfluid Releasenotes 1.2.0

This release is a bugfix and maintenance only release it contains:

1. Make TranslateViewHelper compatible with TYPO3 8

The render method was removed and the implementation of the core was used and only renderStatic is overwritten now.

2. Allow to override 'view' on plugin basis

The plugin configuration is merged together in the AbstractBaseController but the view property was not part of this merge. So it was not possible to override the view typoscript configuration to define custom templates on a per plugin basis. This is now possible and allows configurations like this:

```
tt_content.lexiconSearch = USER
tt_content.lexiconSearch {
    userFunc = TYPO3\CMS\Extbase\Core\Bootstrap->run
    extensionName = Solrfluid
    pluginName = pi_result
    vendorName = ApacheSolrForTypo3

    view {
        templateRootPaths {
            100 = xxx/Resources/Private/Templates/Solr/MyPath
        }
    }

    search {
        targetPage = 4711

        query {
            filter {
                onlyLexiconPages = documentType_stringS:lexicon
            }
        }
```

```
    }
}
```

3. Make search form cacheable

The form action is now cacheable.

4. buildSearchRequest should be protected

The method buildSearchRequest is now protected, to allow to overwrite it.

5. Use objectManager to create facet and result objects

The facet and result objects are now created with the object manager to allow dependency injection when using custom classes.

Thanks

Thanks to all contributors:

- Andreas Allacher
- Markus Friedrich
- Sascha Nowak
- Sascha Egerer
- Timo Hund

In addition we also want to say "thank you" to all of our EB Partners that subscribed a new partnership for 2016:

- Arrabiata Solutions GmbH & Co. KG
- avonis
- Bank CIC AG
- Bitmotion GmbH
- Citkomm services GmbH
- cron IT
- CS2 AG
- Cosmoblonde GmbH
- Daniz online markting
- datenwerk innovationsagentur gmbh
- Die Medialen GmbH
- die_schnittsteller GmbH
- E-magineurs
- Fernando Hernáez Lopez
- Future Connection AG
- Gernot Leitgab
- .hausformat
- Hirsch & Wölfl GmbH

- hs-digital GmbH

- IHK Neubrandenburg

- internezzo AG

- jweiland.net

- L.N. Schaffrath DigitalMedien GmbH

- mehrwert intermediale kommunikation GmbH

- netlogix GmbH & Co. KG

- Pixel Ink

- Pixelpark AG

- pixolith GmbH & Co. KG

- polargold GmbH

- portrino GmbH

- Q3i GmbH & Co. KG

- raphael gmbh

- RUAG Corporate Services AG

- sitegeist media solutions GmbH

- ST3 Elkartea

- Star Finanz-Software Entwicklung und Vertriebs GmbH

- Stefan Galinski Interndienstleistungen

- Speedpartner GmbH

- sunzinet AG

- Systime A/S

- SYZYGY Deutschland GmbH

- tecsis GmbH

- web-vision GmbH

- websedit AG - Internetagentur

- Webstobe GmbH

- werkraum GmbH

- WIND Internet

- wow! solution

- zdreicon AG

Thanks also to our partners who already singed up for a 2017 partnership (EB2017):

- Amedick & Sommer Neue Medien GmbH

- cron IT GmbH

- b:dreizehn GmbH

- Die Medialen GmbH

- Leibniz Universität IT Services, Hannover

- LOUIS INTERNET

- polargold GmbH

inspiring people to **share**. TYPO3

- Mercedes-AMG GmbH

- Triplesense Reply GmbH

- zdreicom AG

Thanks to everyone who helped in creating this release!

Support us in 2017 by becoming an EB partner:

EXT:solrfluid Releasenotes 1.1.0

This release provides a few bugfixes and new features from the first expierences of solrfluid.

1. New ViewHelper: "Uri/Facet/RemoveFacet"

This ViewHelper can be used to remove all selected items for a whole facet.

```
<a href="{s:uri.facet.removeFacet(facet: facet)}">Reset values</a>
```

Thanks to Frans Saris for implementing this.

2. Implement ArrayAccess for "AbstractCollection"

With this you are able to retrieve a facet in the template just by using the name.

```
{resultSet.facets.<facetName>}
```

Thanks to Frans Saris for implementing this.

3. Fixed plugin instance name of the Search Form to "pi_search".

Because of a bug in version 1.0.0 you where not able to add a search form as plugin instance. This is fixed now by using a consistent, same key "pi_search" as in EXT:solr.

Thanks to Thomas Beck for reporting this problem.

4. Moving the query initialization to initializeAction.

We now initialize the query parameter in initializeAction and pass it as argument to have it available in the extbase context.

Thanks to Daniel Siepmann for implementing this.

5. Add missing s:translate calls

We added some missing calls for s:translate to have the possibility to translate all labels.

Thanks to Thomas Beck for reporting this problem.

EXT:solrfluid Releasenotes 1.0.0

We're very happy to announce the release of EXT:solrfluid 1.0.0 today. EXT:solrfluid allows you to render your search results with the fluid templating engine.

Key Features

With solr fluid we provide the following key features:

Flexible Domain Model to render Search Results and Facets

Solr fluid extends the domain model of EXT:solr and provides a re-implementation of the facets to have an optimal domain model for rendering in the view.

Documentation included

With EXT:solrfluid 1.0.0 we ship a documentation of the extension in the rst format. This should help you to get started with EXT:solrfluid very quickly.

High Performance

To ensure the performance of solrfluid is excellent we did several profiling sessions. Beside small adjustments we did the following bigger optimizations:

Compilable ViewHelpers

Most of the ViewHelpers that we ship in solrfluid are compileable. TYPO3 compiles them into a single PHP closure and caches them in the filesystem. With this approach ViewHelpers in EXT:solr are build to be fast.

Smart Link Building

Since links for new facets are often the same, the typolink calls are replaced with markers and cached to reduce the creation of links to a minimum.

Ajaxified Search Results

When you browse in the results page, all subpages can be retrieved with ajax over a custom TYPO3 pagetype. Since the overhead is much less, a sub-request, is more then two times faster then a regular request.

Unit and Integration tested

At the beginning we started with porting the existing integration tests from EXT:solr to EXT:solrfluid and re implement the frontend rendering based on that. Beside that, we tried to cover new code with unit test. All changes that are done, are checked with travis-ci to make sure that the quality of the code is good.

Minor Features

Beside that solrfluid provides some minor features that simplify the implementation of solr projects

Easy Filter configuration in the backend

When you add a new Plugin instance of EXT:solr in the TYPO3 Backend you can now simply configure the entry filters with a flexform. The flexform shows you the available fields from the solr server that can be used to create filters.

Single View for solr documents

The SearchController provides a detailAction. It requires the arguments $documentId and can be used to render a single view of a solr document directly with fluid.

If you want to know more details about the features and how to use them, you should read the documentation, that is linked below this article.

Thanks

Thanks to the following contributors and partners that supported the work on solrfluid.

- Frans Saris:

Thanks for working on the fluid integration and a productive codesprint in Venlo.

- Daniel Siepmann

Thanks for using solrfluid and contributing to the backend and documentation.

- Edward Lenssen

Thanks for hosting the first codesprint @beechit.

- Olivier Dobberkau

Thanks for supporting the codesprint and the development.

In addition we also want to say "thank you" to all of our EB Partners that subscribed a new partnership for 2016:

- Bank CIC AG

- CS2 AG

- Cosmoblone GmbH

- Daniz online markting

- datenwerk innovationsagentur gmbh

- die_schnittsteller GmbH

- E-magineurs

- Fernando Hernáez Lopez

- Future Connection AG

- Hirsch & Wölfl GmbH

- hs-digital GmbH

- L.N. Schaffrath DigitalMedien GmbH

- pixolith GmbH & Co. KG

- Q3i GmbH & Co. KG

- RUAG Corporate Services AG

- ST3 Elkartea

- Star Finanz-Software Entwicklung und Vertriebs GmbH

- Stefan Galinski Interndienstleistungen

- Systime A/S

- websedit AG - Internetagentur

- Webstobe GmbH

- web-vision GmbH

How to get it?

All EB partner can download the release 1.0 from our download page. Not yet a partner? Call dkd +49 (0)69 - 247 52 18-0 to sign up for a partner ship and be one of the first users of solrfluid.