# Documentation for EXT:solr the Apache Solr integration for TYPO3

***Release 6.1.0***

**Ingo Renner, Timo Hund, Markus Friedrich**

**2017-04-06 11:38**

# CONTENTS

CHAPTER

ONE

# WHAT DOES IT DO?

Apache Solr for TYPO3 is the search engine you were looking for with special features such as **Facetted Search** or **Synonym Support** and an incredibly fast response times of results within milliseconds.

When development started, the primary goal was to create a replacement for Indexed Search. With the initial public release at T3CON09 in Frankfurt, Germany that goal was reached and even passed by adding features which Indexed Search does not support.

CHAPTER

TWO

# FEATURE LIST

- Facetted Search
- Spellchecking **/ Did you mean**
- **Multi Language Support**
- Search word highlighting
- Field Boosting for fine tuning the importance of certain index fields
- **Frontend User Group Access Restrictions Support**
- Stop word Support
- Synonym Support
- **Auto complete / Auto suggest**
- Language Analysis / Support for inflected word forms
- Content Elevation **/ Paid Search Results /** Editorial Content
- Sorting of Results
- Content indexing through a near instant backend **Index Queue**
- Auto correction (search for the first suggestion)
- and more...

CHAPTER

THREE

# DEVELOPMENT AND PARTNERSHIP

The extension is developed in an open source way and the source code is completely available on github. Releases to the TYPO3 TER are done in regular time frames.

To make the development possible you can join a partner ship with dkd. By joining the partner program you have the following benefits:

- You support the further development on EXT:solr
- **You get access to addon's that are not public available**
  - Indexing of files from FAL (TYPO3 File Abstraction Layer)
  - Fluid templating (Result rendering with Fluid Template engine)
- You can be mentioned as a sponsor on http://www.typo3-solr.com
- You can included support based on your subscription

If you are interested to become a partner visit http://www.typo3-solr.com or call dkd +49 (0)69 - 247 52 18-0.

CHAPTER

FOUR

# GETTING STARTED

In this chapter we would like to give you a quick introduction into the very first steps with EXT:solr.

After reading this chapter you will know:

How to...

- Start a solr server
- Install the extension and connect your TYPO3 system to your solr server
- Index a few pages and see the search results on your website

## Solr

First you need to install Solr itself. There are several ways to do so:

### Using Hosted-solr.com

If you want to start simple and just create a solr core with a click. You can use hosted-solr.com. For a small fee you get your own solr core in seconds, configured to be used with EXT:solr.

### Shipped install script

With the extension we ship and install script that can be used for a **development** context. It creates a solr server with a core for all languages. This script is located in "Resources/Private/Install" an it installs a configured solr server that is useable with EXT:solr.

By default this script is not executable and you need to add the execute permissions to your user to run it.

The example below shows how to install a solr server to /home/developer

```
chmod u+x ./Resources/Private/Install/install-solr.sh
./Resources/Private/Install/install-solr.sh -p /home/developer
```

After running the script you are able to open a solr server with over the loopback address. Which means, when you want to access it from outside, you need to create an ssh tunnel.

## Docker

You can use docker to install your solr server with a small effort. With the extension we provide a Dockerfile, that creates a container with a core for all languages ready to run. This helps you to setup a container very quickly.

To build the images, simply type one of the following:

```
docker build -t typo3-solr .
```

Prepare the data folder (data is shared with the docker container by user and group with UID/GID 8983):

```
mkdir -p .solrdata
chmod g+w .solrdata
chown :8983 .solrdata
```

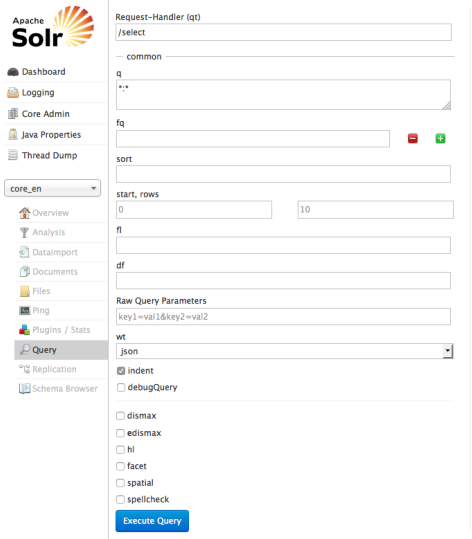
To run the container (only run one of the following):

```
docker run -d -p 127.0.0.1:8983:8983 -v "$PWD/.solrdata:/opt/solr/server/solr/data/
↪" typo3-solr
```

To check whether Solr is up and running head over to:

`http://<ip>:8983/solr/#/core_en/query`.

You should see the web interface of Solr to run queries:



**Important**: The image "typo3-solr" ships a default core for all languages. The data of the cores is stored on a data volume. When you want to update the container, you can just start a new container using the data volume of the old container. But at the same time this has the limitation, that you should only use this image with the default cores! If you want to create custom cores with a different configuration please read the section "Advanced Docker Usage"

#### Advanced Docker Usage

Our image has the intension to create running cores out of the box. This implies, that the schema is inside the container. The intension in our integration was to stay as close as possible to the official Apache Solr docker images. Sometimes it might make sence that you use the official image directly instead of our image. An example could be when you want to have the solrconfig, schema and data outside of the container.

The following example shows how you can run our configuration with the official Apache Solr docker container by mounting the configuration and data from a volume (When using Docker on macOS make sure you've added the volume folder to "Preferences -> File Sharing").

```
mkdir -p ~/mysolr
cp -r Resources/Private/Solr/* ~/mysolr
sudo chown -R :8983 ~/mysolr
docker run -d -p 127.0.0.1:8983:8983 -v ~/mysolr:/opt/solr/server/solr/ solr:6.3.0
```

### Other Setup

Beside the install script and Docker there are various possibilities to setup solr. All of these possibilities are not officially supported, but the simplify the setup i want to mention them shortly here and summarize the needed steps.

#### Known Installers

All of these installers can be used to setup a plain, reboot save solr server:

- Use the installer shipped with solr itself (bin/install_solr_service.sh):

Allows to install solr on many distributions including init scripts (At the time of development ubuntu 16.04 was not supported and therefore it was no option for us to use it).

- Use chef / ansible / whatever dev ops tool:

Allows you to setup a solr server with your DevOps tool.

e.g. https://galaxy.ansible.com/geerlingguy/solr/ (ansible) or https://supermarket.chef.io/cookbooks/solr (chef)

#### Deployment of EXT:solr configuration into Apache Solr

Since EXT:solr 6.0.0 the configuration and all jar files are shipped in one "configSet". The goal of this approach is to make the deployment much easier.

All you need to do is, you need to copy the configSet directory into your prepared solr installation and replace the solr.xml file. In the installer we do it like this:

```
cp -r ${EXTENSION_ROOTPATH}/Resources/Private/Solr/configsets ${SOLR_INSTALL_DIR}/
↪server/solr
cp ${EXTENSION_ROOTPATH}/Resources/Private/Solr/solr.xml ${SOLR_INSTALL_DIR}/server/
↪solr/solr.xml
```

After this, you can decide if you want to create the default cores by copying the default core.properties files or if you want to create a core with the solr rest api.

Copy the default cores:

```
cp -r ${EXTENSION_ROOTPATH}/Resources/Private/Solr/cores ${SOLR_INSTALL_DIR}/server/
↪solr
```

Create a core with the rest api:

```
curl "http://localhost:8983/solr/admin/cores?action=CREATE&name=core_de&
↪configSet=ext_solr_6_0_0&schema=german/schema.xml&dataDir=dataDir=../../data/
↪german"
```

After installing the solr server and deploying all schemata, the TYPO3 reports module helps you to verify if your setup fits to the requirements of EXT:solr

You now have a fully working, pre configured Solr running to start with

started-install-extension.

## Install EXT:solr

### Install from TER using the TYPO3 Extension Manager

You can simply install stable versions of EXT:solr using the Extension Manager from the TYPO3 backend.

1. Go to the **Extension Manager**, select **Get Extensions** and search for "solr".
2. Install the Extension.
3. The Extension Manager will also install EXT:scheduler if not installed already for running the indexing tasks
4. While developing we recommend installing devlog for easier error detection, too.

### Install from git

Alternatively, you can also get the latest development version from GitHub:

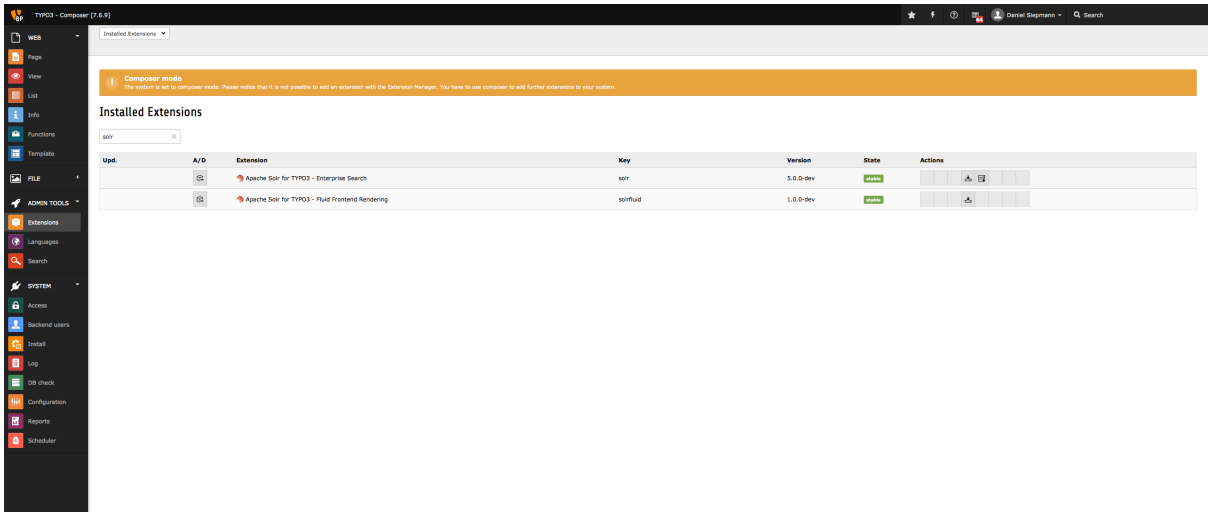
```
$ git clone git@github.com:TYPO3-Solr/ext-solr.git solr
```

### Install with composer

Install this TYPO3 Extension `solr` via TYPO3 Extension Manager as usual, or via `composer` by running:

```
composer require apache-solr-for-typo3/solr
```

Head over to the Extension Manager module and activate the Extension.



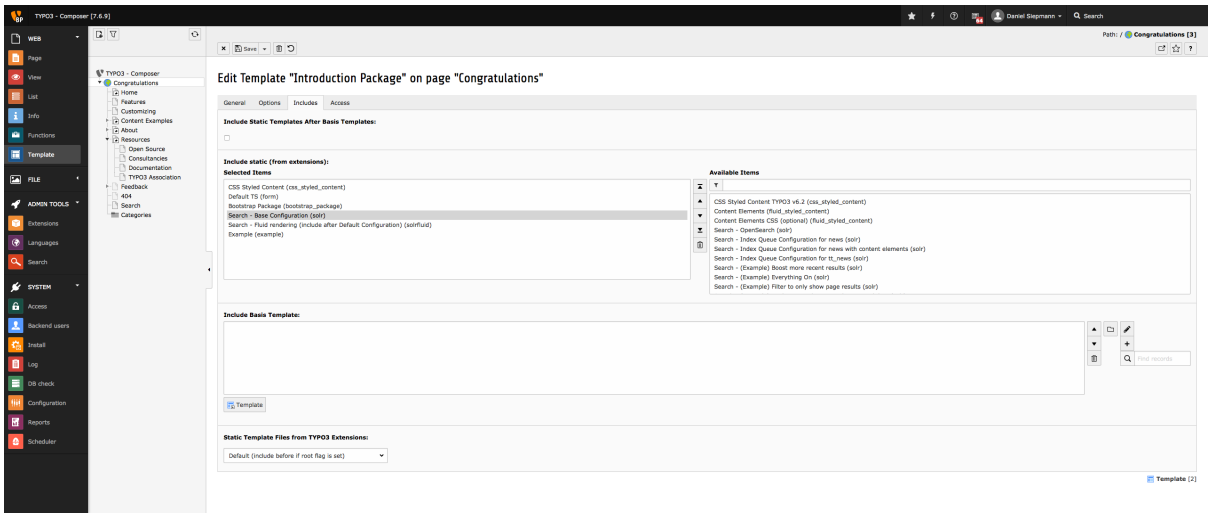

That's all you have to do, now head over to *Configure Extension*.

# Configure Extension

After started-install-extension you need to configure the extension. Only a few steps from below are necessary for *Index the first time*.

## Static TypoScript

The extension already comes with basic configuration that will work for small pages out of the box. For now create, or edit an existing, TypoScript Template record in your page tree and add the provided static TypoScript:




Update the constants to match the current setup:

```
plugin {
    tx_solr {
        solr {
            host = 192.168.99.100
            port = 8983
```

```
        }
    }
}
```

Adjust the host according to where your Solr is reachable, see *Solr*.

### Search Markers

EXT:solr is indexing everything on a page between *<!– TYPO3SEARCH_begin –>* and *<!– TYPO3SEARCH_end –>* to ensure this is the case, check the output of you website and add the markers to your template.

If the markers are missing, you should add them to your template. To increase the quality of the search results the markes should only wrap the relevant content of a page and exclude e.g. menus, because they are same on each page.

The most simple configuration for my page was:

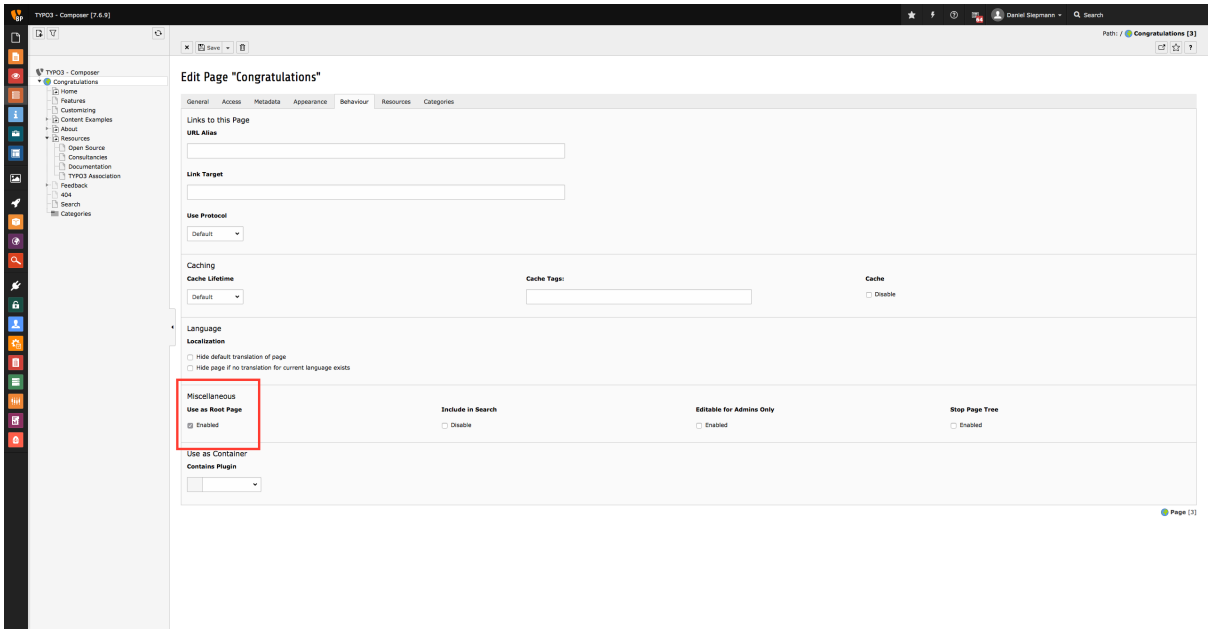
```
page.10 {
    stdWrap.dataWrap = <!--TYPO3SEARCH_begin-->|<!--TYPO3SEARCH_end-->
}
```

### Domain Records and Indexing

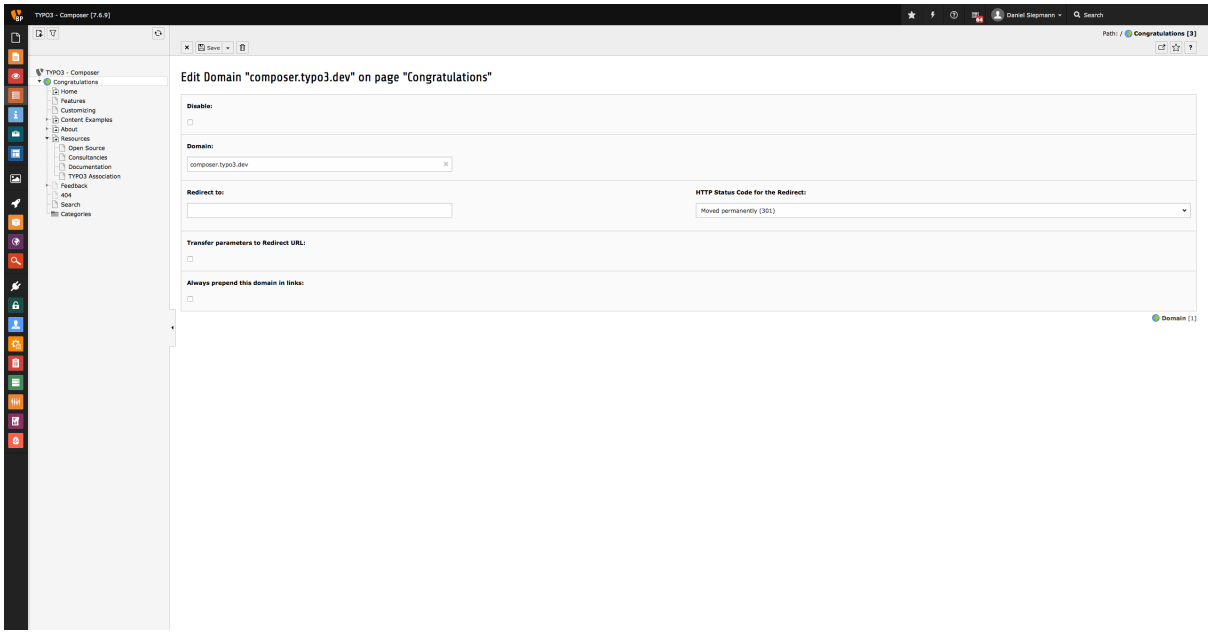
To enable Solr connections, the extension needs a Domain Record and indexing has to be enabled. Therefore enable indexing by setting the following TypoScript:

```
config {
    index_enable = 1
}
```

Also define that your root page is actually a root page:



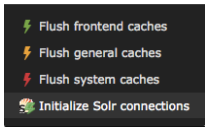

Last but not least, add the domain record to the root page:



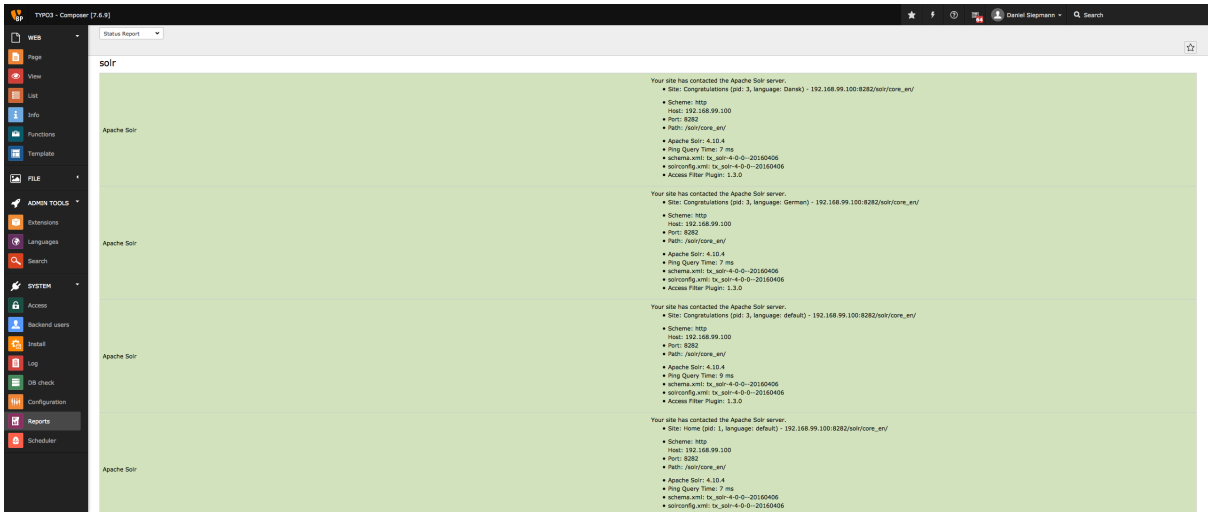

## Initialize Solr Connection

Next, initialize the Solr Connection from TYPO3 and check whether everything works as expected.

To initialize the connection, open the Cache-Menu and start Initialization.



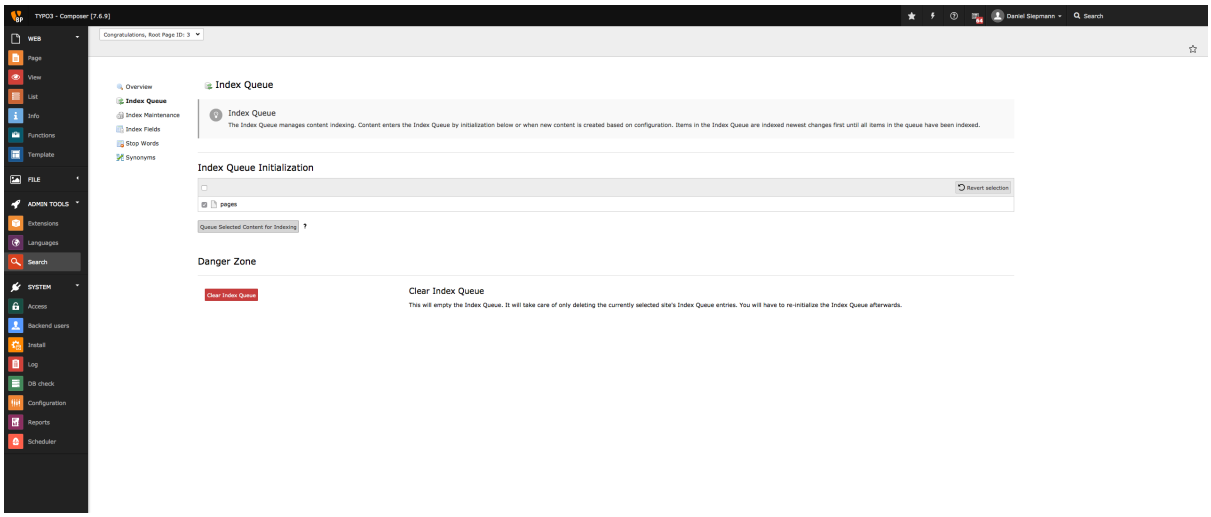

Check whether connections to Solr could be established by opening the *Reports* module and go to *Status Report* view:




That's it, head over to *Index the first time*.

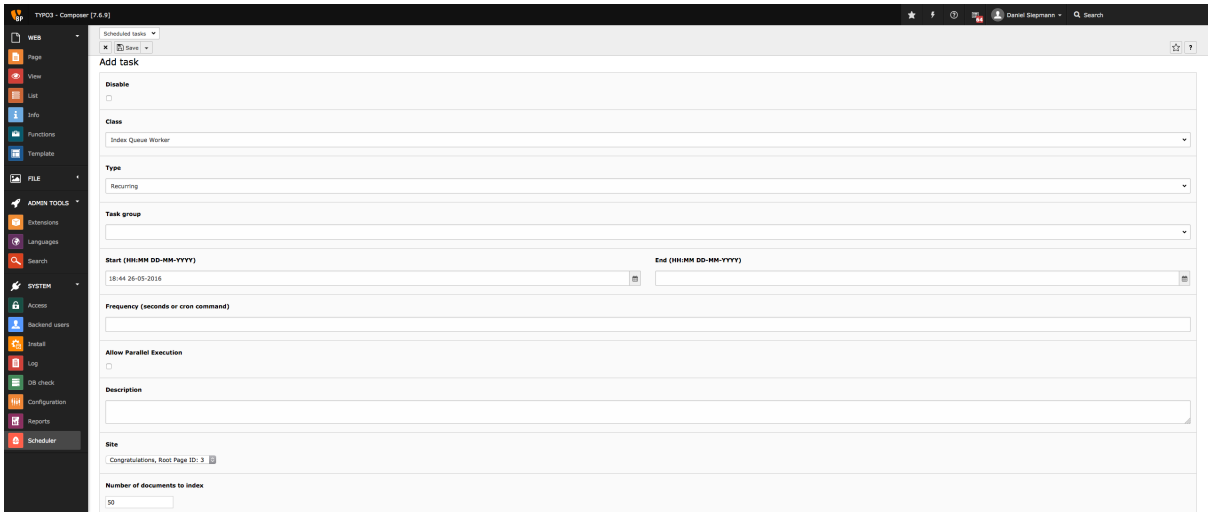
## Index the first time

After everything is setup, you need to index the contents of TYPO3 to enable searching in Solr. To do so open the *Search* module and navigate to the *Index Queue*. Select the contents to index and *Queue Selected Content for Indexing*.

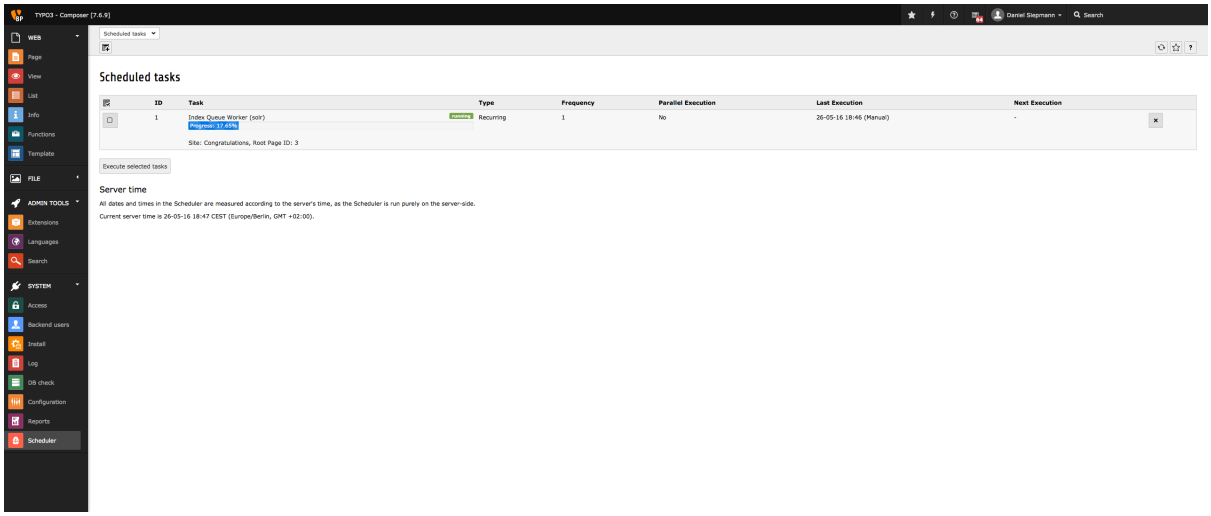



Switch to the *Scheduler* module. If the module is not available, make sure to enable the extension first. It comes bundled with TYPO3 CMS but is not enabled by default.

Create a new scheduler task to run the indexing:




After the task was created, run it manually. The page will indicate a reload but won't reload after the task was run. Therefore you can reload the module to see the progress bar indicating the current progress of indexing:



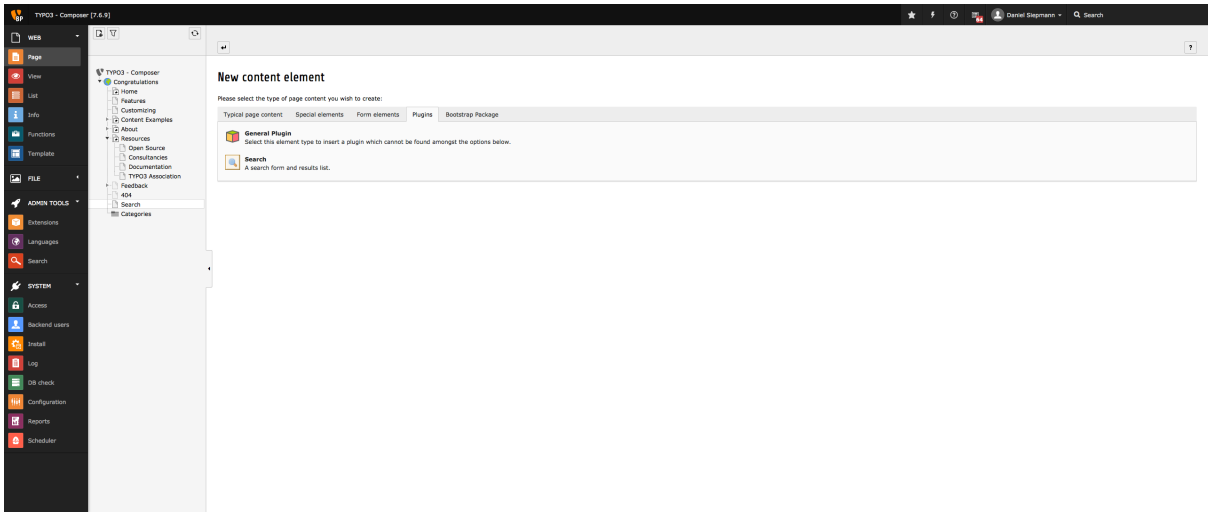

The duration depends on things like the number of records to index and the number of languages configured in your system. Also whether caching is enabled and warmed up.

The extension will now index all records in the queue and send them to Solrs index.

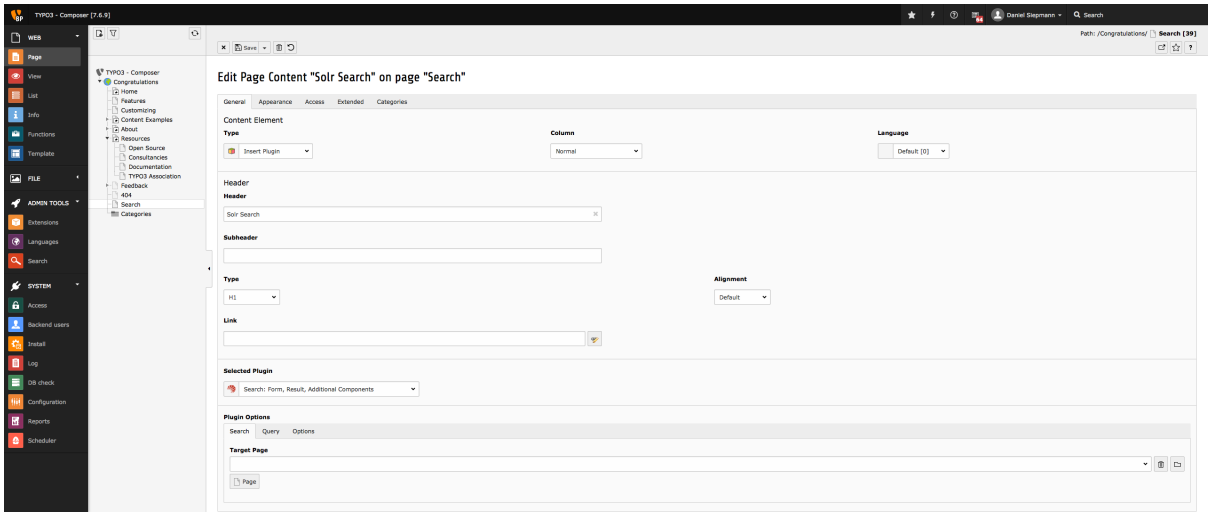
Once you have some records inside the index, you can *Display search and results*.

## Display search and results

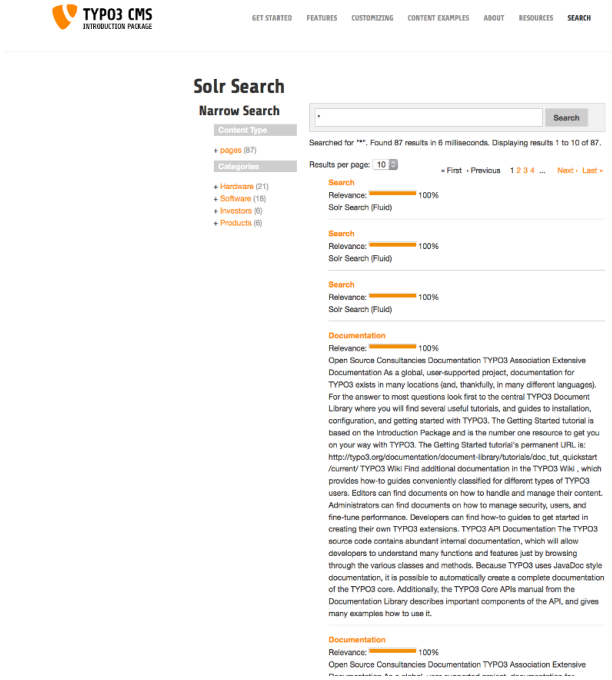
After Solr has some documents inside his index, you can insert the plugin to provide a search with results from Solr. To do so create a new content record of type *Search* on a page:




Select *Search: Form, Result, Additional Components* if not already selected inside the content element:




Open the page and search for *, you should see all currently indexed records from Solr:



That's it. You now have a working TYPO3 Installation with Solr integration. You are able to queue items for indexing, index them and provide an interface for visitors to search the indexed records.

You can now start to adjust the templates according to tx_solr.templateFiles, configure indexing or searching and facets.

CHAPTER

FIVE

# BACKEND

The chapter before gives you a short introduction about how to setup a solr server, the extension and index a few documents into solr. In this chapter we want to go deeper and learn how to write more complex indexing configurations and see what other possibilities the backend of EXT:solr provides.

## ConnectionManager

In EXT:solr all the configuration, including options affecting backend functions, are done in TypoScript. The clear cache menu provides an entry to initialize the Solr connections configured in TypoScript.

### How it works

- Configure the Solr connection in TypoScript under plugin.tx_solr.solr, providing host, port, and path.
- On your site's root page set the flag Use as Root Page on the Behaviour tab.
- Initialize the Solr connections through the clear cache menu

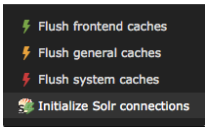



Fig. 5.1: Initialize all solr connections

When initializing the Solr connections the extensions looks for all the pages with the root flag set, generates the TypoScript configuration for that page like in the frontend and reads the Solr connection parameters.

The extension also repeats that process for each language configured on the installation's root (uid = 0). This way you can configure different Solr cores for each language by using regular conditions that change the path of the Solr connection depending on the currently selected language.

Once all the configured Solr connections in the installation are found, they're stored in TYPO3's registry so that they can easily be retrieved without needing to reevaluate the TypoScript configuration every time we connect to Solr.

All that magic happens in class source:Classes/ConnectionManager.php. The connection manager and it's public API actually must be used whenever a Solr connection is needed.

## IndexQueue Configuration

As you already learned you can index pages very easy with EXT:solr and setup a search for pages in seconds. Beside pages, there might be other records in your TYPO3 CMS that you want to have available in your search results.

## Indexing custom records

As a core feature EXT:solr allows you to write custom typoscript configuration to index records from any extension just with configuration. To see how this is working, we open the content of the TypoScript example **"Search - Index Queue Configuration for news"** that can be found in "Configuration/TypoScript/Examples/IndexQueueNews":

```
plugin.tx_solr.index.queue {

    news = 1
    news {
        table = tx_news_domain_model_news

        fields {
            abstract = teaser

            author = author
            authorEmail_stringS = author_email

            title = title

            content = SOLR_CONTENT
            content {
                cObject = COA
                cObject {
                    10 = TEXT
                    10 {
                        field = bodytext
                        noTrimWrap = || |
                    }
                }
            }

            category_stringM = SOLR_RELATION
            category_stringM {
                localField = categories
                multiValue = 1
            }

            keywords = SOLR_MULTIVALUE
            keywords {
                field = keywords
            }

            tags_stringM = SOLR_RELATION
            tags_stringM {
                localField = tags
                multiValue = 1
            }

            url = TEXT
            url {
                typolink.parameter =  {$plugin.tx_news.settings.detailPid}
                typolink.additionalParams = &tx_news_pi1[controller]=News&
                    tx_news_pi1[action]=detail&tx_news_pi1[news]={field:uid}
                typolink.additionalParams.insertData = 1
                typolink.useCacheHash = 1
                typolink.returnLast = url
            }
```

```
        }

        attachments {
            fields = related_files
        }
    }

}

plugin.tx_solr.logging.indexing.queue.news = 1
```

By reading the example above you might recognize the following facts:

- The indexing configuration is done in the TypoScript path 'plugin. tx_solr. index. queue. [configName]' and there can be multiple queue configurations.
- The database table is configured in the property 'plugin.tx_solr.index.queue.[configName].table'. This allows you to have multiple index queue configurations for the same database table. This can be helpful when you have multiple queue configurations for news (e.g. if you have a press & corporate news section on your website).
- The solr fields are configured in 'plugin.tx_solr.index.queue.[configName].fields'. This allows you to flexibly fill any solr field. The combination of dynamic fields (*Appendix - Dynamic Fields*) and the queue configuration allows you to write any kind of data into solr without adapting the solr schema.
- **There are custom TypoScript objects from EXT:solr that are used in the index queue configuration**
  - *SOLR_CONTENT*
  - *SOLR_RELATION*
  - *SOLR_MULTIVALUE*

When the index queue configuration of your custom record is ready, you can check the index queue in the backend module and add the news items to the queue.

### Links in custom records

In the example above *typolink* is used to build a link to the detail view of the news. This is required, because EXT:solr can not know the business logic of the news extension to build a detail link. The typoscript constant "plugin.tx_news.settings.detailPid" is used to configure the target pageId of the news single view. This has two important impacts:

- The constant (*plugin.tx_news.settings.detailPid*) need to point to a valid news single page.
- The page with the news single view, should be configured with *"Include in Search => Disable"* because indexing this page with the normal page indexing without a news id will produce an error page.

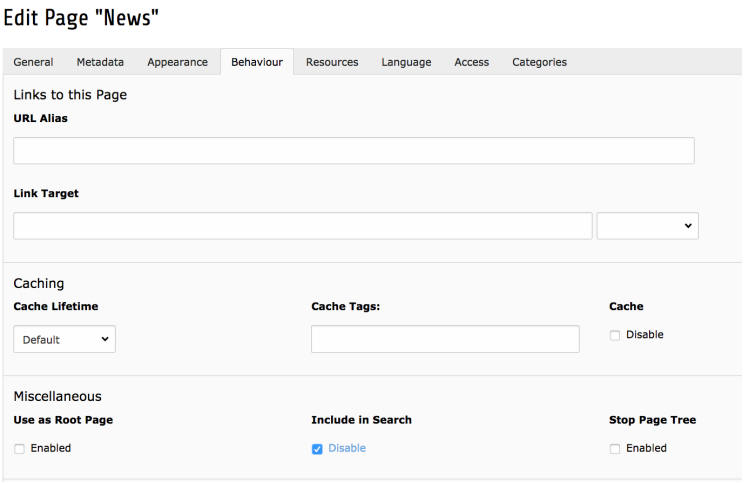
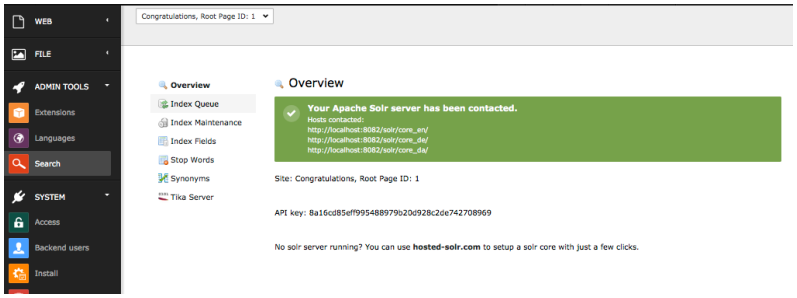



Fig. 5.2: Include in Search - Disable

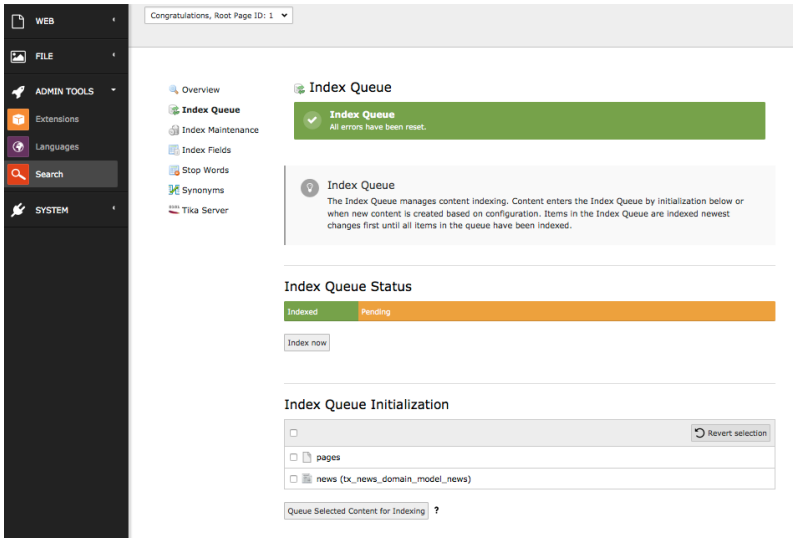
## Backend Module

The solr backend module is available for admins only in the "Admin Tools" section and it helps you to do maintenance tasks and get an overview on the system status:




During the next paragraphs we will go over the modules and explain, what could be done with them.

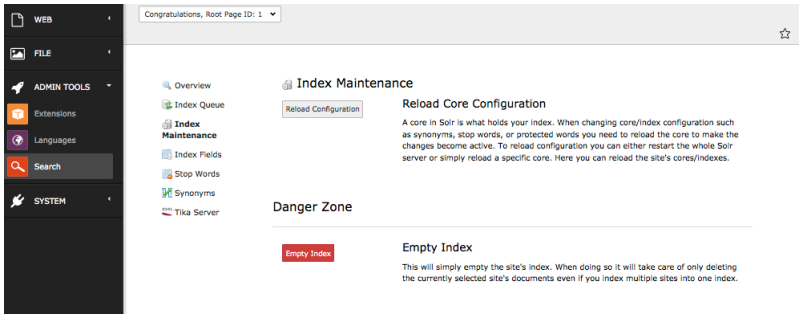
### Index Queue




The **Index Queue** module is the most important module. It allows you to do the following things:

- Select item types and add them for indexing to the indexing queue.
- See the fill state of the indexing queue.
- Check the indexing queue for errors when the indexing of an items failed.
- Start an instant indexing run, directly from the module.
- Clear the indexing queue and re-queue items.

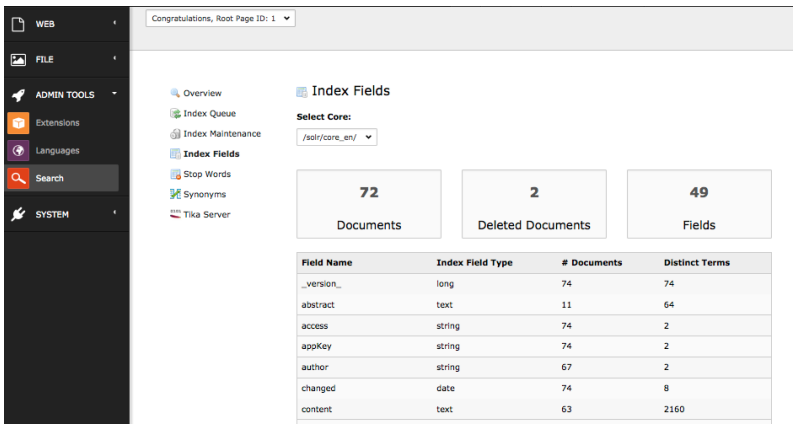
## Index Maintenance




The **Index Maintenance** module allows you, to do the following administrative operations on your solr index:

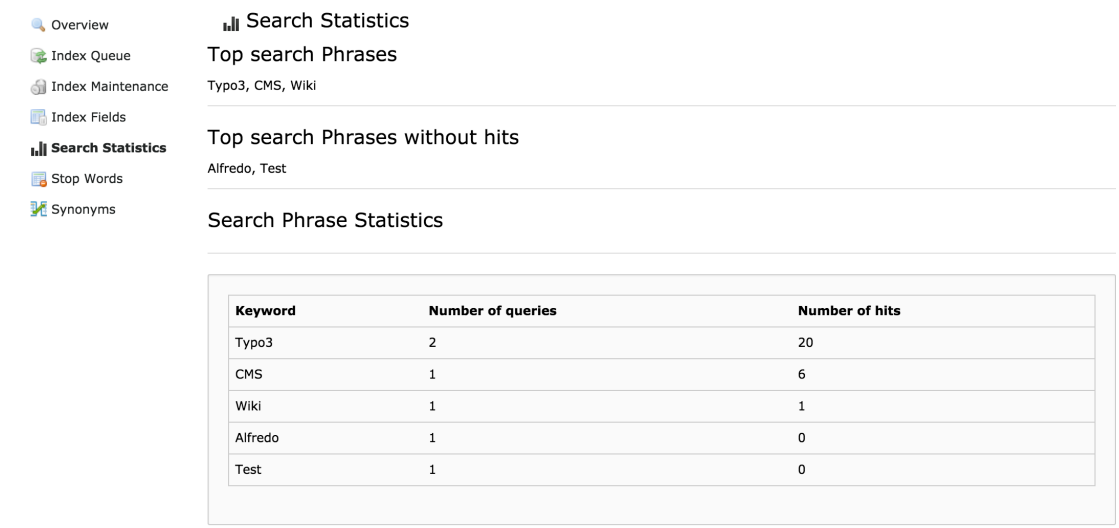
- Reload the solr configuration.
- Empty your solr index. This removes all documents from the index of the current selected site.

## Index Fields



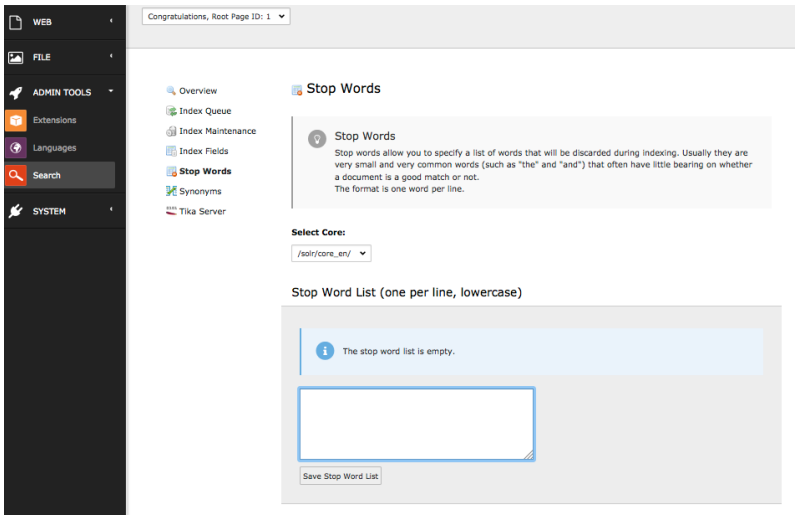

The **Index Fields** module allows you to see, how many documents you have in which solr core and which fields those documents have.

## Search Statistics




The **Search Statistics** module allows you to see Top Search Phrases with and without results. In addition it possible to see a complete listing with hits etc. ranked by Top search keywords.

## Stop Words

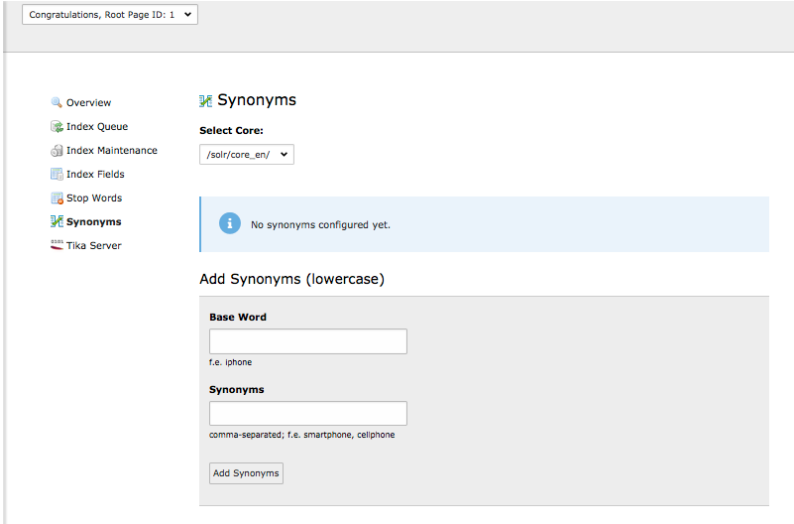



With the stopwords module you can define a list of words that should be excluded from the search.

Common usecases are:

- Very often occurring words like “the”, “and” ... are excluded are filtered out because they are more or less “noize words”.
- You can add words that you want to avoid from indexing.

### Synonyms




With the synonyms module you can allow to find documents by words that do not occurre in the document but have the same meaning:

- E.g. smartphone, cellphone, mobile, mobilephone

**Note:** The word that you want replace with a synonym needs to appear on **both** sides when you want to find it with the term itself later

Example

smartphone => smartphone, cellphone, mobile, mobilephone will match "smartphone, cellphone, mobile, mobilephone", when smartphone is missing on the right side, you will not find the document for smartphone anymore!

## Index Inspector

Beside the own backend module, EXT:solr provides a feature called **"Search Index Inspector"**. This Tool allows you to select a page or sysfolder and check what data is stored for this entity in the solr index.

The **"Search Index Inspector"** can be opened with the TYPO3 Info Module (*"Web > Info > Search Index Inspector"*) it shows the stored data in solr from the page or sysfolder that is selected in the pagetree:

## Scheduler

When you want to index content from TYPO3 into solr automatically EXT:solr ships scheduler tasks, that can be called at a configured time or directly from the backend.

### Index Queue Worker

Changes that are done in the backend by an editor are written into a queue. This queue is processed asynchronously with a scheduler task and each item in the queue is indexed into solr.

The **"Index Queue Worker"** task has the following custom properties:

**Site:** Here you select the solr site, that you want to index with this task instance. **Number of documents to Index:** Here you can configure how many documents you want to index in one run. Depending on the performance of your system and the expected update time of the search you can choose a realistic number here. **Forced webroot:** The scheduler task can be executed in the cli context, because no webserver is used there, TYPO3 is unable to detect your webroot. As assumption we use PATH_site as default here. When you need to configure something else, you

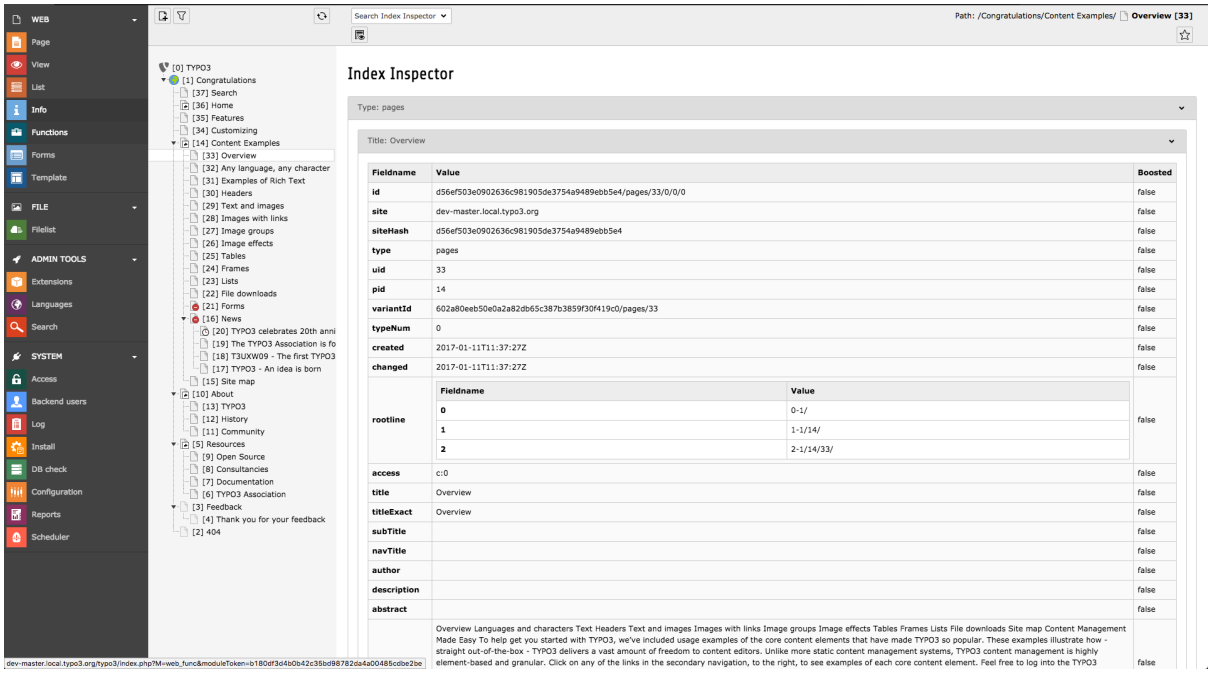



Fig. 5.3: The EXT:solr Search Index Inspector

can do it with this option. You can use the marker ###PATH_site### and ###PATH_typo3### to define relative pathes here, to be independent from the concrete instance.

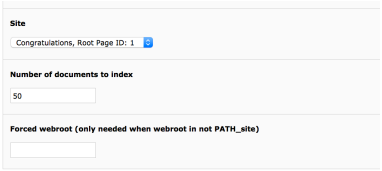



Fig. 5.4: The EXT:solr Index Queue Worker - Scheduler Task

## Force Re-Indexing of a site

This task allows you to force the re-indexing of a site & indexing configuration at a planned time.

The **Force Re-Indexing of a site** task has the following custom properties:

**Site**: Here you select the solr site, that you want to index with this task instance. **Index Queue configurations to re-index**: Here you can limit the set of indexing configurations that should be Re-Indexed.

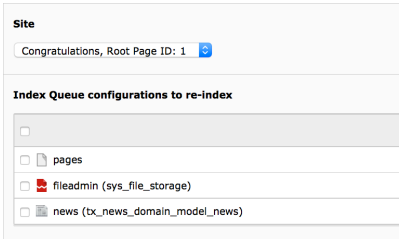



Fig. 5.5: The EXT:solr Force Re-Indexing of a site - Scheduler Task

CHAPTER

SIX

# FRONTEND

This part of the documentation covers the frontend of EXT:solr.

NOTE: Since July 2016 EXT:solrfluid is available as an addon and it will be integrated into EXT:solr later.

The following part of the documentation describes the old way of templating. If you want to create your template in the new way, that will be the standard later please use EXT:solrfluid.

## 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 sense. 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 treated 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:solr 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:


### 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 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:

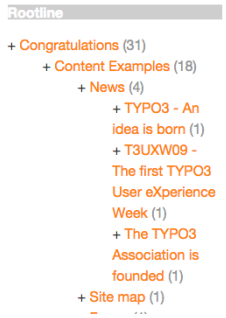

### Hierarchical

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:




**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

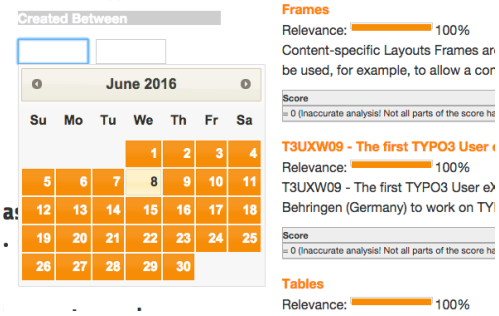
When you want to provide a range filter on a date field in EXT:solr, you can use the type **“dateRange”**.

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
        }
    }
}
```

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




### Numeric Range

Beside dates ranges are also useful 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
            }
        }
    }
}
```

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


## Languages

We recommend to create one solr core per language. The shipped solr example configuration provides a setup for the following languages:

- Arabic
- Armenian
- Basque
- Brazilian portuguese
- Bulgarian
- Burmese
- Catalan
- Chinese
- Czech
- Danish
- Dutch
- English
- Finnish
- French
- Galician
- German
- Greek
- Hindi
- Hungarian
- Indonesian
- Italian
- Japanese
- Khmer
- Korean
- Lao
- Norwegian
- Persian
- Polish
- Portuguese

- Romanian
- Russian
- Spanish
- Swedish
- Thai
- Turkish
- Ukrainian

The configuration of the connection between solr cores and sites is done in typoscript.

The following typoscript snipped shows an example how to configure multiple languages for the introduction package (EN, DE and DA):

```
plugin.tx_solr.solr {
   scheme = http
   port   = 8082
   path   = /solr/core_en/
   host   = localhost
}

[globalVar = GP:L = 1]
plugin.tx_solr.solr.path = /solr/core_de/
[end]

[globalVar = GP:L = 2]
plugin.tx_solr.solr.path = /solr/core_da/
[end]
```

After setting up the languages with typoscript you need to initialize the solr connections with the connection manager (ConnectionManager).

CHAPTER

SEVEN

# LOGGING

When you need to debug something in EXT:solr the extension allows you to write several information to the TYPO3 logging framework.

This can be helpful to analyze e.g.:

- What is happening during the indexing?
- What is going when a search in the frontend is triggered?

## Configure Logging

There are several options that can be configured with TypoScript:

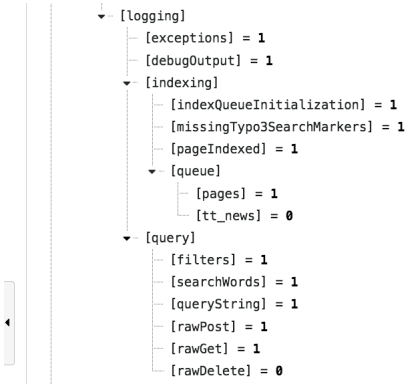



Fig. 7.1: EXT:solr logging settings

You can find a detailed description of all options in the "*tx_solr.logging*" section in the reference.

Note: Make sure that you log on production only what you need, because log data can increase very quickly.

In the next steps we want to see as example how we can use the debug output to see which query was triggered in solr.

The first step is to debug the output for "rawGet":

```
plugin.tx_solr.logging.query.rawGet = 1
```

## Show Logging Output

As written in the introduction, the logging framework functionally is used to write the log entries.

For a quick debugging without any extension there is the possibility to show the debug messages directly:

- For the frontend as formatted output.
- For the backend as TYPO3 console output.

**Note:** You should use this on a development system only and make sure that the devIPmask does not allow other users to see the debug output.

To enable this features you need to:

- Make sure that your ip address is configured to match the devIPMask ($GLOBALS['TYPO3_CONF_VARS']['SYS']['devIPmask'])
- Enable the debug output in the EXT:solr typoscript:

```
plugin.tx_solr.logging.debugOutput = 1
```

You can now see the written log in the frontend:

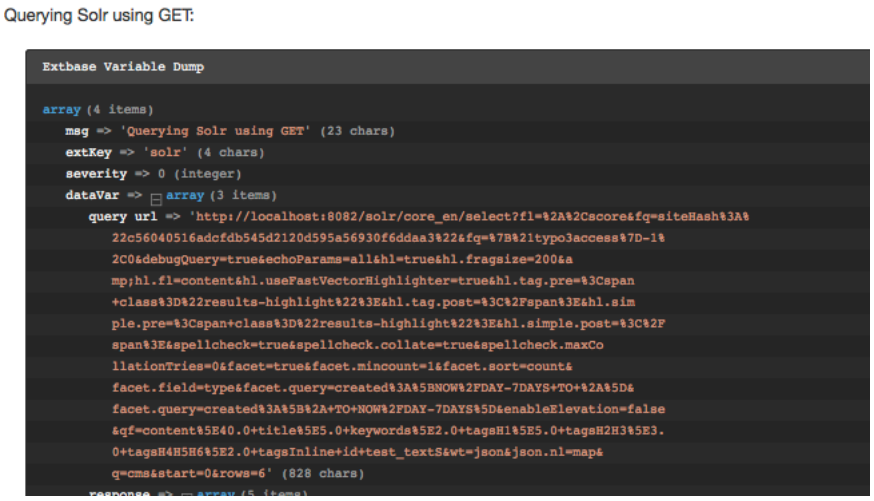



Fig. 7.2: EXT:solr frontend debug output

CHAPTER

EIGHT

# THE APACHE SOLR SERVER

The other chapters focused on the frontend or backend part of the extension. In this part we want to focus on the server part of apache solr.

## Configuration Structure

The configuration can be found in the folder “Resources/Private/Solr”.

This folder contains:

- The folder “configsets”: This folder a set of configuration files that can be deployed into a solr server, as a template.

It contains the “solrconfig.xml” file, the “schema.xml” files for all languages and the accessfilter libary that belongs to this version as a jar file. This configSet needs to be in place on a solr server to create cores that are compatible to the EXT:solr extension.

- The folder “cores”: This folder ships an example “core.properties” file for all languages that are compatible with EXT:solr.

A “core.properties” file references a “configSet” that should be used. The path to the schema that is bound to a core is configured as “schema” relative to the root folder of the “configSet”.

By example a “core.properties” file looks like this:

```
configSet=ext_solr_6_0_0
schema=german/schema.xml
name=core_de
dataDir=../../data/german
```

- The solr.xml file: This file configures solr as required for the used Apache Solr version.

### The EXT:solr configSet

As mentioned before the configSet is one single package, that contains all to customize a plain Apache Solr Server, to an Apache Solr Server, ready for EXT:solr & TYPO3.

The configset of the current release is located in “Resources/Private/Solr/configsets/ext_solr_<release>”.

Inside the configSet you find the following folders:

- conf: This folder contains the solr configuration and all schemata files. There is one directory for each language which ships the schema.xml file for this language.

The schema.xml file contains the language specific adaptions and includes all general schema fields and types with an XInclude statement.

```
<!-- xinclude fields -->
<xi:include href="../general_schema_types.xml" xmlns:xi="http://www.w3.org/2001/
↪XInclude"/>

<!--  xinclude fields-->
<xi:include href="../general_schema_fields.xml" xmlns:xi="http://www.w3.org/2001/
↪XInclude"/>
```

- typo3lib: This folder ships the compiled access filter jar file, that should be used with this EXT:solr version.

The solrconfig (conf/solrconfig.xml) is configured to load all jar files from typo3lib.

## Setup steps

With the extension we ship an installer for development and a docker images that can be used to install solr.

When you want to install solr on your system in another way the following steps are required.

- Install the solr server
- Copy the configsets into the configset folder (by default $SOLR_HOME/server/solr/configsets)
- Make sure that the solr.xml file ($SOLR_HOME/server/solr/solr.xml) is in place an fits to your solr version
- Create an init script that start solr on boottime.
- Secure your solr port from outside.
- Make sure that solr is running with an own user.
- Backup your data folders

*Hint:* Apache Solr ships an install script in newer version that might cover your requirements for production ($SOLR_HOME/bin/install_solr_service.sh). We don't use it in EXT:solr because there are currently problems when using it with ubuntu xenial (16.04)

## Index Maintenance

Solr offers a lot of request handlers to do maintenance tasks.

### Committing pending documents

```
curl http://host:port/solr-path/update -H "Content-Type: text/xml"
    --data-binary '<commit />'
```

### Clearing the index

```
curl http://host:port/solr-path/update -H "Content-Type: text/xml"
    --data-binary '<delete><query>*:*</query></delete>'

curl http://host:port/solr-path/update -H "Content-Type: text/xml"
    --data-binary '<commit />'
```

### Optimizing the index

You should do this every once in a while, f.e. every day. For TYPO3, there is already a scheduler task available for this.

```
curl http://host:port/solr-path/update -H "Content-Type: text/xml"
    --data-binary '<optimize />'
```

### Searching the index from the command line

Parameters:

**q** what to search for. Format: fieldName:fieldValue

**qt** defines the query type, for the command line we recommend "standard", the extension itself uses "dismax"

**fl** comma separated list of fields to return

**rows** number of rows to return

**start** offset from where to return results

```
curl 'http://host:port/path-to-solr/select?q=hello&qt=standard&fl=title,content'
```

### Getting information / statistics about the index

```
curl 'http://host:port/path-to-solr/admin/luke'
```

### Create cores with the core admin api

The CoreAdmin API (https://cwiki.apache.org/confluence/display/solr/CoreAdmin+API) allows you, manipulate the cores in your solr server.

Since we support configSets a core could be generated with the following http call:

```
curl 'http://host:port/path-to-solr/admin/cores?action=CREATE&name=core_de&
↪configSet=ext_solr_6_0_0&schema=german/schema.xml&dataDir=dataDir=../../data/
↪german'
```

## SolrConfig Parameters

There are several parameters in the solrconfig.xml that can be used to tune your solr server. Our solrconfig.xml is designed to ship a reasonable configuration for the most standard use cases.

For use cases with very large indexes or high performance requirements it makes sence to tune those parameters

### indexConfig.useCompoundFile

This value is “true” by default in our configuration. By setting this value to true solr only writes one file for indexes instead of many. This is a little bit slower but more robust to prevent errors with “Too many open files”.

CHAPTER

NINE

# CONFIGURATION REFERENCE

## tx_solr

This section defines general configuration options.



### enabled

**Type** Boolean

**TS Path** plugin.tx_solr.enabled

**Default** 1

**Options** 0, 1

**Since** 1.2

A switch to completely turn on / off EXT:solr. Comes in handy with multi site installations where you want to enable EXT:solr only for certain sites, but still have the extension's configuration at a single place and include that for each site. Just set enabled = 0 for each site's root TS template or use conditions where you do not want EXT:solr.

**Important:** This also influences the connection manager; connections will be registered / detected only for enabled = 1.

### enableDebugMode

**Type** Boolean

**TS Path** plugin.tx_solr.enableDebugMode

**Default** 0

**Options** 0, 1

**Since** 1.0

**See** http://wiki.apache.org/solr/CommonQueryParameters#debugQuery

If enabled, the debugQuery query parameter is added to the Solr queries. Solr will then return additional information explaining the the query, scoring, timing, and other information.

# tx_solr.general

This section defines general settings.



## dateFormat.date

**Type** String

**TS Path** plugin.tx_solr.general.dateFormat.date

**Default** d.m.Y H:i

**Since** 1.0

**See** http://www.php.net/manual/de/function.strftime.php

Defines the format that is used for dates throughout the extension like in view helpers for example. The format uses the *strftime()* php function syntax, please consult the php documentation for available options.

## tx_solr.solr

This section defines the address of the Solr server. As the communication with the Solr server happens over HTTP this is just a simple URL. Each of the URL's components can be defined separately.



### scheme

**Type** String

**TS Path** plugin.tx_solr.solr.scheme

**Default** http

**Options** http, https

**cObject supported** yes

**Since** 1.2 2.0

Allows to set the connection scheme to "https" instead of the default "http".

### host

**Type** String

**TS Path** plugin.tx_solr.solr.host

**Default** localhost

**cObject supported** yes

**Since** 1.0

Sets the host portion of the URL.

### port

**Type** Integer

**TS Path** plugin.tx_solr.solr.port

**Default** 8983

**cObject supported** yes

**Since** 1.0

Sets the port portion of the URL.

### path

**Type** String

**TS Path** plugin.tx_solr.solr.path

**Default** /

**cObject supported** yes

**Since** 1.0

Sets the path portion of the URL. Make sure to have the path end with a slash (/).

### username

**Type** String

**TS Path** plugin.tx_solr.solr.username

**Since** 6.0

**cObject supported** yes

Sets the username required to access the solr server.

### password

**Type** String

**TS Path** plugin.tx_solr.solr.password

**Since** 6.0

**cObject supported** yes

Sets the password required to access the solr server.

### timeout

**Type** Float

**TS Path** plugin.tx_solr.solr.timeout

**Default** 0.0

**Since** 1.0

**cObject supported** no

Can be used to configure a connection timeout.

## tx_solr.templateFiles

This section defines the template files used in the plugins.



### frequentSearches

**Type** String

**TS Path** plugin.tx_solr.templateFiles.frequentSearches

**Default** EXT:solr/Resources/Templates/PiFrequentSearches/frequentsearches.htm

**Since** 2.0

The template used for the frequent searches plugin.

### pagebrowser

**Type** String

**TS Path** plugin.tx_solr.templateFiles.pagebrowser

**Default** EXT:solr/Resources/Templates/PiResults/pagebrowser.htm

**Since** 2.0

Template for the page browser.

### results

**Type** String

**TS Path** plugin.tx_solr.templateFiles.results

**Default** EXT:solr/Resources/Templates/PiResults/results.htm

**Since** 1.0

Template for rendering the results plugin with its different components.

### search

**Type** String

**TS Path** plugin.tx_solr.templateFiles.search

**Default** EXT:solr/Resources/Templates/PiSearch/search.htm

**Since** 2.0

Template for the search plugin that you can use to put a search field on every page of your website. This plugin is cached.

# tx_solr.cssFiles



Here you can configure what CSS files you want to use for different areas of the extension. The section is a definition of key-value pairs where the key is the name of a part of the extension and the value points to the CSS file to be used for styling it.

To prevent loading of a file just set it empty like this:

```
plugin.tx_solr.cssFiles.results =
```

or clear the setting like this:

```
plugin.tx_solr.cssFiles.results >
```

To prevent the extension from loading any default CSS files simple clear the whole cssFiles settings:

```
plugin.tx_solr.cssFiles >
```

## results

**Type** String

**TS Path** plugin.tx_solr.cssFiles.results

**Default** EXT:solr/Resources/Css/PiResults/results.css

**Since** 2.0

Defines the stylesheet to be used for styling the search results page.

## ui

**Type** String

**TS Path** plugin.tx_solr.cssFiles.ui

**Default** EXT:solr/Resources/Css/JQueryUi/jquery-ui.custom.css

**Since** 2.0

Defines the stylesheet to be used for styling the auto suggestions.

## tx_solr.javascriptFiles



Here you can configure what Javascript files you want to use for different areas of the extension. The section is a definition of key-value pairs where the key is the name of a part of the extension and the value points to the Javascript file to be used for styling it.

In general we use jQuery as the default Javascript framework, but through this configuration you are free to use any other framework you like.

To prevent loading of a file just set it empty like this:

```
plugin.tx_solr.javascriptFiles.library =
```

or clear the setting like this:

```
plugin.tx_solr.javascriptFiles.library >
```

To prevent the extension from loading any default Javascript files simple clear the whole javascriptFiles settings:

```
plugin.tx_solr.javascriptFiles >
```

### loadIn

**Type** String

**TS Path** plugin.tx_solr.javascriptFiles.loadIn

**Default** footer

**Options** header, footer, none

**Since** 2.2

Controls where to load the Javascript files, load Javascript in the header, the footer or not at all if you want to take care of it manually.

### library

**Type** String

**TS Path** plugin.tx_solr.javascriptFiles.library

**Default** EXT:solr/Resources/JavaScript/JQuery/jquery.min.js

**Since** 2.0

Defines the general Javascript library to use. By default this is jQuery.

### ui

**Type** String

**TS Path** plugin.tx_solr.javascriptFiles.ui

**Default** EXT:solr/Resources/JavaScript/JQuery/jquery-ui.min.js

**Since** 6.1

Defines the user interface components library to use. By default this is jQuery UI. This is minified and concatenated jQuery UI, which includes following components:

> widget position keycode unique-id widgets/autocomplete widgets/datepicker widgets/menu widgets/-mouse widgets/slider

### ui.datepicker

The date picker in jQuery UI is localizable, the localization labels are defined in separate files which are loaded depending on the current page's language. The localization label files are defined as followed:

```
plugin.tx_solr.javascriptFiles.ui.datepicker {
  de = EXT:solr/Resources/JavaScript/JQuery/ui-i18n/jquery.ui.datepicker-de.js
  fr = EXT:solr/Resources/JavaScript/JQuery/ui-i18n/jquery.ui.datepicker-fr.js
  nl = EXT:solr/Resources/JavaScript/JQuery/ui-i18n/jquery.ui.datepicker-nl.js
}
```

### suggest

**Type** String

**TS Path** plugin.tx_solr.javascriptFiles.suggest

**Default** EXT:solr/Resources/JavaScript/EidSuggest/suggest.js

**Since** 2.0

Defines the suggest Javascript file used in autocomplete / suggest to make the actual request from the website through the TYPO3 eID script to Solr and back.

### faceting.limitExpansion

**Type** String

**TS Path** plugin.tx_solr.javascriptFiles.faceting.limitExpansion

**Default** EXT:solr/Resources/JavaScript/PiResults/results.js

**Since** 2.0

A small script used for the collapse/expand feature of facet option lists.

### faceting.dateRangeHelper

**Type** String

**TS Path** plugin.tx_solr.javascriptFiles.faceting.dateRangeHelper

**Default** EXT:solr/Resources/JavaScript/PiResults/date_range_facet.js

**Since** 2.0

A small glue script used with date range facets and the date picker.

### faceting.numericRangeHelper

**Type** String

**TS Path** plugin.tx_solr.javascriptFiles.faceting.numericRangeHelper

**Default** EXT:solr/Resources/JavaScript/PiResults/numeric_range_facet.js

**Since** 2.0

A small glue script used with numeric range facets and the slider.

# tx_solr.index

This part contains all configuration that is required to setup your indexing configuration. You can use EXT:solr to easily index pages or any kind of records of your TYPO3 CMS.



Allows to prevent frontend indexing of pages when a backend editor is logged in and browsing the website.

## additionalFields (deprecated)

**Type** String, cObject (since 1.1)

**TS Path** plugin.tx_solr.index.additionalFields

**Since** 1.0

**Deprecated** 2.0

A mapping of Solr field names to additional string values to be indexed with page documents. Use dynamic fields to index additional data, this way you don't have to modify the schema.xml

Example:

```
plugin.tx_solr.index.additionalFields {
  myFirstAdditionalField_stringS = some string

  mySecondAdditionalField_stringS = TEXT
  mySecondAdditionalField_stringS {
    value = some other value that can be constructed using any TypoScript cObject
    case = upper
    // more processing here as needed
  }
}
```

Since version 1.1 you can use cObjects to generate the value for the field. The only thing to observe is that you generate strings. Other values may work, but haven't been tested yet.

Deprecated since 2.0, please use the Index Queue indexing configurations instead as it allows you to define more precisely for which types of documents you want which fields to be indexed.

### fieldProcessingInstructions

**Type** cObject

**TS Path** plugin.tx_solr.index.fieldProcessingInstructions

**Since** 1.2 2.0

**Options** timestampToIsoDate, uppercase, pathToHierarchy (2.5-dkd), pageUidToHierarchy (2.5-dkd)

Assigns processing instructions to Solr fields during indexing (Syntax: Solr index field = processing instruction name). Currently it is not possible to extend / add own processing instructions. Before documents are sent to the Solr server they are processed by the field processor service. Currently you can make a filed's value all uppercase, convert a UNIX timestamp to an ISO date, or transform a path into a hierarchy for hierarchical facets (2.0 only). Currently you can use only one processing instruction at a time.

Example:

```
fieldProcessingInstructions {
    changed = timestampToIsoDate
    created = timestampToIsoDate
    endtime = timestampToIsoDate
}
```

### queue

The Index Queue is a powerful feature introduced with version 2.0. It allows you to easily index any table in your TYPO3 installation by defining a mapping of SolrFieldName = DatabaseTableFieldNameOrContentObject. The table must be configured / described in TCA, though. To index other, external data sources you might want to check out Solr's Data Import Handler (DIH).

The Index Queue comes preconfigured to index pages (enabled by default) and an example configuration for tt_news (provided as a separate TypoScript template).

**Type** Array

**TS Path** plugin.tx_solr.index.queue

**Since** 2.0

**Default** pages

Defines a set of table indexing configurations. By convention the name of the indexing configuration also represents the table name. You can name the indexing configuration differently though by explicitly defining the table as a parameter within the indexing configuration. That's useful when indexing records from one table with different configuration - different single view pages / URLs for example.

Example:

```
// enables indexing of tt_news records
plugin.tx_solr.index.queue.news = 1
plugin.tx_solr.index.queue.news.fields {
    abstract = short
    author = author
    description = short
    title = title

    // the special SOLR_CONTENT content object cleans HTML and RTE fields
    content = SOLR_CONTENT
    content {
        field = bodytext
    }

    // the special SOLR_RELATION content object resolves relations
    category_stringM = SOLR_RELATION
    category_stringM {
        localField = category
        multiValue = 1
    }

    // the special SOLR_MULTIVALUE content object allows to index multivalue fields
    keywords = SOLR_MULTIVALUE
    keywords {
        field = keywords
    }

    // build the URL through typolink, make sure to use returnLast = url
    url = TEXT
    url {
        typolink.parameter = {$plugin.tt_news.singlePid}
        typolink.additionalParams = &tx_ttnews[tt_news]={field:uid}
        typolink.additionalParams.insertData = 1
        typolink.returnLast = url
        typolink.useCacheHash = 1
```

```
    }

    sortAuthor_stringS = author
    sortTitle_stringS  = title
}
```

## queue.[indexConfig]

**Type** Boolean, Array

**TS Path** plugin.tx_solr.index.queue.[indexConfig]

**Since** 2.0

**Default** pages

An indexing configuration defines several parameters about how to index records of a table. By default the name of the indexing configuration is also the name of the table to index.

By setting *plugin.tx_solr.index.queue.[indexConfig] = 1 or 0* you can en- / disable an indexing configuration.

**Note**: you could add *L={field:__solr_index_language}* in the additionalParams of the typolink to link to the correct language version (this was removed from the example above to simplify the example)

## queue.[indexConfig].additionalWhereClause

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].additionalWhereClause

**Since** 2.0

A WHERE clause that is used when initializing the Index Queue, limiting what goes into the Queue. Use this to limit records by page ID or the like.

```
// only index standard and mount pages, enabled for search
plugin.tx_solr.index.queue.pages.additionalWhereClause = doktype IN(1, 7)
```

## queue.[indexConfig].initialPagesAdditionalWhereClause

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].initialPagesAdditionalWhereClause

**Since** 6.1

A WHERE clause that is used when initializing the Index Queue, limiting pages that goes into the Queue. This filter is applied **prior** to the plugin.tx_solr.index.queue.[indexConfig].additionalWhereClause filter and hence provides an even stronger filter mechanism - since it can be used to filter away page ID's that shouldn't be processed at all. |

```
// Filter away pages that are "spacer" and have no_search, hidden and nav_hide set␣
↪to zero
plugin.tx_solr.index.queue.pages.initialPagesAdditionalWhereClause = doktype <>␣
↪199 AND no_search = 0 AND hidden = 0 AND nav_hide = 0
```

### queue.[indexConfig].additionalPageIds

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].additionalPageIds

**Since** 2.0

Defines additional pages to take into account when indexing records for example. Especially useful for indexing DAM records or if you have your news outside your site root in a shared folder to use for multiple sites.

Additional page IDs can be provided as comma-separated list.

### queue.[indexConfig].table

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].table

**Since** 2.0

Sometimes you may want to index records from a table with different configurations, f.e., to generate different single view URLs for tt_news records depending on their category or storage page ID. In these cases you can use a distinct name for the configuration and define the table explicitly.

```
plugin.tx_solr.index.queue.generalNews {
  table = tt_news
  fields.url = URL for the general news
  // more field configurations here ...
}

// extends the general news configuration
plugin.tx_solr.index.queue.pressNews < plugin.tx_solr.index.queue.generalNews
plugin.tx_solr.index.queue.pressNews {
  fields.url = overwriting URL for the press announcements
  // may overwrite or unset more settings from the general configuration
}

// completely different configuration
plugin.tx_solr.index.queue.productNews {
  table = tt_news
  fields.url = URL for the product news
}
```

### queue.[indexConfig].initialization

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].initialization

**Since** 2.0

When initializing the Index Queue through the search backend module the queue tries to determine what records need to be indexed. Usually the default initializer will be enough for this purpose, but this option allows to define a class that will be used to initialize and add records to the Index Queue in special ways.

The extension uses this option for initializing the pages and more specifically to resolve Mount Page trees so they can be indexed too, although only being virtual pages.

### queue.[indexConfig].indexer

**Type** String, Array

**TS Path** plugin.tx_solr.index.queue.[indexConfig].indexer

**Since** 2.0

When configuring tables to index a default indexer is used that comes with the extensions. The default indexer resolves the Solr field to database table field mapping as configured. However, in some cases you may reach the limits of TypoScript, when this happens you can configure a specialized indexer using this setting.

The indexer class is loaded using TYPO3's auto loading mechanism, so make sure your class is registered properly. The indexer must extend tx_solr_indexqueue_Indexer.

Example, pages use a specialized indexer:

```
plugin.tx_solr.index.queue.pages {
    indexer = tx_solr_indexqueue_PageIndexer
    indexer {
        // add options for the indexer here
    }
}
```

Within the indexer configuration you can also define options for the specialized indexer. These are then available within the indexer class in $this->options.

Example, the TypoScript settings are available in PHP:

TypoScript:

```
plugin.tx_solr.index.queue.myIndexingConfiguration {
    indexer = tx_myextension_indexqueue_MyIndexer
    indexer {
        someOption = x
        someOtherOption = y
    }
}
```

PHP:

```
namespace MyVendor\Namespace;

use ApacheSolrForTypo3\Solr\IndexQueue\Indexer;

class MyIndexer extends Indexer {
  public function index(tx_solr_indexqueue_Item $item) {
    if ($this->options['someOption']) {
      // ...
    }
  }
}
```

### queue.[indexConfig].indexingPriority

**Type** Integer

**TS Path** plugin.tx_solr.index.queue.[indexConfig].indexingPriority

**Since** 2.2

**Default** 0

Allows to define the order in which Index Queue items of different kinds are indexed. Items with higher priority are indexed first.

### queue.[indexConfig].fields

**Type** Array

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields

**Since** 2.0

Mapping of Solr field names on the left side to database table field names or content objects on the right side. You must at least provide the title, content, and url fields. TYPO3 system fields like uid, pid, crdate, tstamp and so on are added automatically by the indexer depending on the TCA information of a table.

Example:

```
plugin.tx_solr.index.queue.[indexConfig].fields {
  content = bodytext
  title = title
  url = TEXT
  url {
    typolink.parameter = {$plugin.tx_extensionkey.singlePid}
    typolink.additionalParams = &tx_extenionkey[record]={field:uid}
    typolink.additionalParams.insertData = 1
    typolink.returnLast = url
  }
}
```

### queue.[indexConfig].attachments.fields

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].attachments.fields

**Since** 2.5-dkd

Comma-separated list of fields that hold files. Using this setting allows to tell the file indexer in which fields to look for files to index from records.

Example:

```
plugin.tx_solr.index.queue.tt_news.attachments.fields = news_files
```

### queue.[indexConfig].recursiveUpdateFields

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].recursiveUpdateFields

**Since** 6.1

**Default** Empty

Allows to define a list of additional fields from the pages table that will trigger a recursive update.

```
plugin.tx_solr.index.queue.pages.recursiveUpdateFields = title
```

The example above will trigger a recursive update of pages if the title is changed.

Please note that the following columns should NOT be configured as recursive update fields: "hidden" and "extendToSubpages". These fields are handled by EXT:solr already internally and thus they will have not effect.

### queue.pages.excludeContentByClass

**Type** String

**TS Path** plugin.tx_solr.index.queue.pages.excludeContentByClass

**Since** 4.0

Can be used for page indexing to exclude a certain css class to be indexed.

Example:

```
plugin.tx_solr.index.queue.pages.excludeContentByClass = removeme
```

The example above will remove the content of items in the page that have the css class "removeme".

### queue.pages.allowedPageTypes

**Type** List of Integers

**TS Path** plugin.tx_solr.index.queue.pages.allowedPageTypes

**Since** 3.0

**Default** 1,7

Allows to set the pages types allowed to be indexed.

Even if you have multiple queues for pages, e.g. via different `additionalWhereClause`'s, you have to set this value to allow further `doktype`'s. Restrict the pages to be indexed by each queue via `additionalWhereClause`.

### queue.pages.indexer.authorization.username

**Type** String

**TS Path** plugin.tx_solr.index.queue.pages.indexer.authorization.username

**Since** 2.0

Specifies the username to use when indexing pages protected by htaccess.

### queue.pages.indexer.authorization.password

**Type** String

**TS Path** plugin.tx_solr.index.queue.pages.indexer.authorization.password

**Since** 2.0

Specifies the password to use when indexing pages protected by htaccess.

### queue.pages.indexer.frontendDataHelper.scheme

**Type** String

**TS Path** plugin.tx_solr.index.queue.pages.indexer.frontendDataHelper.scheme

**Since** 2.0

Specifies the scheme to use when indexing pages.

### queue.pages.indexer.frontendDataHelper.host

**Type** String

**TS Path** plugin.tx_solr.index.queue.pages.indexer.frontendDataHelper.host

**Since** 2.0

Specifies the host to use when indexing pages.

### queue.pages.indexer.frontendDataHelper.path

**Type** String

**TS Path** plugin.tx_solr.index.queue.pages.indexer.frontendDataHelper.path

**Since** 2.0

Specifies the path to use when indexing pages.

## Indexing Helpers

To make life even easier the Index Queue provides some indexing helpers. These helpers are content objects that perform cleanup tasks or content transformations.

### SOLR_CONTENT

**Since** 2.0

Cleans a database field in a way so that it can be used to fill a Solr document's content field. It removes HTML markup, Javascript and invalid utf-8 chracters.

The helper supports stdWrap on its configuration root.

Example:

```
content = SOLR_CONTENT
content {
    field = bodytext
}
```

**Parameters:**

**value**

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields.[fieldName].value

**Since** 2.0

Defines the content to clean up. In this case the value would be hard-coded.

### SOLR_MULTIVALUE

**Since** 2.0

Turns comma separated strings into an array to be used in a multi value field of an Solr document.

The helper supports stdWrap on its configuration root.

Example:

```
keywords = SOLR_MULTIVALUE
keywords {
    field = tags
    separator = ,
    removeEmptyValues = 1
}
```

**Parameters:**

**value**

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields.[fieldName].value

**Since** 2.0

Defines the content to clean up. In this case the value would be hard-coded.

**separator**

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields.[fieldName].separator

**Since** 2.0

**Default** ,

The separator by which to split the content.

**removeEmptyValues**

**Type** Boolean

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields.[fieldName].removeEmptyValues

**Since** 2.0

**Options** 0,1

**Default** 1

The helper will clean the resulting array from empty values by default. If, for some reason, you want to keep empty values just set this to 0.

**removeDuplicateValues**

**Type** Boolean

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields.[fieldName].removeDuplicateValues

**Since** 2.9

**Options** 0,1

**Default** 0

Cleans the result from duplicate values.

#### SOLR_RELATION

**Since** 2.0

Resolves relations between tables.

Example:

```
category_stringM = SOLR_RELATION
category_stringM {
    localField = category
    multiValue = 1
}
```

**Parameters:**

**localField**

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields.[fieldName].localField

**Since** 2.0

**Required** yes

The current record's field name to use to resolve the relation to the foreign table.

**foreignLabelField**

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields.[fieldName].foreignLabelField

**Since** 2.0

Usually the label field to retrieve from the related records is determined automatically using TCA, using this option the desired field can be specified explicitly. To specify the label field for recursive relations, the field names can be separated by a dot, e.g. for a category hierarchy to get the name of the parent category one could use "parent.name" (since version:2.9).

**multiValue**

**Type** Boolean

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields.[fieldName].multiValue

**Since** 2.0

**Options** 0,1

**Default** 0

Whether to return related records suitable for a multi value field. If this is disabled the related values will be concatenated using the following singleValueGlue.

**singleValueGlue**

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields.[fieldName].singleValueGlue

**Since** 2.0

**Default** ", "

When not using multiValue, the related records need to be concatenated using a glue string, by default this is ", " (comma followed by space). Using this option a custom glue can be specified. The custom value must be wrapped by pipe (|) characters to be able to have leading or trailing spaces.

**relationTableSortingField**

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields.[fieldName].relationTableSortingField

**Since** 2.2

Field in an mm relation table to sort by, usually "sorting".

**enableRecursiveValueResolution**

**Type** Boolean

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields.[fieldName].enableRecursiveValueResolution

**Since** 2.9

**Options** 0,1

**Default** 1

If the specified remote table's label field is a relation to another table, the value will be resolve by following the relation recursively.

**removeEmptyValues**

**Type** Boolean

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields.[fieldName].removeEmptyValues

**Since** 2.9

**Options** 0,1

**Default** 1

Removes empty values when resolving relations.

**removeDuplicateValues**

**Type** Boolean

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields.[fieldName].removeDuplicateValues

**Since** 2.9

**Options** 0,1

**Default** 0

Removes duplicate values

**additionalWhereClause**

**Type** String

**TS Path** plugin.tx_solr.index.queue.[indexConfig].fields.[fieldName].additionalWhereClause

**Since** 5.0

Where clause that could be used to limit the related items to a subset that matches this where clause

Example:

```
category_stringM = SOLR_RELATION
category_stringM {
    localField = tags
    multiValue = 1
    additionalWhereClause = pid=2
}
```

## enableCommits

**Type** Boolean

**TS Path** plugin.tx_solr.index.enableCommits

**Since** 6.1

**Default** true

This setting controls whether ext-solr will implicitly cause solr commits as part of its operation.

If this settings is set to false, you need to ensure that something else will periodically call commits. The solr daemons AutoCommit feature would be a natural choice.

This feature is mainly useful, when you have many installations in the same solr core.

**Note**: Calling some APIs may still cause commits, but these can always be explicitly disabled.

## tx_solr.search

The search section, you probably already guessed it, provides configuration options for the all things related to actually searching the index, setting query parameters, formatting and processing result documents and the result listing.

### targetPage

**Type** Integer

**TS Path** plugin.tx_solr.search.targetPage

**Default** 0

**Since** 1.0

Sets the target page ID for links. If it is empty or 0, the current page ID will be used.

Note: This setting can be overwritten by the plugins flexform.

### initializeWithEmptyQuery

**Type** Boolean

**TS Path** plugin.tx_solr.search.initializeWithEmptyQuery

**Default** 0

**Options** 0,1

**Since** 1.0

If enabled, the results plugin (pi_results) 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 showResultsOfInitialEmptyQuery below.

### showResultsOfInitialEmptyQuery

**Type** Boolean

**TS Path** plugin.tx_solr.search.showResultsOfInitialEmptyQuery

**Default** 0

**Options** 0,1

**Since** 1.0

Requires initializeWithEmptyQuery (above) to be enabled to have any effect. If enabled together with initializeWithEmptyQuery 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.

### keepExistingParametersForNewSearches

**Type** Boolean

**TS Path** plugin.tx_solr.search.keepExistingParametersForNewSearches

**Default** 0

**Options** 0,1

**Since** 2.0

When doing a new search, existing parameters like filters will be carried over to the new search. This is useful for a scenario where you want to list all available documents first, then allow the user to filter the documents using facets and finally allow him to specify a search term to refine the search.

## query

The query sub-section defines a few query parameters for the query that will be sent to the Solr server later on. Some query parameters are also generated and set by the extension itself, f.e. when using facets.

### query.allowEmptyQuery

**Type** Boolean

**TS Path** plugin.tx_solr.search.query.allowEmptyQuery

**Default** 0

**Options** 0,1

**Since** 1.4

If enabled, empty queries are allowed.

### query.allowedSites

**Type** String

**TS Path** plugin.tx_solr.search.query.allowedSites

**Since** 2.2

**Default** __solr_current_site

When indexing documents (pages, records, files, ...) into the Solr index, the solr extension adds a "siteHash". The siteHash is used to allow indexing multiple sites into one index and still have each site only find its own documents. This is achieved by adding a filter on the siteHash.

Sometimes though, you want to search across multiple domains, then the siteHash is a blocker. Using the allowedSites setting you can set a comma-separated list of domains who's documents are allowed to be included in the current domain's search results. The default value is **__solr_current_site** which is a magic string/variable that is replaced with the current site's domain when querying the Solr server.

**Since** 3.0

Version 3.0 introduced a couple more magic keywords that get replaced:

- **__current_site** same as **__solr_current_site**
- **__all** Adds all domains as allowed sites
- * (asterisk character) Everything is allowed as siteHash (same as no siteHash check). This option should only be used when you need a search across multiple system and you know the impact of turning of the siteHash check.

### query.getParameter

**Type** String

**TS Path** plugin.tx_solr.search.query.getParameter

**Since** 2.2

**Default** tx_solr|q

The GET query parameter name used in URLs. Useful for cases f.e. when a website tracking tool does not support the default array GET parameters.

The option expects a string, you can also define an array in the form of arrayName|arrayKey.

Example:

```
plugin.tx_solr.search.query.getParameter = q
```

#### query.queryFields (query.fields)

**Type** String

**TS Path** plugin.tx_solr.search.query.queryFields

**Since** 1.0

**Default** content^40.0, title^5.0, keywords^2.0, tagsH1^5.0, tagsH2H3^3.0, tagsH4H5H6^2.0, tagsInline^1.0

**Note** query.fields has been renamed to query.queryFields in version 3.0

Defines what fields to search in the index. Fields are defined as a comma separated list. Each field can be given a boost by appending the boost value separated by the ^ character, that's Lucene query language. The boost value itself is a float value, pay attention to using a dot as the separator for the fractions. Use this option to add more fields to search.

The boost take influence on what score a document gets when searching and thus how documents are ranked and listed in the search results. A higher score will move documents up in the result listing. The boost is a multiplier for the original score value of a document for a search term.

By default if a search term is found in the content field the documents gets scored / ranked higher as if a term was found in the title or keywords field. Although the default should provide a good setting, you can play around with the boost values to find the best ranking for your content.

#### query.returnFields

**Type** String

**TS Path** plugin.tx_solr.search.query.returnFields

**Since** 3.0

**Default** *, score

Limits the fields returned in the result documents, by default returns all field plus the virtual score field.

#### query.minimumMatch

**Type** String

**TS Path** plugin.tx_solr.search.query.minimumMatch

**Since** 1.2, 2.0

**Default** (empty)

**See** Apache Solr Wiki mm / Minimum Should Match

Sets the minimum match mm query parameter. By default the mm query parameter is set in solrconfig.xml as 2<-35%. This means that for queries with less than three words they all must match the searched fields of a document. For queries with three or more words at least 65% of them must match rounded up.

Please consult the link to the Solr wiki for a more detailed description of the mm syntax.

#### query.boostFunction

**Type** String

**TS Path** plugin.tx_solr.search.query.boostFunction

**Since** 1.2, 2.0

**Default** (empty)

**See** Apache Solr Wiki / TheDisMaxQueryParser BoostFunction

**See** Apache Solr Wiki / Function Queries

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

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')*

#### query.boostQuery

**Type** String

**TS Path** plugin.tx_solr.search.query.boostQuery

**Since** 2.0

**Default** (empty)

**See** Apache Solr Wiki / TheDisMaxQueryParser BoostQuery

Sets the boost function **bq** query parameter.

Allows to further manipulate the score of a document by using Lucene syntax queries. A common use case for boost queries is to rank documents of a specific type higher than others.

Please consult the link to the Solr wiki for a more detailed description of boost functions.

Example (boosts tt_news documents by factor 10):

```
plugin.tx_solr.search.query.boostQuery = (type:tt_news)\^10
```

#### query.filter

**Type** Array

**TS Path** plugin.tx_solr.search.query.filter

**Since** 1.0

**See** Lucene Documentation / Query Parser Syntax

Allows to predefine filters to apply to a search query. You can add multiple filters through a name to Lucene filter mapping. The filters support stdWrap.

Example:

```
plugin.tx_solr.search.query.filter {
    pagesOnly = type:pages
    johnsPages = author:John
    badKeywords = {foo}
    badKeywords.wrap = -keywords:|
    badKeywords.data = GP:q
}
```

#### query.filter.__pageSections

**Type** comma-separated list of page IDs

**TS Path** plugin.tx_solr.search.query.filter.__pageSections

**Since** 3.0

This is a magic/reserved filter (thus the double underscore). It limits the query and the results to certain branches/-sections of the page tree. Multiple starting points can be provided as a comma-separated list of page IDs.

#### query.sortBy

**Type** String

**TS Path** plugin.tx_solr.search.query.sortBy

**Since** 1.0

Allows to set a custom sorting for the query. By default Solr will sort by releavance, using this setting you can sort by any sortable field.

Needs a Solr field name followed by asc for ascending order or desc for descending.

Example:

```
plugin.tx_solr.search.query.sortBy = title asc
```

### results

#### results.resultsHighlighting

**Type** Boolean

**TS Path** plugin.tx_solr.search.results.resultsHighlighting

**Since** 1.0

**Default** 0

**See** Apache Solr Wiki / FastVectorHighlighter

En-/disables search term highlighting on the results page.

Note: The FastVectorHighlighter is used by default (Since 4.0) if fragmentSize is set to at least 18 (this is required by the FastVectorHighlighter to work).

#### results.resultsHighlighting.highlightFields

**Type** String

**TS Path** plugin.tx_solr.search.results.resultsHighlighting.highlightFields

**Since** 1.0

**Default** content

A comma-separated list of fields to highlight.

Note: The highlighting in solr (based on FastVectorHighlighter requires a field datatype with **termVectors=on**, **termPositions=on** and **termOffsets=on** which is the case for the content field). If you add other fields here, make sure that you are using a datatype where this is configured.

#### results.resultsHighlighting.fragmentSize

**Type** Integer

**TS Path** plugin.tx_solr.search.results.resultsHighlighting.fragmentSize

**Since** 1.0

**Default** 200

The size, in characters, of fragments to consider for highlighting. “0” indicates that the whole field value should be used (no fragmenting).

#### results.resultsHighlighting.fragmentSeparator

**Type** String

**TS Path** plugin.tx_solr.search.results.resultsHighlighting.fragmentSeparator

**Since** 3.0

**Default** [...]

When highlighting is activated Solr highlights the fields configured in highlightFields and can return multiple fragments of fragmentSize around the highlighted search word. These fragments are used as teasers in the results list. fragmentSeparator allows to configure the glue string between those fragments.

#### results.resultsHighlighting.wrap

**Type** String

**TS Path** plugin.tx_solr.search.results.resultsHighlighting.wrap

**Since** 1.0

**Default** |

The wrap for search terms to highlight.

#### results.siteHighlighting

**Type** Boolean

**TS Path** plugin.tx_solr.search.results.siteHighlighting

**Since** 2.0

**Default** 0

Activates TYPO3’s highlighting of search words on the actual pages. The words a user searched for will be wrapped with a span and CSS class csc-sword Highlighting can be styled using the CSS class csc-sword, you need to add the style definition yourself for the complete site.

#### results.resultsPerPage

**Type** Integer

**TS Path** plugin.tx_solr.search.results.resultsPerPage

**Since** 1.0

**Default** {$plugin.tx_solr.search.results.resultsPerPage}

Sets the number of shown results per page.

#### results.resultsPerPageSwitchOptions

**Type** String

**TS Path** plugin.tx_solr.search.results.resultsPerPageSwitchOptions

**Since** 1.0

**Default** 10, 25, 50

Defines the shown options of possible results per page.

#### results.fieldProcessingInstructions

**Type** Array

**TS Path** plugin.tx_solr.search.results.fieldProcessingInstructions

**Since** 1.0

**Options** timestamp, utf8Decode, skip

Mapping of fieldname to processing instructions. Available instructions: timestamp, utf8Decode, skip (removes the field from the result).

The utf8Decode option has been removed in version 2.8.

#### results.fieldRenderingInstructions

**Type** cObject

**TS Path** plugin.tx_solr.search.results.fieldRenderingInstructions

**Since** 1.0

Additional rendering instructions for specified fields.

#### results.pagebrowser

**Since** 2.0

**TS Path** plugin.tx_solr.search.results.pagebrowser

Allows to set configuration options for the pagebrowser provided by EXT:pagebrowse.

#### results.ignorePageBrowser

**Type** Boolean

**TS Path** plugin.tx_solr.search.results.ignorePageBrowser

**Since** 1.0

**Default** 0

**Options** 0, 1

If enabled, the selected page will be ignored. Useful if you place two search plugins on a page but want one of the to always show the first results only.

#### results.showDocumentScoreAnalysis

**Type** Boolean

**TS Path** plugin.tx_solr.search.results.showDocumentScoreAnalysis

**Since** 2.5-dkd

**Default** 0

**Options** 0,1

If enabled, the analysis and display of the score analysis for logged in backend users will be initialized.

#### results.markResultTypeBoundaries

**Type** Boolean

**TS Path** plugin.tx_solr.search.results.markResultTypeBoundaries

**Since** 2.0, 2.5-dkd

**Default** 0

**Options** 0,1

This option allows for some kind of fake or client side (EXT:solr being the client) grouping of results. First you need to sort results by the type field. Then whenever the type field changes during rendering of the results, special fields are added to the documents at the result document type boundary. In your template you can then use an IF condition to insert markup at those boundaries based on those fields' values. The special document field names are "typeBegin" and "typeEnd". They have a value set to the new/old document type appended with "_begin"/"end".

Example:

A result set with documents of types tt_news, pages, and tt_address:

```
1 pages      -> typeBegin = pages_begin
2 pages
3 pages      -> typeEnd   = pages_end
4 tt_address -> typeBegin = tt_address_begin
5 tt_address
6 tt_address -> typeEnd   = tt_address_end
7 tt_news    -> typeBegin = tt_news_begin
8 tt_news
9 tt_news    -> typeEnd   = tt_news_end
```

Example template snippet:

```
<!-- ###LOOP:RESULT_DOCUMENTS### begin -->
<!-- ###LOOP_CONTENT### -->

<!-- ###IF:###RESULT_DOCUMENT.typeBegin###|==|pages_begin### begin -->
<li>Pages</li>
<!-- ###IF:###RESULT_DOCUMENT.typeBegin###|==|pages_begin### end -->

<!-- ... regular page results here ... -->

<!-- ###IF:###RESULT_DOCUMENT.typeEnd###|==|pages_end### begin -->
<li>Pages End</li>
<!-- ###IF:###RESULT_DOCUMENT.typeEnd###|==|pages_end### end -->

...

<!-- ###IF:###RESULT_DOCUMENT.typeBegin###|==|tt_address_begin### begin -->
<li>Contacts</li>
<!-- ###IF:###RESULT_DOCUMENT.typeBegin###|==|tt_address_begin### end -->

<!-- ###IF:###RESULT_DOCUMENT.typeBegin###|==|tt_news_begin### begin -->
<li>News</li>
```

```
<!-- ###IF:###RESULT_DOCUMENT.typeBegin###|==|tt_news_begin### end -->

<!-- ... more markup here ... -->

<!-- ###LOOP_CONTENT### -->
<!-- ###LOOP:RESULT_DOCUMENTS### end -->
```

## spellchecking

### spellchecking

**Type** Boolean

**TS Path** plugin.tx_solr.search.spellchecking

**Since** 1.0

**Default** 0

Set *plugin.tx_solr.search.spellchecking = 1* to enable spellchecking / did you mean.

### spellchecking.wrap

**Type** String

**TS Path** plugin.tx_solr.search.spellchecking.wrap

**Since** 1.0

**Default** | ###LLL:didYouMean### | |

This can be used to configure a custom wrap for your did you mean rendering.

## lastSearches

### lastSearches

**Type** Boolean

**TS Path** plugin.tx_solr.search.lastSearches

**Since** 1.3-dkd

**Default** 0

Set *plugin.tx_solr.lastSearches = 1* to display a list of the last searches.

### lastSearches.limit

**Type** Integer

**TS Path** plugin.tx_solr.search.lastSearches.limit

**Since** 1.3-dkd

**Default** 10

Defines the number of last searches, that should get minded.

#### lastSearches.mode

**Type** String

**TS Path** plugin.tx_solr.search.lastSearches.mode

**Since** 1.3-dkd

**Default** user

**Options** user, global

If mode is user, keywords will get stored into the session. If mode is global keywords will get stored into the database.

### frequentSearches

#### frequentSearches

**Type** Boolean

**TS Path** plugin.tx_solr.search.frequentSearches

**Since** 1.3-dkd, 2.8

**Default** 0

Set *plugin.tx_solr.search.frequentSearches = 1* to display a list of the frequent / common searches.

#### frequentSearches.useLowercaseKeywords

**Type** Boolean

**TS Path** plugin.tx_solr.search.frequentSearches.useLowercaseKeywords

**Since** 2.9

**Default** 0

When enabled, keywords are written to the statistics table in lower case.

#### frequentSearches.minSize

**Type** Integer

**TS Path** plugin.tx_solr.search.frequentSearches.minSize

**Since** 1.3-dkd, 2.8

**Default** 14

The difference between frequentSearches.maxSize and frequentSearches.minSize is used for calculating the current step.

#### frequentSearches.maxSize

**Type** Integer

**TS Path** plugin.tx_solr.search.frequentSearches.maxSize

**Since** 1.3-dkd, 2.8

**Default** 32

The difference between frequentSearches.maxSize and frequentSearches.minSize is used for calculating the current step.

#### frequentSearches.limit

**Type** Integer

**TS Path** plugin.tx_solr.search.frequentSearches.limit

**Since** 1.3-dkd, 2.8

**Default** 20

Defines the maximum size of the list by frequentSearches.select.

#### frequentSearches.select

**Type** cObject

**TS Path** plugin.tx_solr.search.frequentSearches.select

**Since** 1.3-dkd, 2.8

Defines a database connection for retrieving statistics.

### sorting

#### sorting

**Type** Boolean

**TS Path** plugin.tx_solr.search.sorting

**Since** 1.0

**Default** 0

Set *plugin.tx_solr.search.sorting = 1* to allow sorting of results.

#### sorting.defaultOrder

**Type** String

**TS Path** plugin.tx_solr.search.sorting.defaultOrder

**Since** 1.0

**Default** ASC

**Options** ASC, DESC

Sets the default sort order for all sort options.

#### sorting.options

This is a list of sorting options. Each option has a field and label to be used. By default the options title, type, author, and created are configured, plus the virtual relevancy field which is used for sorting by default.

Example:

```
plugin.tx_solr.search {
    sorting {
        options {
            relevance {
                field = relevance
                label = Relevance
            }

            title {
                field = sortTitle
                label = Title
            }
        }
    }
}
```

Note: As mentioned before **relevance** is a virtual field that is used to **reset** the sorting. Sorting by relevance means to have the order provided by the scoring from solr. That the reason why sorting **descending** on relevance is not possible.

#### sorting.options.[optionName].label

**Type** String / stdWrap

**TS Path** plugin.tx_solr.search.sorting.options.[optionName].label

**Since** 1.0

Defines the name of the option's label. Supports stdWrap.

#### sorting.options.[optionName].field

**Type** String / stdWrap

**TS Path** plugin.tx_solr.search.sorting.options.[optionName].field

**Since** 1.0

Defines the option's field. Supports stdWrap.

#### sorting.options.[optionName].defaultOrder

**Type** String

**TS Path** plugin.tx_solr.search.sorting.options.[optionName].defaultOrder

**Since** 2.2

**Default** ASC

**Options** ASC, DESC

Sets the default sort order for a particular sort option.

#### sorting.options.[optionName].fixedOrder

**Type** String

**TS Path** plugin.tx_solr.search.sorting.options.[optionName].fixedOrder

**Since** 2.2

**Default** ASC

**Options** ASC, DESC

Sets a fixed sort order for a particular sort option that can not be changed.

### faceting

#### faceting

**Type** Boolean

**TS Path** plugin.tx_solr.search.faceting

**Since** 1.0

**Default** 0

Set *plugin.tx_solr.search.faceting = 1* to enable faceting.

#### faceting.minimumCount

**Type** Integer

**TS Path** plugin.tx_solr.search.faceting.minimumCount

**Since** 1.0

**Default** 1

**See** Apache Solr Wiki / Faceting mincount Parameter

This indicates the minimum counts for facet fields should be included in the response.

#### faceting.sortBy

**Type** String

**TS Path** plugin.tx_solr.search.faceting.sortBy

**Since** 1.0

**Default** count

**Options** count, index, 1, 0, true, false, alpha (1.2, 2.0), lex (1.2, 2.0)

**See** Apache Solr Wiki / Faceting sortParameter Parameter

Defines how facet options are sorted, by default they are sorted by count of results, highest on top. count, 1, true are aliases for each other.

Facet options can also be sorted alphabetically (lexicographic by indexed term) by setting the option to index. index, 0, false, alpha (from version 1.2 and 2.0), and lex (from version 1.2 and 2.0) are aliases for index.

#### faceting.limit

**Type** Integer

**TS Path** plugin.tx_solr.search.faceting.limit

**Since** 1.0

**Default** 10

Number of options to display per facet. If more options are returned by Solr, they are hidden and can be expanded by clicking a "show more" link. This feature uses a small javascript function to collapse/expand the additional options.

#### faceting.facetLimit

**Type** Integer

**TS Path** plugin.tx_solr.search.faceting.facetLimit

**Since** 6.0

**Default** 100

Number of options of a facet returned from solr.

#### faceting.singleFacetMode

**Type** Boolean

**TS Path** plugin.tx_solr.search.faceting.singleFacetMode

**Since** 1.2, 2.0

**Default** 0

**Options** 0, 1

If enabled, the user can only select an option from one facet at a time.

Lets say you have two facets configured, type and author. If the user selects a facet option from type its filter is added to the query. Normally when selecting another option from the other facet - the author facet - this would lead to having two facet filters applied to the query. When this option is activated the option from the author facet will simply replace the first option from the type facet.

#### faceting.keepAllFacetsOnSelection

**Type** Boolean

**TS Path** plugin.tx_solr.search.faceting.keepAllFacetsOnSelection

**Since** 2.2

**Default** 0

**Options** 0, 1

When enabled selecting an option from a facet will not reduce the number of options available in other facets.

#### faceting.removeFacetLinkText

**Type** String

**TS Path** plugin.tx_solr.search.faceting.removeFacetLinkText

**Since** 1.0

**Default** @facetLabel: @facetText

Defines the text for a link used for removing a given facet from the search results. You can use the following placeholders in this text: @facetValue, @facetName, @facetLabel, @facetText

#### faceting.showAllLink.wrap

**Type** String

**TS Path** plugin.tx_solr.search.faceting.showAllLink.wrap

**Since** 1.0

**Default** <li>|</li>

Defines the output of the "Show more" link, that is rendered if there are more facets given than set by faceting.limit.

#### faceting.showEmptyFacets

**Type** Boolean

**TS Path** plugin.tx_solr.search.faceting.showEmptyFacets

**Since** 1.3

**Default** 0

**Options** 0, 1

By setting this option to 1, you will allow rendering of empty facets. Usually, if a facet does not offer any options to filter a resultset of documents, the facet header will not be shown. Using this option allows the header still to be rendered when no filter options are provided.

#### faceting.facetLinkATagParams

**Type** String

**TS Path** plugin.tx_solr.search.faceting.facetLinkATagParams

**Since** 2.0

**Default** rel="nofollow"

With this option you can add A-Tag attributes for links of all facet-options.

```
plugin.tx_solr.search.faceting.facetLinkATagParams = class="green"
```

#### faceting.facetLinkUrlParameters

**Type** String

**TS Path** plugin.tx_solr.search.faceting.facetLinkUrlParameters

**Since** 2.8

Allows to add URL GET parameters to the links build in facets.

#### faceting.facetLinkUrlParameters.useForFacetResetLinkUrl

**Type** Boolean

**TS Path** plugin.tx_solr.search.faceting.facetLinkUrlParameters.useForFacetResetLinkUrl

**Since** 2.8

Allows to prevent adding the URL parameters to the facets reset link by setting the option to 0.

#### faceting.facets

**Type** Array

**TS Path** plugin.tx_solr.search.faceting.facets

**Since** 1.0

**Default** type

**See** Apache Solr Wiki / Faceting Overview

Defines which fields you want to use for faceting. It's a list of facet configurations.

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

  category {
    field = category_stringM
    label = Category
  }
}
```

#### faceting.facets.[facetName] - single facet configuration

You can add new facets by simply adding a new facet configuration in TypoScript. [facetName] represents the facet's name and acts as a configuration "container" for a single facet. All configuration options for a facet are defined within that "container".

A facet will use the values of a configured index field to offer these values as filter options to your site's visitors. You need to make sure that the facet field's type allows to sort the field's value; like string, int, and other primitive types.

To configure a facet you only need to provide the label and field configuration options, all other configuration options are optional.

#### faceting.facets.[facetName].field

**Type** String

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].field

**Since** 1.0

**Required** yes

Which field to use to create the facet.

#### faceting.facets.[facetName].label

**Type** String

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].label

**Since** 1.0

**Required** yes

Used as a headline or title to describe the options of a facet.

#### faceting.facets.[facetName]. selectingSelectedFacetOptionRemovesFilter

**Type** Boolean

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].selectingSelectedFacetOptionRemovesFilter

**Since** 1.2, 2.0

**Default** 0

**Options** 0, 1

Activating this option for a facet makes the facets option links behave like on/off switches: You click them once to activate a facet, you click them a second time to deactivate the facet again.

Feel free to suggest a better name for this option.

#### faceting.facets.[facetName].keepAllOptionsOnSelection

**Type** Boolean

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].keepAllOptionsOnSelection

**Since** 1.2, 2.0

**Default** 0

**Options** 0, 1

Normally, when clicking any option link of a facet this would result in only that one option being displayed afterwards. By setting this option to one, you can prevent this. All options will still be displayed.

This is useful if you want to allow the user to select more than one option from a single facet.

#### faceting.facets.[facetName].singleOptionMode

**Type** Boolean

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].singleOptionMode

**Since** 1.3, 2.0

**Default** 0

**Options** 0, 1

When enabled together with keepAllOptionsOnSelection a user can select one option of the facet only at a time. Selecting a different option than the currently selected option results in the new option to replace the old one. The behavior thus is similar to a select box or a set of radio buttons.

This option can not be used together with selectingSelectedFacetOptionRemovesFilter as it overrides its behavior.

#### faceting.facets.[facetName].operator

**Type** String

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].operator

**Since** 1.2, 2.0

**Default** AND

**Options** OR, AND

When configuring a facet to allow selection of multiple options, you can use this option to decide whether multiple selected options should be combined using AND or OR.

#### faceting.facets.[facetName].sortBy

**Type** String

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].sortBy

**Since** 1.2

**Default**

-

**Options** alpha (aliases: index, lex)

Sets how a single facet's options are sorted, by default they are sorted by number of results, highest on top. Facet options can also be sorted alphabetically by setting the option to alpha.

#### faceting.facets.[facetName].manualSortOrder

**Type** String

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].manualSortOrder

**Since** 2.2

By default facet options are sorted by the amount of results they will return when applied. This option allows to manually adjust the order of the facet's options. The sorting is defined as a comma-separated list of options to re-order. Options listed will be moved to the top in the order defined, all other options will remain in their original order.

Example - We have a category facet like this:

```
News Category
+ Politics (256)
+ Sports (212)
+ Economy (185)
+ Culture (179)
+ Health (132)
+ Automobile (99)
+ Travel (51)
```

Using *faceting.facets.[facetName].manualSortOrder = Travel, Health* will result in the following order of options:

```
News Category
+ Travel (51)
+ Health (132)
+ Politics (256)
+ Sports (212)
+ Economy (185)
+ Culture (179)
+ Automobile (99)
```

#### faceting.facets.[facetName].reverseOrder

**Type** Boolean

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].reverseOrder

**Since** 3.0

**Default** 0

**Options** 0, 1

Reverses the order of facet options.

#### faceting.facets.[facetName].showEvenWhenEmpty

**Type** Boolean

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].showEvenWhenEmpty

**Since** 2.0

**Default** 0

**Options** 0, 1

Allows you to display a facet even if it does not offer any options (is empty) and although you have set *plugin.tx_solr.search.faceting.showEmptyFacets = 0*.

#### faceting.facets.[facetName].includeInAvailableFacets

**Type** Boolean

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].includeInAvailableFacets

**Since** 1.3

**Default** 1

**Options** 0, 1

By setting this option to 0, you can prevent rendering of a given facet within the list of available facets.

This is useful if you render the facet somewhere eles on the page using the facet view helper and don't want the facet to be rendered twice.

#### faceting.facets.[facetName].includeInUsedFacets

**Type** Boolean

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].includeInUsedFacets

**Since** 2.0

**Default** 1

**Options** 0, 1

By setting this option to 0, you can prevent rendering of a given facet within the list of used facets.

#### faceting.facets.[facetName].type

**Type** String

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].type

**Since** 2.0

Defines the type of the facet. By default all facets will render their facet options as a list. PHP Classes can be registered to add new types. Using this setting will allow you to use such a type and then have the facet's options rendered and processed by the registered PHP class.

#### faceting.facets.[facetName].[type]

**Type** Array

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].[type]

**Since** 2.0

When setting a special type for a facet you can set further options for this type using this array.

Example (numericRange facet displayed as a slider):

```
plugin.tx_solr.search.faceting.facets.size {
      field = size_intS
      label = Size

      type = numericRange
      numericRange {
          start = 0
          end = 100
          gap = 1
     }
}
```

#### faceting.facets.[facetName].requirements.[requirementName]

**Type** Array

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].requirements.[requirementName]

**Since** 2.2

Allows to define requirements for a facet to be rendered. These requirements are dependencies on values of other facets being selected by the user. You can define multiple requirements for each facet. If multiple requirements are defined, all must be met before the facet is rendered.

Each requirement has a name so you can easily recognize what the requirement is about. The requirement is then defined by the name of another facet and a list of comma-separated values. At least one of the defined values must be selected by the user to meet the requirement.

There are two magic values for the requirement's values definition:

- __any: will mark the requirement as met if the user selects any of the required facet's options

- __none: marks the requirement as met if none of the required facet's options is selected. As soon as any of the required facet's options is selected the requirement will not be met and thus the facet will not be rendered

Example of a category facet showing only when the user selects the news type facet option:

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

                category {
                    label = Category
                    field = category_stringS
                    requirements {
                        typeIsNews {
                          # typeIsNews is the name of the requirement, c
                          # choose any so you can easily  recognize what it does
                          facet = type
                          # The name of the facet as defined above
                          values = news
                          # The value of the type facet option as
                          # it is stored in the Solr index
                        }
                    }
                }
            }
        }
    }
}
```

#### faceting.facets.[facetName].renderingInstruction

**Type** cObject

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].renderingInstruction

**Since** 1.0

Overwrites how single facet options are rendered using TypoScript cObjects.

Example: (taken from issue #5920)

```
plugin.tx_solr {
    search {
        faceting {
            facets {
                type {
                    renderingInstruction = CASE
                    renderingInstruction {
                        key.field = optionValue
```

```
                        pages = TEXT
                        pages.value = Pages
                        pages.lang.de = Seiten

                        tx_solr_file = TEXT
                        tx_solr_file.value = Files
                        tx_solr_file.lang.de = Dateien

                        tt_news = TEXT
                        tt_news.value = News
                        tt_news.lang.de = Nachrichten
                    }
                }

                language {
                    renderingInstruction = CASE
                    renderingInstruction {
                        key.field = optionValue

                        0 = TEXT
                        0.value = English
                        0.lang.de = Englisch

                        1 = TEXT
                        1.value = German
                        1.lang.de = Deutsch
                    }
                }
            }
        }
    }
}
```

EXT:solr provides the following renderingInstructions that you can use in your project:

**FormatDate**:

This rendering instruction can be used in combination with a date field or an integer field that hold a timestamp. You can use this rendering instruction to format the facet value on rendering. A common usecase for this is, when the datatype in solr needs to be sortable (date or int) but you need to render the date as readable date option in the frontend:

```
plugin.tx_solr.search.faceting.facets {
    created {
        field = created
        label = Created
        sortBy = alpha
        reverseOrder = 1
        renderingInstruction = TEXT
        renderingInstruction {
           field = optionValue
           postUserFunc =␣
↪ApacheSolrForTypo3\Solr\Domain\Search\ResultSet\Facets\RenderingInstructions\FormatDate-
↪>format
        }
    }
}
```

#### faceting.facets.[facetName].facetLinkATagParams

**Type** String

**TS Path** plugin.tx_solr.search.faceting.facets.[facetName].facetLinkATagParams

**Since** 2.0

With this option you have the possibility to add A-Tag attributes for all option-links of a single facet. This option overwrites the global setting "faceting.facetLinkATagParams". See "faceting.facetLinkATagParams" for more information.

### elevation

#### elevation

**Type** Boolean

**TS Path** plugin.tx_solr.search.elevation

**Since** 3.0

**Default** 0

Set plugin.tx_solr.search.elevation = 1 to enable content elevation in search results.

#### elevation.forceElevation

**Type** Boolean

**TS Path** plugin.tx_solr.search.elevation.forceElevation

**Since** 3.0

**Default** 1

Forces content elevation to be active.

#### elevation.markElevatedResults

**Type** Boolean

**TS Path** plugin.tx_solr.search.elevation.markElevatedResults

**Since** 3.0

**Default** 1

If enabled, elevated results are marked with CSS class "results-elevated".

### variants

By using variants you can shrink down multiple documents with the same value in one field into one document and make similar documents available in the variants property. By default the field variantId is used as Solr collapsing criteria. This can be used e.g. as one approach of deduplication to group similar documents into on "root" SearchResult.

To use the different variants of the documents you can access "document.variants" to access the expanded documents.

This can be used for example for de-duplication to list variants of the same document below a certain document.

Note: Internally this is implemented with Solr field collapsing

**Type** Boolean

**TS Path** plugin.tx_solr.search.variants

**Since** 6.0

**Default** 0

Set plugin.tx_solr.search.variants = 1 to enable the variants in search results.

#### variants.expand

Used to expand the document variants to the document.variants property.

**Type** Boolean

**TS Path** plugin.tx_solr.search.variants.expand

**Since** 6.0

**Default** 1

#### variants.variantField

Used to expand the document variants to the document.variants property.

**Note:**: The field must be a numeric field or a string field! Not a text field!

**Type** String

**TS Path** plugin.tx_solr.search.variants.variantField

**Since** 6.0

**Default** variantId

#### variants.limit

Limit of expanded documents.

**Type** Integer

**TS Path** plugin.tx_solr.search.variants.limit

**Since** 6.0

**Default** 10

## tx_solr.suggest

This feature allows you to show a suggest layer that suggest terms that start with the letters that have been typed into the search field

### numberOfSuggestions

**Type** Integer

**TS Path** plugin.tx_solr.suggest.numberOfSuggestions

**Since** 1.1

**Default** 10

Sets the number of suggestions returned and displayed in the layer attached to the search field.

### suggestField

**Type** String

**TS Path** plugin.tx_solr.suggest.suggestField

**Since** 1.1

**Default** spell

Sets the Solr index field used to get suggestions from. A general advice is to use a field without stemming on it. For practical reasons this is currently the spell checker field.

### forceHttps

**Type** Boolean

**TS Path** plugin.tx_solr.suggest.forceHttps

**Since** 1.1

**Default** 0

**Options** 0,1

If enabled, HTTPS will be used for querying suggestions. Otherwise HTTP will be used.

### treatMultipleTermsAsSingleTerm

**Type** Boolean

**TS Path** plugin.tx_solr.suggest.treatMultipleTermsAsSingleTerm

**Since** 1.4 / 1.7-dkd

**Default** 0

**Options** 0,1

When a user types multiple words into your search field they usually are split up into full keywords used in the query's q parameter and the last part being the partial keyword in the facet.prefix parameter. Enabling this setting moves everything into the facet.prefix parameter. This is usually only useful when using a string field as suggest / auto-complete source field.

Example - Here "Hello Solr" are the full keywords and the user started typing "World" so that "Wo" is used as the partial keyword:

“Hello Solr Wo” -> q=Hello Solr, facet.prefix=Wo (default) “Hello Solr Wo” -> q=<empty>, facet.prefix=Hello Solr Wo (treatMultipleTermsAsSingleTerm)

## tx_solr.statistics

This section allows you to configure the logging for statistics.

**Note**: The statistics are logged into a mysql table. This might not make sense for high frequently used searches. In this case you should think about to connect a dedicated tracking tool.

### statistics

**Type** Boolean

**TS Path** plugin.tx_solr.statistics

**Since** 2.0

**Default** 0

Set *plugin.tx_solr.statistics = 1* to log statistics.

### statistics.anonymizeIP

**Type** Integer

**TS Path** plugin.tx_solr.statistics.anonymizeIP

**Since** 2.0

**Default** 0

Defines the number of octets of the IP address to anonymize in the statistics log records.

### statistics.addDebugData

**Type** Boolean

**TS Path** plugin.tx_solr.statistics.addDebugData

**Since** 6.1

**Default** 0

Adds debug data to the columns *time_total*, *time_preparation* and *time_processing* in the table *tx_solr_statistics* from the result of the search query.

**Note**: Enabling addDebugData can have performance impact since debugMode is appended to queries.

## tx_solr.viewHelpers

This sections defines settings concerning solr's viewhelpers.

### crop.maxLength

**Type** Integer

**TS Path** plugin.tx_solr.viewHelpers.crop.maxLength

**Since** 1.2

**Default** 30

Sets the maximum length the given string should be shortened to.

### crop.cropIndicator

**Type** String

**TS Path** plugin.tx_solr.viewHelpers.crop.cropIndicator

**Since** 1.2

**Default** ...

Sets the crop indicator that is used for marking the cropped part.

## tx_solr.logging

This section defines logging options. All loggings will be available in the TYPO3 logging framework.



### debugOutput

**Type** Boolean

**TS Path** plugin.tx_solr.logging.debugOutput

**Default** 0

**Options** 0,1

**Since** 6.1

If enabled the written log entries will be printed out as debug message in the frontend or to the TYPO3 debug console in the backend. This setting replaces the previous setting *plugin.tx_solr.logging.debugDevLogOutput* which was needed, when the devLog was used.

### exceptions

**Type** Boolean

**TS Path** plugin.tx_solr.logging.exceptions

**Default** 1

**Options** 0,1

**Since** 1.0

If enabled, thrown exceptions are logged.

### indexing

**Type** Boolean

**TS Path** plugin.tx_solr.logging.indexing

**Default** 1

**Options** 0,1

**Since** 1.0

If enabled, logs when pages / documents are indexed.

### indexing.indexQueueInitialization

**Type** Boolean

**TS Path** plugin.tx_solr.logging.indexing.indexQueueInitialization

**Default** 1

**Options** 0,1

**Since** 2.0

If enabled, logs the query used to initialize the indexing queue.

### indexing.indexQueuePageIndexerGetData

**Type** Boolean

**TS Path** plugin.tx_solr.logging.indexing.indexQueuePageIndexerGetData

**Default** 1

**Options** 0,1

**Since** 2.0

If enabled, the requested data will be logged. Request data includes item, url, parameters, headers, data, decodedData and report.

### query.filters

**Type** Boolean

**TS Path** plugin.tx_solr.logging.query.filters

**Default** 1

**Options** 0,1

**Since** 1.0

If enabled, filters will be logged when they get added to the Solr query.

### query.searchWords

**Type** Boolean

**TS Path** plugin.tx_solr.logging.query.searchWords

**Default** 1

**Options** 0,1

**Since** 1.0

If enabled, received search queries will be logged.

### query.queryString

**Type** Boolean

**TS Path** plugin.tx_solr.logging.query.queryString

**Default** 1

**Options** 0,1

**Since** 1.0

If enabled, query string parameters and the respective response will be logged.

### query.rawPost

**Type** Boolean

**TS Path** plugin.tx_solr.logging.query.rawPost

**Default** 1

**Options** 0,1

**Since** 1.0

If enabled, POST requests against the Solr server will be logged.

### query.rawGet

**Type** Boolean

**TS Path** plugin.tx_solr.logging.query.rawGet

**Default** 1

**Options** 0,1

**Since** 1.0

If enabled, GET requests against the Solr server will be logged.

# Extension Configuration

The following settings can be defined in the extension manager

## useConfigurationFromClosestTemplate

**Type** Boolean

**Since** 6.1

**Default** 0

When this setting is active the closest page with a typoscript template will be used to fetch the configuration. This improves the performance but limits also the possibilities. E.g. conditions can not be used that are related to a certain page.

## useConfigurationTrackRecordsOutsideSiteroot

**Type** Boolean

**Since** 6.1

**Default** 1

A common common scenario is to have a site and a storage folder for records parallel to it on the same level (f.e.) If you don't want this behaviour - it should be set to false.

## allowSelfSignedCertificates

**Type** Boolean

**Since** 6.1

**Default** 0

Can be used to allow self signed certificates - when using the SSL protocol.

CHAPTER

TEN

# DATABASE

## Database indexes

Some of the SQL statements performed on the pages table in TYPO3 perform extensive operations while copying page-trees. These operations can be speeded by by adding 2 indexes to the standard table pages.

The indexes are: * content_from_pid_deleted (content_from_pid, deleted), * doktype_no_search_deleted (doktype, no_search, deleted)

It is not required that these indexes are created by in above scenarios considerable performance gains can be achieved.

CHAPTER

ELEVEN

# DEVELOPMENT

There are many ways to extend and hook into EXT:solr to customize EXT:solr for your needs.

## Indexing

In this section i describe the possibilities to extend page indexing in EXT:solr with custom code.

### Page Indexing

There are several points to extend the Typo3PageIndexer class and register own classes that are used during the indexing.

#### indexPageAddDocuments

Registered classes can be used to add additional documents to solr when a page get's indexed.

Registration with: $GLOBALS['TYPO3_CONF_VARS']['EXTCONF']['solr']['Indexer']['indexPageAddDocuments']
Required Interface: AdditionalPageIndexer

#### indexPageSubstitutePageDocument

Registered classes can be used to replace/substitute a Solr document of a page.

Registration with: $GLOBALS['TYPO3_CONF_VARS']['EXTCONF']['solr']['Indexer']['indexPageSubstitutePageDocument']
Required Interface: SubstitutePageIndexer

#### indexPagePostProcessPageDocument

Registered classes can be used to post process a Solr document of a page.

Registration with: $GLOBALS['TYPO3_CONF_VARS']['EXTCONF']['solr']['Indexer']['indexPagePostProcessPageDocument']
Required Interface: PageDocumentPostProcessor

CHAPTER

TWELVE

# FAQ - FREQUENTLY ASKED QUESTIONS

**What does the term '"Core"<https://cwiki.apache.org/confluence/display/solr/Solr+Cores+and+solr.xml>'_ mean?**

This term relates to Apache Solr indexes and means a single distinct part of an index. It is possible to use multiple cores on one single Apache Solr instance. Good examples are using a different Apache Solr core for each language or of course a separate core for each website. For more informations please refer to the Apache Solr documentation.

**Where can I report a bug?**

Please make sure that this bug is not reported already, use also the search function of our issue tracker. Our issue tracker is on GitHub.

**Where can I report a security issue?**

If you have found a security issue in our extension, please do not post about it in a public channel. Please send us an email with detailed description of found vulnerability.

**Is there some chat/irc channel for EXT:solr available?**

Join us on the official Slack for TYPO3 and get answers related to EXT:solr in the #ext-solr channel immediately!

**When i open the search page i see the message 'Search is currently not available. ', whats wrong?**

Did you configure your solr connection as required?

- Please read "*Configure Extension*" and check if you have configured everything
- Did you configure solr server and port and does the scheme and path match?
- Did you click "Initialize connection" after configuring the solr server?
- Can you access the solr server with wget or curl from the command line?

- Is the system report of EXT:solr green?

**In which cases do I want to trigger indexing manually?**

- after changing any configuration file.
- after modifying synonyms, stop words, protected words in TYPO3 Backend -> Search

Moreover by changing core/index configuration you need to reload the core to make the changes become active. To reload configuration you can either restart the whole Solr server or simply reload a specific core.

**I want to index files with EXT:solr. How can i do that?**

We provide an addon called EXT:solrfal, that allows you to index files from FAL into Solr. This addon is currently available for partner only.

**How can i use fluid templates with EXT:solr?**

For the fluid rendering we provide the addon EXT:solrfluid, that allows you to render your search results with fluid.

**Which versions of EXT:solr / EXT:solrfal and EXT:solrfluid work together?**

Please check the *Appendix - Version Matrix*, the you can find the proposed version combinations.

**My indexed documents are empty, i can not find the content of a page?**

Did you configure the search markers(<!– *TYPO3SEARCH_begin* –> and <!– *TYPO3SEARCH_end* –>) on your page? Check the paragraph Search Markers and make sure your page renders them.

**I have languages in TYPO3 that are not relevant for the search. How can i exclude them?**

You need to enable the search just for the relevant languages.

Example:

```
plugin.tx_solr.enabled = 0

[globalVar = GP:L = 0]
    plugin.tx_solr {
        enabled = 1
        solr.path = /solr/core_de/
    }
[globalVar = GP:L = 8|9]
    plugin.tx_solr {
        enabled = 1
        solr.path = /solr/core_en/
    }
[END]
```

**The extension is indexing into the wrong core for multi-language sites. What's wrong?**

When indexing pages the page indexer retrieves the core from the TypoScript configuration. That configuration is determined by the language (GET L parameter). However, when the indexer tries to index a page that has not been translated TYPO3 will by default still render the page but falling back to the default language page. By that TYPO3 will also use the TypoScript configuration for the default language which usually points to a different Solr core.

Solution: Make sure you have configured config.sys_language_mode to use content_fallback. This way TYPO3 will fall back to the configured language's content, but will use the TypoScript configuration for the requested language.

**When i change a record, no update is detected. What's wrong?**

Are your records inside of your site root? EXT:solr record monitor processes records that belong to your site, which means they need to be below your site root. If you want to index records that are outside your sideroot, you need to configure the page id's of the sysfolder as additionalPageIds:

```
plugin.tx_solr.index.queue.[yourQueueName].additionalPageIds = 4711,4712
```

**There are two datatypes for text stringS and textS. When should i choose which datatype?**

String data types like stringS store the *raw* string. No processing, like stemming, splitting etc. is applied. The processing is useful when you want to search in the field and support more then exact matches. When you just want to display the content you should choose a *stringS* type, when you want to search in the field you should choose *textS*.

**I am adding content to a dynamic field but when i search for the content i can not find the document. What's wrong?**

Beside the indexing part you need to configure the query part. Make sure that all relevant fields are configured as query fields:

```
plugin.tx_solr.search.query.queryFields := addToList(test_textS\^1.0)
```

**I don't find the expected document on the first position. What can i do?**

:) That's a good question. In the end, solr is a search and the sorting depends on the score, not as in a database on one or two simple criterion.

In the end solr provides a lot of settings that influence the score calculation and you need to tune the results to you needs. The following settings are helpful to tune your results.

*Check your data*

The quality of you data is important. Maybe a document is on the first position because, the search term is really relevant for it? Maybe it is an option to change the content?

*Adjust the query field boost factors*

For each query field there is a boost value after the ^ sign. To increase the factor of a single field for the whole query, you can increase the number in the query fields.

Example:

```
plugin.tx_solr.search.query.queryFields = title\^20.0, title\^15.0
```

*Use boostFunctions or boostQueries*

For use cases like "*news* are always more important then *pages*" or "Newer documents should be at the beginning" you can use boostFunctions (*query.boostFunction*) or boostQueries (*query.boostQuery*)

*The search term only exists as a synonym*

You can use the backend module synonyms (Synonyms) to maintain synonyms and configure solr to retrieve documents by a term that is not naturally inside the document.

*Ask DKD support*

Beside that, there are more options to tune. The DKD support can help you, to analyze and tune your search results. Call +49 (0)69 - 247 52 18-0.

**Non ASCII characters like german umlauts do not work when i search, how do I fix that?**

To allow search with umlauts Tomcat needs to be configured to use UTF-8 encoded urls. Go to apache-tomcat/conf/server.xml and change the URIEncoding parameter:

```
<Connector port="8080" protocol="HTTP/1.1"
    connectionTimeout="20000" redirectPort="8443"
    URIEncoding="UTF-8" />
```

**How can I change Solr's schema and add custom fields?**

Please do not change the shipped solr schema. There are a lot of dynamic fields (*Appendix - Dynamic Fields*) that can be used to index any kind of datatype.

**I am using varnish before my site. How can i index pages properly?**

SOLR Indexer might have some issues, when the page to index is behind a Varnish Proxy. We have collected two ways of solving this issue

*Bypassing varnish:*

Bypass when X-Tx-Solr-Iq is present

The SOLR indexer request send the header X-Tx-Solr-Iq.

To have bypass the Varnish caching, put this into your sub vcl_recv part of the configuration

```
if (req.http.X-Tx-Solr-Iq) {
    return(pipe);
}
```

*Using Cache-Control:*

Put this into your sub vcl_fetch part of the configuration

```
if (req.http.Cache-Control ~ "no-cache") {
    set beresp.ttl = 0s;
    # Make sure ESI includes are processed!
    esi;
    set beresp.http.X-Cacheable = "NO:force-reload";
    # Make sure that We remove all cache headers, so the Browser does not cache it␣
↪for us!
    remove beresp.http.Cache-Control;
    remove beresp.http.Expires;
    remove beresp.http.Last-Modified;
    remove beresp.http.ETag;
    remove beresp.http.Pragma;

      return (deliver);
}
```

**I want to build the Dockerfile_full image on my mac with a local volume, how can i do that?**

The following example shows how to build the Dockerfile image and start a container with a mapped local volume (only for the data). This was tested with "Docker for Mac" (not Docker Toolbox). Before executing the example, make sure, that you have added "~/solrdata" as allowed volume in the docker configuration.

```
# build the image
docker build -t typo3-solr -f Dockerfile .
```

```
# create volume directory locally
mkdir -p ~/solrdata

# add solr group to volume directory
sudo chown :8983 ~/solrdata

# run docker container from image with volume
docker run -d -p 127.0.0.1:8282:8983 -v ~/solrdata:/opt/solr/server/solr/data typo3-
↪solr
```

**Can i index a https (SSL) site?**

Yes. You need a ssl certificate (can be self signed) and change the following setting:

```
plugin.tx_solr.index.queue.pages.frontendDataHelper.scheme = https
```

**I want to index a value into a multiValue field from a user function. How can i do that?**

You can do that, by using SOLR_MULTIVALUE

```
plugin.tx_solr.index.queue.indexConfigName {
    fields {
      somevalue_stringM = SOLR_MULTIVALUE
      somevalue_stringM {
           stdWrap.cObject = USER
           stdWrap.cObject.userFunc = Vendor\Ext\Classname->getValues
           separator=,
      }
    }
}
```

**How can i use a configuration from AdditionalConfiguration.php when i deploy my application on several instances?**

The configuration of the connection is done with typoscript. When you want to use a configuration from TYPO3_CONF_VARS or from the system environment, you can apply an stdWrap on the configuration that reads from these configurations.

The following example shows how a host can be configured in the AdditionalConfiguration.php and used in your typoscript to connect to solr:

The following line is added to AdditionalConfiguration.php

```
$GLOBALS['TYPO3_CONF_VARS']['EXTCONF']['solr']['host'] = 'mysolrserver.de';
```

To use this configuration for the host, you can use a TEXT element in the configuration and use override.data to use the value from the AdditionalConfiguration.php

```
plugin.tx_solr.solr {
   host = TEXT
   host {
     value = localhost
     override.data = global:TYPO3_CONF_VARS|EXTCONF|solr|host
   }
}
```

**How can I replace jQuery and/or jQuery UI versions or use different JavaScript library for searching field used by EXT:solr?**

You need to add following lines in your TypoScript setup:

```
plugin.tx_solr.solr {
    javascriptFiles {
        library = EXT:your_site_extension/Resources/JavaScript/JQuery/jquery.XYZ.
↪min.js
        ui = EXT:your_site_extension/Resources/JavaScript/JQuery/jquery-ui.XYZ.min.
↪js
    }
}
```

For more information please see *tx_solr.javascriptFiles*.

**I want to index extension records, what do i need to do?**

EXT:solr provides a flexible indexing for TYPO3 pages and records. You can add a custom indexing configuration for your own records with a valid TCA configuration.

You can read more about this in the section *IndexQueue Configuration*.

The following things are important:

- The extension ships several examples in the Folder "Configuration/TypoScript/Examples", read them and try to undestand them.
- EXT:solr can not know the business logic of an extension to generate a link to a detail view. You need to use typolink to build an url that points to a valid, existing detail page.
- When you index records, e.g. news it these records are indexed in solr and point to a news details page. That's the reason why it makes sence to exclude the news detail page from the normal page indexing. Otherwise the indexing of this page will produce an error message, because only a url with a valid news uid produces a valid output.

CHAPTER

THIRTEEN

# APPENDIX - DYNAMIC FIELDS

Dynamic fields allow you to add custom fields to Solr documents. That said, you never need to modify Solr's schema (which could cause problems or at least unnecessary additional work when updating the Solr extension). The following sections describe how to use dynamic fields with your Solr for TYPO3 installation. Usage of dynamic fields

You can use dynamic fields by following a special naming convention for document fields. E.g. to create a dynamic field that is a string the field name should end with _stringS. So if you want to create a field for storing a title you would name it title_stringS. We suggest you use lower camel case for the field name followed by an underscore followed by the dynamic field type "extension".

We've predefined the following dynamic fields:

| Extension | Type | Multivalue | Comment |
|---|---|---|---|
| *_stringS | String | No | |
| *_stringM | String | Yes | |
| *_boolS | Boolean | No | |
| *_boolM | Boolean | Yes | |
| *_intS | Integer | No | deprecated use _tIntS now |
| *_intM | Integer | Yes | deprecated use _tIntM now |
| *_sIntS | Sortable Integer | No | deprecated use _tIntS now |
| *_sIntM | Sortable Integer | Yes | deprecated use _tIntM now |
| *_tIntS | Trie Integer | No | |
| *_tIntM | Trie Integer | Yes | |
| *_longS | Long | No | deprecated use _tLongS now |
| *_longM | Long | Yes | deprecated use _tLongM now |
| *_sLongS | Sortable Long | No | deprecated use _tLongS now |
| *_sLongM | Sortable Long | Yes | deprecated use _tLongM now |
| *_tLongS | Trie Long | No | |
| *_tLongM | Trie Long | Yes | |
| *_floatS | Float | No | deprecated use _tFloatS now |
| *_floatM | Float | Yes | deprecated use _tFloatM now |
| *_sFloatS | Sortable Float | No | deprecated use _tFloatS now |
| *_sFloatM | Sortable Float | Yes | deprecated use _tFloatM now |
| *_tFloatS | Trie Float | No | |
| *_tFloatM | Trie Float | Yes | |
| *_doubleS | Double | No | deprecated use _tDoubleS now |
| *_doubleM | Double | Yes | deprecated use _tDoubleM now |
| *_sDoubleS | Sortable Double | No | deprecated use _tDoubleS now |
| *_sDoubleM | Sortable Double | Yes | deprecated use _tDoubleM now |
| *_tDoubleS | Trie Double | No | |
| *_tDoubleM | Trie Double | Yes | |
| *_tDouble4S | Trie Double with Precision Step 4 | No | |
| *_tDouble4M | Trie Double with Precision Step 4 | Yes | |

Continued on next page

Table 13.1 – continued from previous page

| Extension | Type | Multivalue | Comment |
|---|---|---|---|
| *_dateS | Date | No | deprecated use _tDateS now |
| *_dateM | Date | Yes | deprecated use _tDateM now |
| *_tDateS | Trie Date | No | |
| *_tDateM | Trie Date | Yes | |
| *_random | Random | No | |
| *_textS | Text | No | |
| *_textM | Text | Yes | |
| *_textTS | Text Tight | No | |
| *_textTM | Text Tight | Yes | |
| *_textSortS | Sortable Text | No | |
| *_textSortM | Sortable Text | Yes | |
| *_textWstS | Whitespace tokenized Text | No | |
| *_textWstM | Whitespace tokenized Text | Yes | |
| *_phoneticS | Phonetic | No | |
| *_phoneticM | Phonetic | Yes | |
| *_textEdgeNgramS | Edge Ngram (hello => hello, hell..) | No | |
| *_textEdgeNgramM | Edge Ngram (hello => hello, hell..) | Yes | |
| *_textNgramS | Ngram (hello => he,ll,lo,hel,llo) | No | |
| *_textNgramM | Ngram (hello => he,ll,lo,hel,llo) | Yes | |

CHAPTER

FOURTEEN

# APPENDIX - VERSION MATRIX

This is a list of EXT:solr versions and what versions of Apache Solr and TYPO3 are supported for each release.

| EXT: solr | Apache Solr | TYPO3 | Schema | Solrconfig | EXT:tika | EXT:solrfal | EXT:solrfluid | EXT:solrgroup | Apache cess-plugin |
|---|---|---|---|---|---|---|---|---|---|
| 3.1 | 4.10 | 6.2 - 7.6 | tx_solr-3-1-0–20150614 | tx_solr-3-1-0–20151012 | 2.0 | 2.1 | N/A | 1.1 | 1.3 |
| 4.0 | 4.10 | 7.6 | tx_solr-4-0-0–20160406 | tx_solr-4-0-0–20160406 | 2.1 | 3.0 | N/A | 1.2 | 1.3 |
| 5.0 | 4.10 | 7.6 | tx_solr-4-0-0–20160406 | tx_solr-4-0-0–20160406 | 2.1 | 3.1 | 1.0 | 1.2 | 1.3 |
| 5.1 | 4.10 | 7.6 | tx_solr-5-1-0–20160725 | tx_solr-4-0-0–20160406 | 2.1 | 3.2 | 1.2 | 1.2 | 1.3 |
| 6.0 | 6.3 | 7.6 | tx_solr-6-0-0–20161209 | tx_solr-6-0-0–20161122 | 2.2 | 4.0 | 1.2 | 1.3 | 1.7 |
| 6.1 | 6.3 | 7.6 - 8.x | tx_solr-6-1-0–20170206 | tx_solr-6-1-0–20161220 | 2.3 | 4.1 | 2.0 | 1.3 | 2.0 |