EXT: tollwerk Lucene search

Extension Key: tw_lucenesearch
Language: en
Version: 2.0.2
Keywords: lucene, search, index, forAdministrators, forDevelopers, forIntermediates, forAdvanced

In case you are reading this manual online at the TYPO3 website, we strongly recommend that you also visit the TYPO3 Extensions & Manuals page respectively the page about the tollwerk Lucene search TYPO3-Extension at our own website. We provide a PDF version of this manual there, which probably renders more nicely than the online version on typo3.org. (Sorry for our website currently being available in German language only. However, the PDF extension manuals are in English of course.)

For the most recent version of this extension always have a look at the corresponding GitHub repository. Please report any issues there as well.

Table of Contents

Introduction

What does it do?

This extension features a lightweight, intentionally simple but yet powerful implementation of the Apache Lucene Index as frontend index and search solution for TYPO3. It is based on extbase / fluid and built around **Zend_Search_Lucene**, which is part of the Zend Framework and offers a pure PHP implementation of the high-performance Apache Lucene Index. Having said this, the extension **doesn't impose any additional software requirements** (like a Java application server or an Apache Solr instance), everything you'll need is just already present if you successfully run TYPO3.

Compared to search frameworks relying on relational databases (like e.g. the well-known *indexed_search* extension in combination with MySQL as database storage) the Lucene Index is **file based** and offers a lot of advantages like e.g.

- **better performance than RDBMS / table based indices** (especially if the index grows larger),
- support for (also leading) **wildcard queries** with reasonable result **ordering by relevance / ranking**,
- proximity queries (aka "**fuzzy search**", so that typos might be tolerated)
- field based queries,
- and much more ...

Besides the implementation of the Lucene Index itself (and the resulting benefits) the extensions also offers a couple of interesting features like

- the ability to **index even uncached pages / page contents**,
- focused search on **subsections of the page tree**,
- search within pages of a specified language only,
- **search term highlighting** at acceptable performance costs (usually a huge problem with the standard Zend_Search_Lucene highlighter),
- view helpers for easily altering e.g. the page title and modification timestamp from within any fluid template,
- fine tuning the importance of certain page properties (like title, keywords etc.) during indexing and searching ("boosting"),
- **custom rewrite mechanisms** for internally altering search queries (possibly interesting if the topic of the website is somewhat special and the search terms you expect are special as well, e.g. if they contain punctuation, digits or complex character combinations),
- and some more ...

Partially in contrast to other TYPO3 extensions built around the Lucene Index this one tries to

- reduce the setup and configuration costs to an absolute minimum,
- use the most recent TYPO3 techniques like extbase / fluid
- and finally find a reasonable subset of Lucene Index features that should be supported and implemented in a first step.

In fact, the main motivation for writing this extension was the impression that the existing extensions with a similar approach either

- seemed to be a little outdated (e.g. haven't been built on extbase / fluid or haven't been updated for a couple of years),
- didn't quite do what the author needed for his projects or

- required way to much effort to get them up and running.

However, some of them have been a valuable inspiration, and there are also some very interesting articles on the web, e.g. (sorry, German only)

- http://www.j2h.com/publications/die-indexed-search-alternative-lucene-typo3-integrieren
- http://blog.marit.ag/2008/09/09/typo3-und-die-lucene-suche-von-zend/

Thanks to the authors!

To learn more about the Apache Lucene Index itself please visit http://lucene.apache.org.

Finally it's worth mentioning that you might also be interested in **Apache Solr**, which is a dedicated enterprise search server also based on the Apache Lucene Index (with even a lot more great features). For detailed information on Apache Solr please visit http://lucene.apache.org/solr. You will need a Java Application Server for running Solr (which is probably the most crucial requirement compared to this extension), but if this isn't a problem for you, you should definitely check out the **Apache Solr extension for TYPO3** – another great piece of software, to be found at http://www.typo3-solr.com and in many respects the most comprehensive way of searching and finding with TYPO3.

Having said all this, it should be emphasized that the extension is still at it's very beginning and that there are plenty of features that haven't been implemented yet. Please see the To-Do list for some thoughts on this. So far, the functionality has been sufficient for the author, but if there's enough public demand the extension will be further improved. So please let us know what you think!

Screenshots

The frontend plugin can render a simple and lightweight search box that is optionally equipped with some JavaScript functionality for displaying a default text while the input is not focused. The search box plugin can also be included via TypoScript.

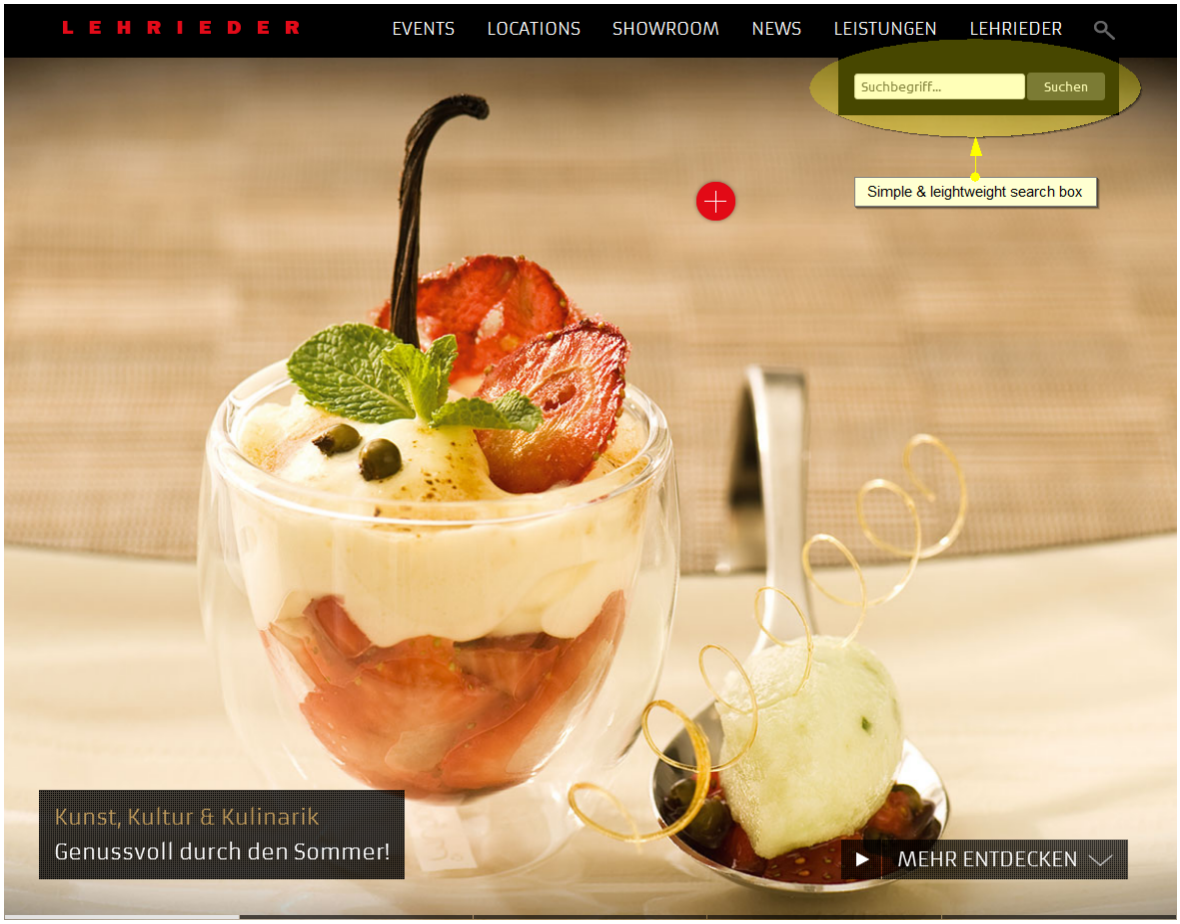


Figure 1: Simple & lightweight search box

The search box and also the search results are rendered by fluid templates, so the appearance and the source code can completely be customized. The search results support **search term highlighting** (via a custom view helper) and **pagination** (using the standard fluid "*widget.paginate*").

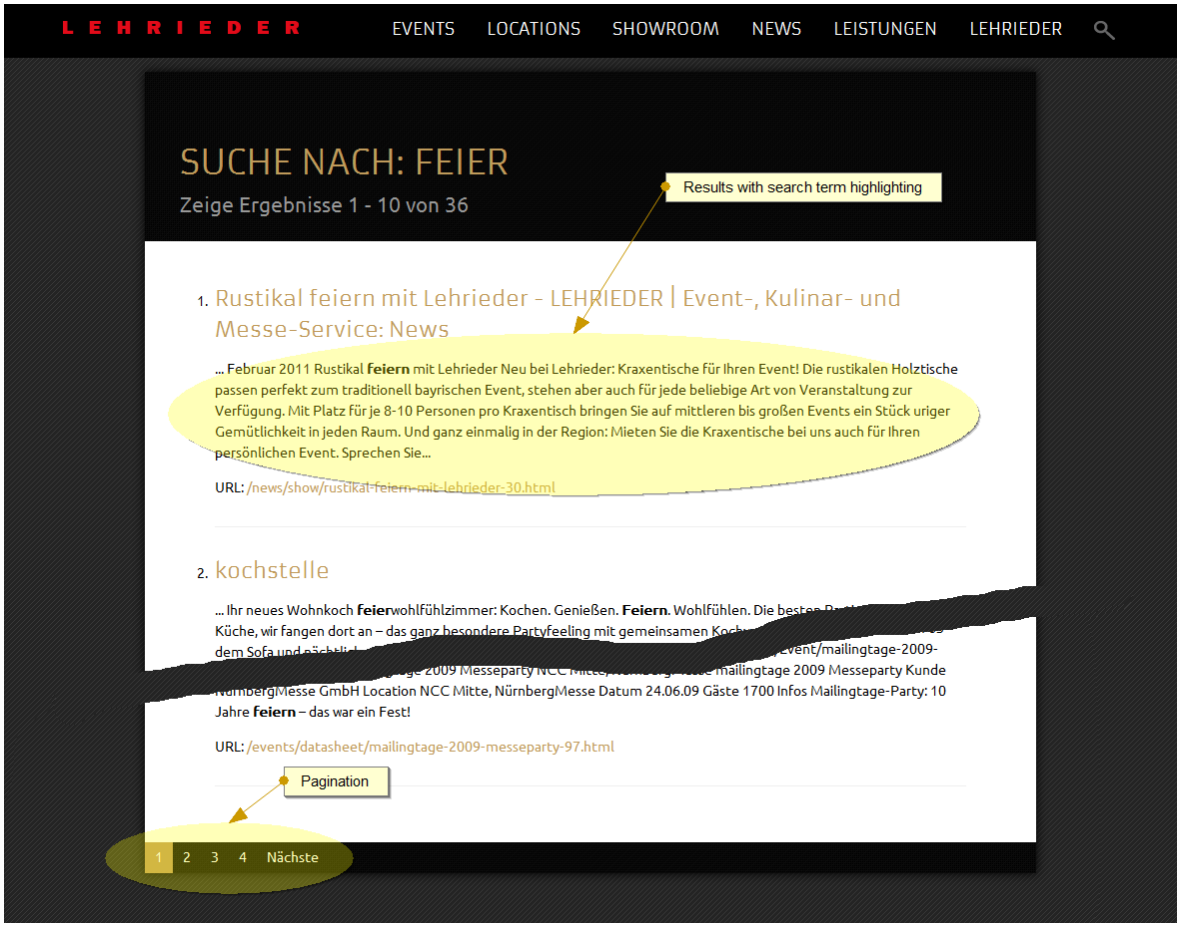


Figure 2: Regular search results with search term highlighting and pagination

There's an alternative fluid template for rendering the search results in "404 search mode", which is – of course – also fully customizable.

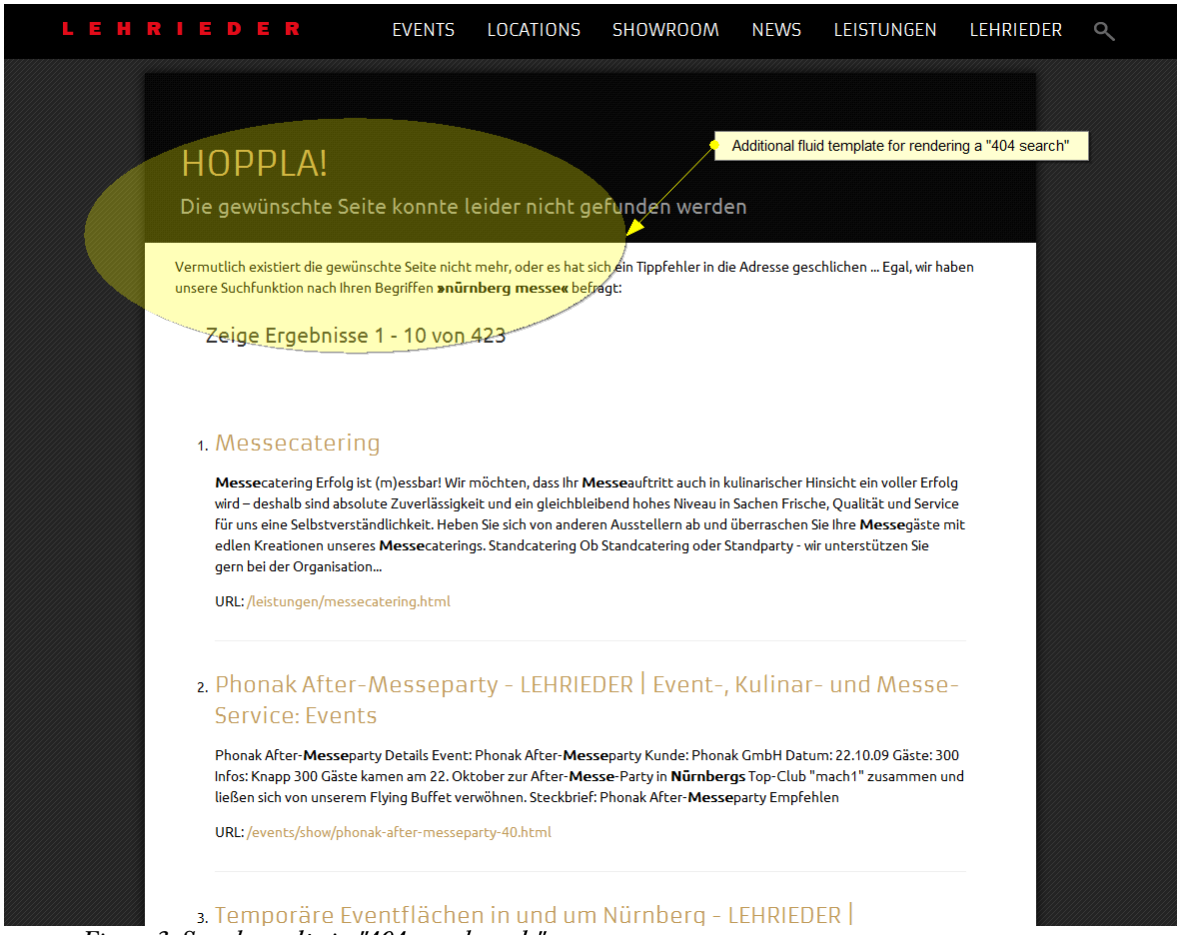


Figure 3: Search results in "404 search mode"

Installation

To install the extension simply download it from the TYPO3 extension repository and enable it in the Extension Manager. The extension doesn't expect any database changes.

Extension configuration

There are 3 settings to be configured via the Extension Manager:

Lucene Index directory

The Lucene Index is – in contrast to some other search solutions for TYPO3 like the popular *indexed_search* extension – **file based** rather than stored in a database. The standard location where the index is stored is the directory

```
/typo3temp/tw_lucenesearch
```

You can change this to your needs, but please be aware, that you will have to create the directory manually if you choose a different path than the default one.

Lucene Index MergeFactor

The index MergeFactor controls the frequency of index optimization. An index is optimized if it doesn't consist of too many segments (i.e. isn't distributed over a lot of single files on your hard drive). The following can be found in the original Zend_Search_Lucene documentation (http://framework.zend.com/manual/en/zend.search.lucene.index-creation.html#zend.search.lucene.index-creation.optimization.mergefactor):

```
MergeFactor determines how often segment indices are merged by addDocument(). With smaller
values, less RAM is used while indexing, and searches on unoptimized indices are faster, but
indexing speed is slower. With larger values, more RAM is used during indexing, and while
searches on unoptimized indices are slower, indexing is faster. Thus larger values (> 10)
are best for batch index creation, and smaller values (< 10) for indices that are
interactively maintained.
MergeFactor is a good estimation for average number of segments merged by one auto-
optimization pass. Too large values produce large number of segments while they are not
merged into new one. It may be a cause of "failed to open stream: Too many open files" error
message. This limitation is system dependent.
```

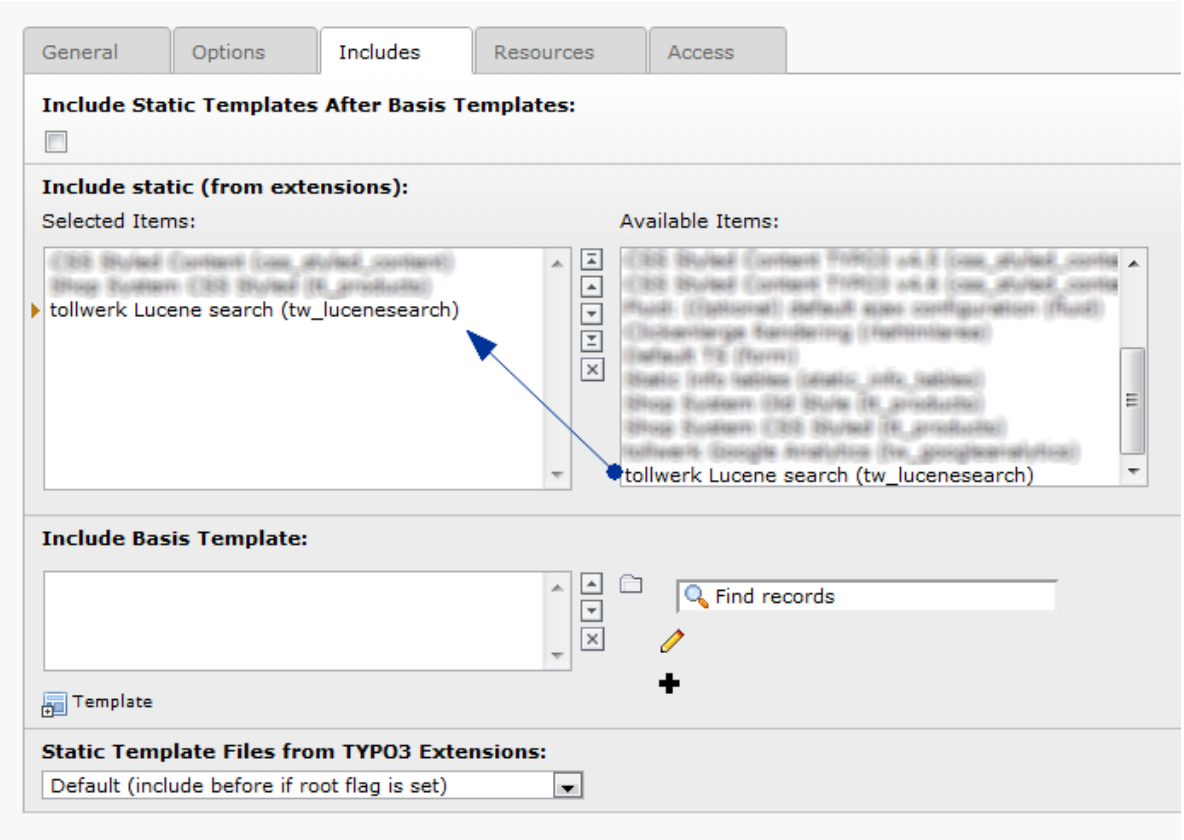
The default value for MergeFactor is 10, which might be a suitable value for most setups. Furthermore, the above text mentions a potential problem you run into regarding your operating system's limitation on open files. Please see the known problems for this. You might want to experiment with the MergeFactor setting if you get into trouble with the “Too many open files” error.

Enable developer tools

There are some developer tools that might become handy while working on your index configuration. Please see the dedicated chapter below for more information. You have to enable the developer tools here in the Extension Manager. It is recommended that they are disabled in production use though.

Static TypoScript

Include the extension's static TypoScript into the TypoScript root template of your site.



Note: The plugin's static TypoScript assumes that your main page object is called “**page**”. If this is not the case, please don't include this file. Instead just copy it's contents (EXT:tw_lucenesearch/Configuration/TypoScript/setup.txt) into a custom TypoScript template that is part of your setup and follow the comments in the template.

Configuration

Constants

All features of the extension can be controlled via the constant editor (e.g. on the TypoScript root template). Almost any constant directly relates to a corresponding setup option that can also be set by manually crafted TypoScript. Please see the setup chapter for details on these options.

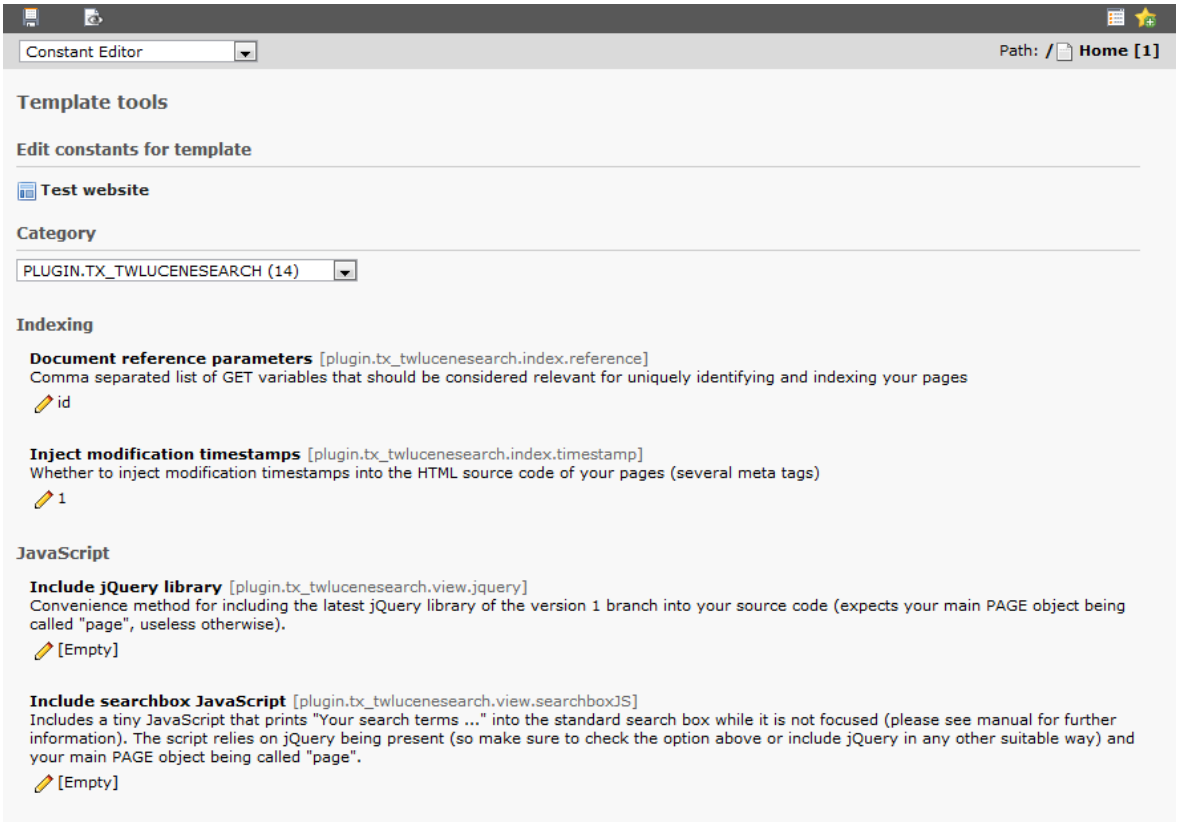


Figure 4: Constants (part 1)

Note: Both JavaScript constants shown in the figure try to act to your convenience, but rely on the fact that your main PAGE object is called “**page**”. If this is not true, then enabling these options will have no effect and you will have to find alternative ways to achieve the same results (please see EXT:tw_lucenesearch/Configuration/TypoScript/setup.txt for inspiration).

Include jQuery library

This option will addthe latest jQuery library (of the version 1 branch) to your included JavaScripts, delivered via http://ajax.googleapis.com. jQuery is required for the following option to work.

Include searchbox JavaScript

This option will include another small JavaScript, which will affect the search boxes you place using the extension's frontend plugin. When this script is included, the search boxes will feature the value “Your search terms ...” along with the CSS class name “default” as long as they are not currently focused by the user. When focused, the input fields will be cleared and the CSS class name will be removed.

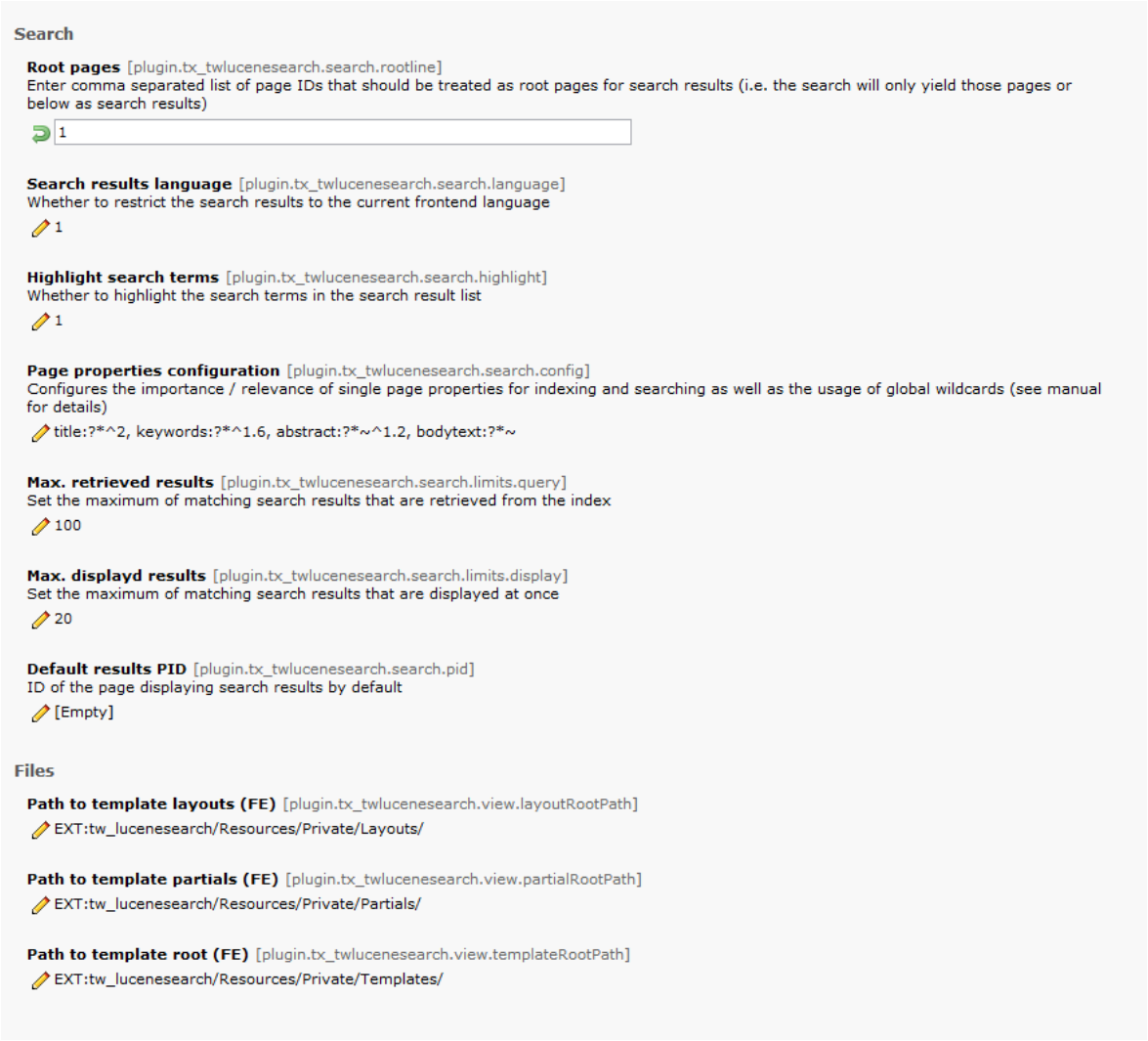


Figure 5: Constants (part 2)

Setup

Frontend plugin setup

There are several TypoScript setup options controlling the behavior of the frontend plugin:

Property:	Data type:	Description:	Default:
view.templateRootPath	dir	Root path for the **fluid templates** of the plugin. The plugin has just one controller (named “Lucene”) with three actions (named “search”, “results” and “notfound”). Accordingly there have to be three fluid templates at the following locations (relative to the template root path): • *Lucene/Search.html* • *Lucene/Results.html* • *Lucene/Notfound.html*	{$plugin.tx_twlucenesearch.view.templateRootPath}
view.partialRootPath	dir	Root path for the **fluid partials** of the plugin. The default fluid templates are using one partial named “Searchbox.html” for rendering the simple search box. If you override the default templates this might become irrelevant.	{$plugin.tx_twlucenesearch.view.partialRootPath}

Property:	Data type:	Description:	Default:
view.layoutRootPath	dir	Root path for the **fluid layouts** of the plugin (not used by default)	{$plugin.tx_twlucenesearch.view.layoutRootPath}
view.widget.Tx_Fluid_ViewHelpers_Widget_PaginateViewHelper.templateRootPath	dir	Root path for the **fluid templates** for widgets used by the plugin. By default the plugin uses a slightly modified version of the standard fluid **paginate widget** for displaying the search results. The modified version supports direct links to the result pages and offers localization support. The corresponding template is located at *ViewHelpers/Widget/Paginate/Index.html* (relative to the template root path).	{$plugin.tx_twlucenesearch.view.templateRootPath}
settings.defaultResultsPage	int	ID of the **default page** that should be used for displaying search results. If not given, the page sending the search request will be used.	{$plugin.tx_twlucenesearch.settings.pid}

[tsref:plugin.tx_twlucenesearch]

Indexer setup

The indexer component – in charge of indexing and storing pages to the Lucene Index – also introduces some TypoScript settings, which belong to the "*config*" namespace:

Property:	Data type:	Description:	Default:
index_enable	boolean	Enables pages to be indexed. This is a standard TYPO3 setting, also see the CONFIG object in the core TypoScript documentation. You definitely have to enable this for indexing to work at all. This setting defaults to false.	false

index_injectTimestamp	boolean	Enables the **injection of modification timestamp meta tags** into the HTML output. The following meta elements will be written to the source code of all pages (with varying timestamp of course):
		
		`<meta name="DC.Date" content="2012-07-23T14:12:06+02:00"/>`
		`<meta name="Date" content="2012-07-23T14:12:06+02:00"/>`
		`<meta name="Last-Modified" content="2012-07-23T14:12:06+02:00"/>`
		
		In general, these meta tags are absolutely harmless and *could* tell search engines if a page has changed recently. However, the author doesn't know if any of the search engines does really respect one of these timestamp tags though.
		
		The indexer uses the modification timestamp meta tags as one of several possible indicators for detecting whether a page has to be re-indexed or whether the index still is up to date. Please see the chapter about the indexing process for details on this process.
		
		This setting defaults to true.
			{$plugin.tx_twlucenesearch.index.timestamp}

Property:	Data type:	Description:	Default:

index_reference	string	This is the list of GET parameters that jointly build up the **unique page references** used as keys for storing (and retrieving) indexable page contents ("index documents") to the search index. These are also the GET parameters that are ultimately necessary for (re-)constructing working URLs for the indexed pages (some common parameters are not absolutely necessary and should be omitted in this list, like e.g. "*cHash*", "*no_cache*" etc.). Please enter a comma separated list here. Don't worry to include GET parameters that are optional: If a parameter is not present when the page reference string is built, and if there is no default value specified (see below), then the parameter will simply be omitted.
		
		The default value of this setting is the page ID parameter:
		
		`config.index_reference = id`
		
		For multilingual sites you may want to include the language parameter as well:
		
		`config.index_reference = id,L`
		
		In case the values of a parameter are constrained to either certain predefined options or an integer value range (or a combination of both), you can (and should) define this as well.
		
		Examples:
		`config.index_reference = L(0|1|2)`
		`config.index_reference = L(0-6)`
		`config.index_reference = xyz(one|two|three)`
		`config.index_reference = ABC(1-9|none)`
		
		Also, you can define a default value that will be used if a parameter is not presentat index time (would be skipped otherwise):
		
		`config.index_reference = L(0-2)=0`
		
		Be aware that the relevant reference parameters might alter from page to page, so you will have to adapt this setting accordingly. Imagine e.g. a website subsection dealing with news. You may want to alter the reference parameters just for this very section like this:
		
		`config.index_reference = id,L(0-2)=0,tx_ttnews[tt_news]`
		
		Note: As a rule of thumb, always keep the list of necessary parameters as short as possible.
			{$plugin.tx_twlucenesearch.index.reference}

[tsref:config]

Search setup

Finally there are some TypoScript settings controlling the search process itself, also belonging to the "*config*" namespace.

Property:	Data type:	Description:	Default:
search_lucene.restrictByRootlinePids	string	Restricts the search results to pages that are **descendants of certain root pages** (or the root pages themselves). This way you can easily restrict a search to a certain portion of your page tree (or the combination of several branches). Furthermore, this setting is quite essential in a multi-domain setup, as you surely want to separate the search results by domain ... Enter a comma separated list of root page IDs. **Example**: `config.search_lucene.restrictByRootlinePids = 1,2` This settings defaults to an empty string.	{$plugin.tx_twlucenesearch.search.rootline}
search_lucene.restrictByLanguage	boolean	Restricts the search results to the **current frontend language**. If enabled, a visitor only gets search results matching the currently active frontend language. Obviously this setting only makes sense for multilingual sites. This setting defaults to true.	{$plugin.tx_twlucenesearch.search.language}

Property:	Data type:	Description:	Default:

search_lucene.searchConfig	string	This setting is quite essential. It controls which **index document fields** are queried when searching, how relevant / important a single field is for ranking / ordering the results and whether you want to run wildcard or fuzzy searches on these fields. Enter a comma separated list of field definitions here. Each field definition must have the format
		
		`<field name>:<search template>[<fuzzy flag>] [<boost factor>]`
		
		<field name> has to be one of the following values. These are the properties of every document in the Lucene Index (the fields in parentheses are mainly intended for internal use and should not be used for search queries):
		• title
		• bodytext
		• keywords
		• abstract
		• language
		• (type)
		• (reference)
		• (rootline)
		• (timestamp)
		
		The **<search template>** is a template string used for building the final query submitted to the search service. There has to be a question mark "?" in this string, which will be substituted by the search terms. A single "?" is the most simple search template possible. Additionally you can prepend or append **wildcards** "*" or constant strings to the search template. Examples:
		
		`?*`
		`*?*`
		`constant-prefix-?`
		
		The optional **<fuzzy flag>** simply is a tilde sign "~". If present, a fuzzy search will be performed for the given search template. That means that even matches with slightly different spelling might be found, thus compensating potential typos. A fuzzy search for "roam" could for example also match "roams" as well as "foam".
		
		Finally, the optional **<boost factor>** controls the importance of a match for the overall ranking of a single search result. E.g. it makes sense to consider a search term match in the title of a document a little bit more significant than a match in the bodytext field. By using a query level boost factor these matches can be influenced so that they are treated more relevant from the ordering point of view. By default, each field definition has a boost factor of 1, meaning "normal" (you can omit the **<boost factor>** component if you don't want to change this). You can lower the boost factor below 1 (meaning "less important") or raise it appropriately. The boost factor has to be denoted after a leading "^" sign.
		
		The searchConfig setting defaults to:
		`title:?*^2, keywords:?*^1.6, abstract:?*~^1.2, bodytext:?*~`
		
		Explained in words: The search will be performed on the fields "title", "keywords", "abstract" and "bodytext", each with a global trailing wildcard. The matches in the field "abstract" and "bodytext" will be fuzzy. The most important match will be a "title" match (with boost factor 2), the least important a "bodytext" match (with normal boost factor, i.e. 1).
		
		Attention: Entering no or an invalid value for this setting will render the whole search completely useless!
			{$plugin.tx_twlucenesearch.search.config}

Property:	Data type:	Description:	Default:
search_lucene.limits.query	int	Limits the **maximum number of internal search term matches** for wildcard and fuzzy search queries. Those queries may match too many terms, causing incredible search performance downgrade. Limiting the max. number of matches will help. Set this to zero to suspend all limitations. This setting defaults to 100.	{$plugin.tx_twlucenesearch.search.limits.query}
search_lucene.limits.display	int	Limits the **maximum number of search results per result page**. If used with the default fluid templates the frontend plugin will render a pagination widget as soon as the number of search results exceeds this value. This setting defaults to 20.	{$plugin.tx_twlucenesearch.search.limits.display}
search_lucene.highlightMatches	boolean	Enables **search term highlighting** in the search results. Highlighting can be a complicated task, especially with fuzzy or wildcard searches. The original highlighter implementation of Zend_Search_Lucene can lead to tremendous performance downgrade. This extension uses it's own improved technique for highlighting, which might still impact performance, but is way faster than the Zend version. This setting defaults to true.	{$plugin.tx_twlucenesearch.search.highlight}

[tsref:config]

Usage

About indexing

From this extension's point of view indexing pages is a pretty straight forward process. It uses two standard hooks ("*contentPostProc-output*" for non-cached pages, "*contentPostProc-all*" for cached pages) to intercept content output and do the indexing just immediately before a page gets delivered. In contrast to several other search extensions and their indexing approaches this technique allows for **even non-cached pages to be indexed**.

When a page is about to be sent to a visitor, the indexer first decides **if indexing should happen at all** (as defined by the *config.index_enable* TypoScript setting and the no_search flag aka disabled "Include in Search" option of this very page). If indexing should occur, the indexer will further analyze the source code of the page in order to find out, if the page has already been indexed and / or if the indexed version needs to be updated.

For the purpose of querying (and storing a document to) the index a unique page reference gets built, reliably identify a certain page. It depends on the GET parameters defined by the *config.index_reference* setting (usually at least the *id* parameter should be member of this list, but also e.g. language and extension parameters could be contained). By querying the index for the reference key of the current page the indexer can determine if the page has ever been indexed before.

Knowing the indexing timestamp of the previous version (if any) the indexer examines several indicators (in the following order) to find the **last modification timestamp** of the current page:

- At first, the indexer looks for the presence of **modification timestamp meta tags** in the HTML source code (*DC.Date, Date, Last-Modified*) and extracts the first occurrence of these if available.
- Secondly, if no suitable meta tags are present, it will look at the *tstamp* property of the current frontend page (*$GLOBALS['TSFE']->page['tstamp']*), which indicates the last **update of the page record** itself. This value also gets updated automatically if any content element on the page is modified.
- If still no non-empty timestamp has been found, the indexer will look for an *index_timestamp* property on the global frontend engine object (*$GLOBALS['TSFE']->index_timestamp*) and use this one if available. This property is not a native property of the frontend engine, but gets introduced by one of the extension's view helpers (tw:index.timestamp) and can theoretically be used by any other extension to indicate a **custom modification date** of the page. This is quite handy if you e.g. display news articles on a page and want the modification timestamp for each article page to depend on the article itself (instead of the page's timestamp or the news plugin residing on it).
- Finally, if none of the above options apply, the **current timestamp** will be used, which generally means that the page gets (re-)indexed by all means.

If the page has never been indexed before, or if the indexed version needs to be updated, then the indexer will go ahead and **extract the indexable contents** out of the HTML source. There are several (text) properties of a page that are considered indexable:

- title
- bodytext
- keywords
- abstract

The **title of the page** will be determined by the first matching option of the following (in this order):

- If there's a non-empty *indexedDocTitle* property of the global frontend engine, it will be used for

indexing (*$GLOBALS['TSFE']->indexedDocTitle*).

- If there's a non-empty *<title>* element in the source code, it's contents will be used for indexing.
- If the global frontend engine has a non-empty *altPageTitle* property, it will be used for indexing (*$GLOBALS['TSFE']->altPageTitle*).
- If the current page record has a non-empty *title* property, it will be used for indexing (*$GLOBALS['TSFE']->page['title']*).
- If none of the above apply, the title will be an empty string.

The **bodytext** value will be obtained by extracting and concatenating all text nodes from the page's HTML source. Furthermore, the well known comment tags commonly used by e.g. the *indexed_seach* extension for marking indexable sections will be respected as well. That means, if at least one instance of the *<!--TYPO3SEARCH_begin-->* comment marker is part of the source code, then the text contents will be analyzed further. Only those text parts will get indexed, that are surrounded by the following marker combination:

```
...
This is text that will not become indexed.
...

<!--TYPO3SEARCH_begin-->
...
Now an indexable section has begun and this part of the text will be indexed.
...
<!--TYPO3SEARCH_end-->

...
The indexable section has ended and this part of the text will not get indexed anymore.
...
```

It is up to you to spread those markers over your HTML output appropriately (by inserting these comments into your HTML templates). It is quite common to exclude e.g. the main and sub navigation from indexing as they are present on every page without contributing really meaningful information from the search point of view.

Finally, both the **keywords** and the **abstract** values will preferably be obtained from the appropriate HTML meta tags out of the source code. If these are missing or empty, the indexer will fallback to the corresponding page record properties (*$GLOBALS['TSFE']->page['keywords']* and *$GLOBALS['TSFE']->page['abstract']* respectively).

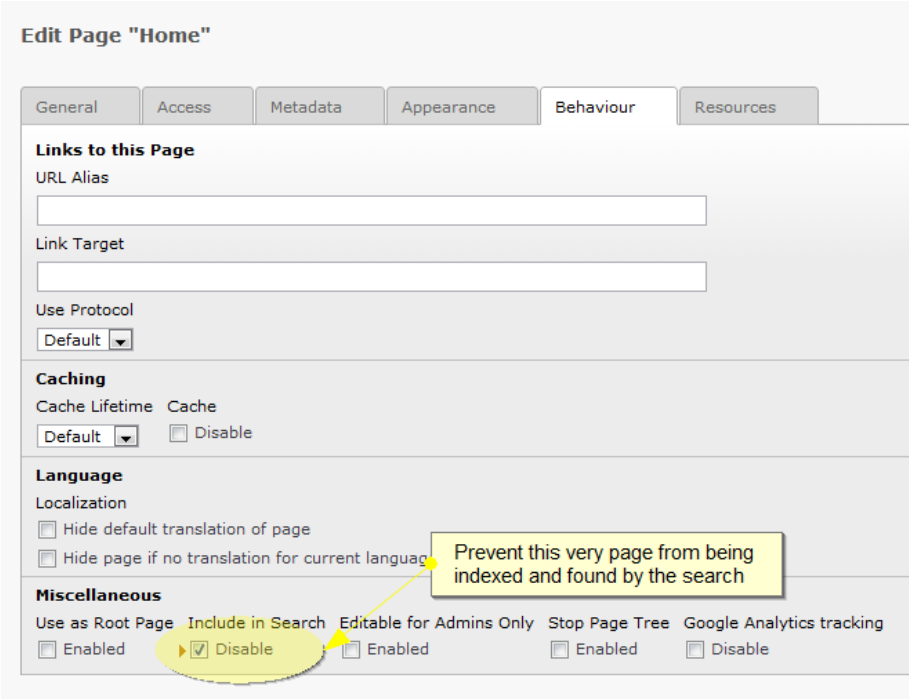
When the indexer has extracted the indexable contents out of the page, a so called **index document** is created and stored to the index (using the the page reference as unique key, see above), along with these additional properties:

- *language* (e.g. “en”)
- *type* (currently only “0” is supported, meaning “TYPO3 page”)
- *reference* (like a primary key, for uniquely identifying the index document)
- *rootline* (space separated list of page IDs belonging to the rootline of the indexed page)
- *timestamp* (last modification timestamps)

The newly constituted document is handed over to the **index service** being in charge of managing the Lucene Index. As soon as the document has been stored, it will be accessible for search queries.

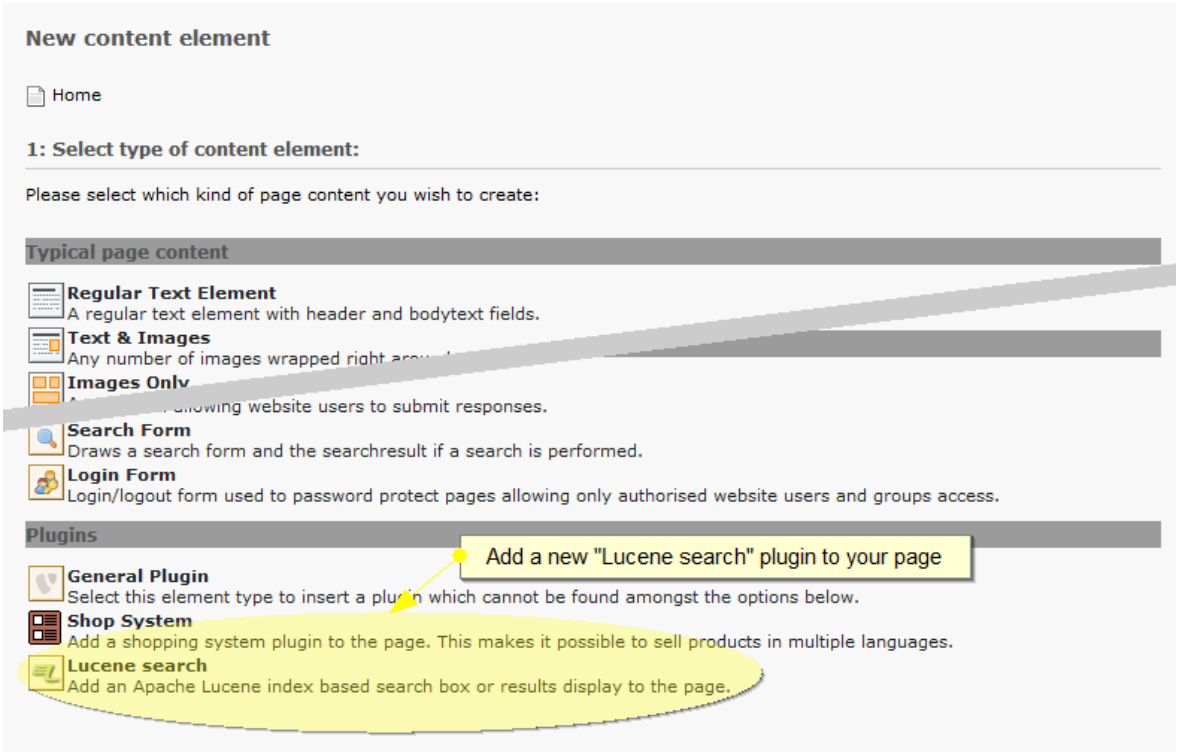
Disable indexing for single pages

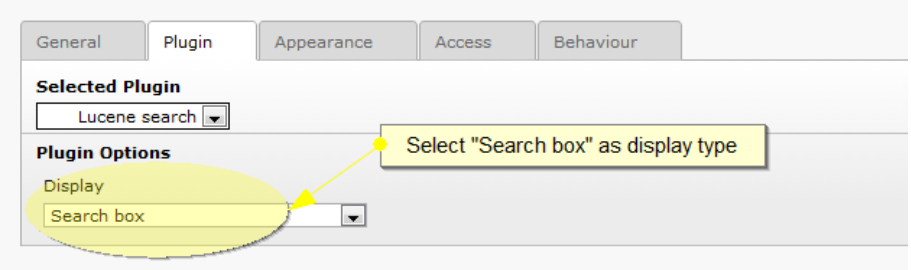
If you want to exclude a single page from being indexed (respectively from being found by the search), you can set the *no_search* flag in the page's properties:



Placing a search box

Probably the easiest way to include a search box on your website is to put a “Lucene search” plugin on your page and select “Search box” as display type:





By default the plugin will render the search box using the fluid template *Lucene/Search.html* (relative to the template root directory set with the *view.templateRootPath* setting). The output of this template is simple and looks somewhat like this:

```
<!--TYPO3SEARCH_end-->
<form class="tx-twlucenesearch-searchform" action="index.php?id=1" method="GET">
      <input name="tx_twlucenesearch_lucene[searchterm]" class="tx-twlucenesearch-sword
        default" type="text" value="Your search terms ..." title="Your search terms ..." />
      <input name="tx_twlucenesearch_lucene[search]" class="tx-twlucenesearch-search"
        type="submit" value="Search" />
</form>
<!--TYPO3SEARCH_begin-->
```

There are no default CSS styles applied, so styling the search box is completely up to you. In case you enabled the “Include searchbox JavaScript” constant (in the constant editor), a simple JavaScript will be included that prints something like “Your search terms here ...” into the search box as long as it's empty and not focused by the visitor. Furthermore, while in this state the search box will carry the CSS class name “default”, so you have full styling control as well.

Secondly, you can also insert the search box via TypoScript:

```
10 = USER_INT
10 {
      userFunc = tx_extbase_core_bootstrap->run
      settings < plugin.tx_twlucenesearch.settings
      persistence < plugin.tx_twlucenesearch.persistence
      view < plugin.tx_twlucenesearch.view
      pluginName = Lucene
      extensionName = TwLucenesearch
      controller = Lucene
      action = search
      switchableControllerActions{
            Lucene {
                  1 = search
            }
      }
}
```

If you use one of the above methods, the search box will potentially pick up a previously submitted search term and display it. That means, if you enter a search term into the box (in the frontend of course), submit the search form and the target page does also feature a search box, the submitted search term will be displayed there from the beginning.

The form containing the search box will send the search request to the page with the ID that you specified by the *settings.defaultResultsPage* setting. Be sure to set this appropriately and place a search results plugin on the target page.

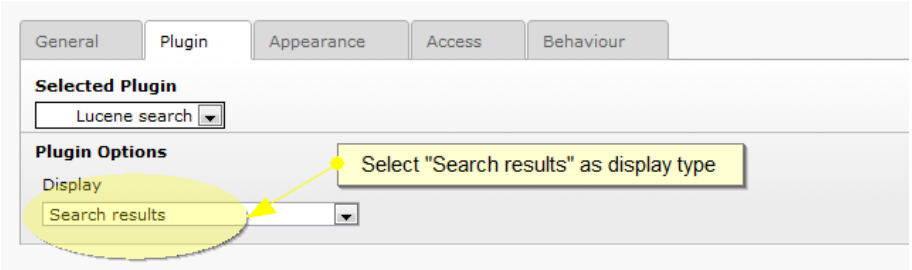
For the sake of completeness it should be mentioned that you can also insert the search box by simply **rendering the corresponding fluid template via TypoScript**. This way you can even control the target page by passing the in variable *page*. But please make sure that you use this insertion style only in an uncached context (e.g. as part of a *USER_INT* object), otherwise you might experience strange behavior with the display of a potentially picked up search term.

```
10 = FLUIDTEMPLATE
10 {
      file = EXT:tw_lucenesearch/Resources/Private/Templates/Lucene/Search.html
```

```
        partialRootPath = EXT:tw_lucenesearch/Resources/Private/Partials
        variables {
            page = TEXT
            page.value = 123
            searchterm = TEXT
            searchterm.data = GP:tx_twlucenesearch_lucene|searchterm
        }
}
```

Displaying search results

Displaying search results is as easy as including the search box: Again, put a “Lucene Search” plugin on your page and choose “Search results” as display type.



By default the search results are rendered with the fluid template *Lucene/Results.html* (relative to the template root directory set with the *view.templateRootPath* setting) and mainly consist of

- a header section re-displaying the terms that have been searched for, as well as the result total,
- an ordered list of search results (with highlighted search terms if you enabled this via the *config.search_lucene.highlightMatches* setting) including links to the respective result pages,
- an optional result pagination rendered automatically if the number of results exceed the *config.search_lucene.limits.display* setting (by default the pagination is rendered using the standard fluid paginator widget).

404 search mode

As a special variant you can include the plugin in “404 search mode”. The plugin will then try to detect if your visitor came from a search engine and requested a non-existing page (normally he would get a “404 Page not found” error in these cases).

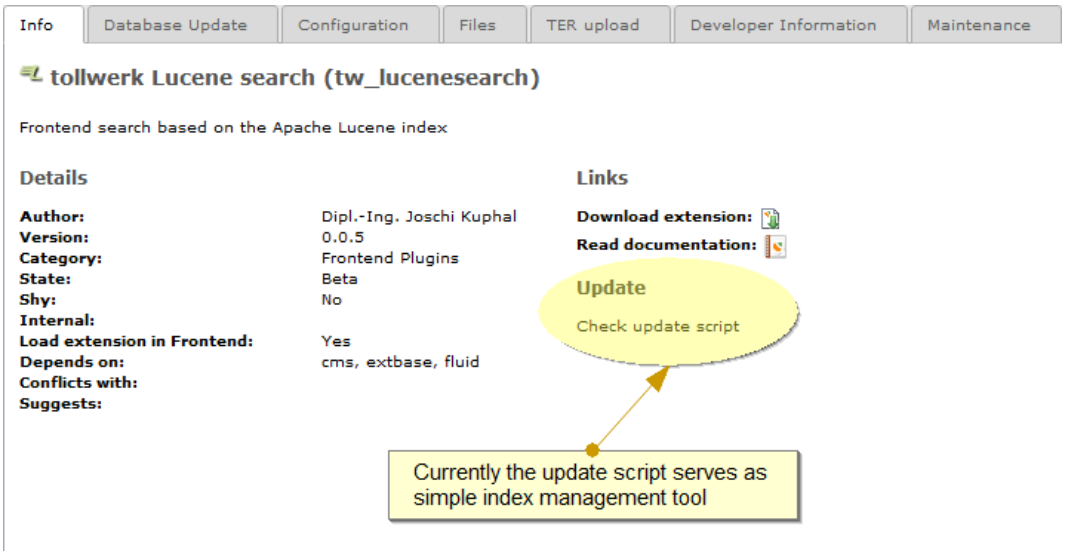
If the plugin is able to extract some search keywords used by the visitor at the search engine (by examining the referrer URL), it will trigger a local site search with these keywords and display the results. This way the visitor might still have luck and find the page he was looking for. If the plugin cannot extract any search keywords, it will render a standard search form (or whatever you customize the corresponding fluid template to) as a fallback.

Unfortunately Google is gradually switching all it's services to SSL, omitting the search keywords from the URL, so this feature doesn't seem to work with Google any longer ...

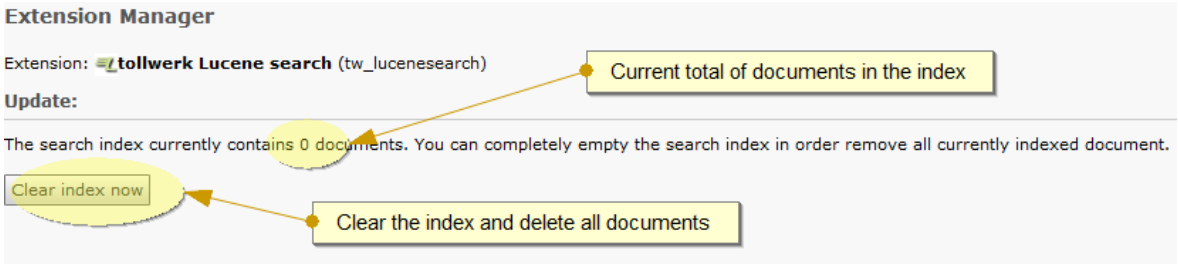
Advanced techniques

Clearing the index

Currently the extension's update script serves as very simple management tool for the Lucene Index. You can call it via the Extension Manager:



The script shows the current total of indexed documents and let's you clear the whole index (delete all existing index documents):



A future version of the extension could feature a dedicated backend module for managing the index. There could be a selective deletion of index documents or extended index statistics. Truly, the update script is definitely not the perfect location for stuff like this ...

View helpers

The extension comes with four custom view helpers that you can use in your fluid templates. If you want to do so, you have to register the appropriate **view helper namespace** in the beginning of your templates, e.g. like this:

```
{namespace tw=Tx_TwLucenesearch_ViewHelpers}

<!--TYPO3SEARCH_end-->
<f:if condition="{error}">
    ...
</f:if>
<!--TYPO3SEARCH_begin-->
```

tw:array.key

This view helper returns the **key of an array** at a certain (zero-based) index / position. As an example, the view helper call

```
<tw:array.key array="{one: 'first', two: 'second', three: 'third'}" position="1" />
```

would return the string

```
two
```

, as “two” is the key at position / index 1 inside the array.

tw:index.timestamp

With the help of the *index.timestamp* view helper you can control the timestamp that the indexer will use as **modification timestamp of the current page**. This is especially useful with hub pages like e.g. a single news display page. Most likely you will want to use the news article's timestamp as modification time (instead of the rendering page's one):

```
<twlucene:index.timestamp timestamp="{article.tstamp}"/>
```

Internally the view helper will set the *index_timestamp* property of the global frontend engine (*$GLOBALS['TSFE']->index_timestamp*), which will be picked up by the indexer later on.

tw:index.title

The *index.title* view helper can be used to **set the title of the current page**, which also impacts the document title used for indexing. The title you set will be injected into the HTML source code output of the current page as *<title>* element (replacing an already existing one if available).

```
<twlucene:index.title title="{article.title}" format="%C - %S: %P"/>
```

As you can see in the example, the *format* parameter is used as a template string for rendering the title content, supporting three substitution placeholders:

- **“%C”** will be replaced with your **custom title** provided via the *title* attribute
- **“%S”** will be replaced with the global **site title** (as defined e.g. by your TypoScript root template)
- **“%P”** will be replaced with the regular **page title** of the current page (as defined by the page's title property)

The new page title will be used in (and written to) several locations:

- As *<title>* element for the HTML source code output
- As explicit page title for indexing (*$GLOBALS['TSFE']->indexedDocTitle*)
- As title of the current frontend page record (*$GLOBALS['TSFE']->page['title']*), so that it is available to other consuming applications as well

tw:search.highlight

This view helper takes care of the **search term highlighting** in search result lists. It can also be used to crop a search result text to a certain length. The arguments are all optional and defined as follows:

- **text**: This is the text that should be given back cropped and / or with highlighted search terms. If not given as an attribute of the view helper element, the rendered content of the element will be used (if not in shortcut notation).
- **search**: This argument represents the search terms to be highlighted (if any) and may be of one of the following formats:
  - Array of literal search terms
  - A query hits object (which is the result of a search query; instance of the PHP class *Tx_TwLucenesearch_Domain_Model_QueryHits*)
  - A search query object (which is what the default fluid templates use, so please see there for an example; instance of the PHP class *Zend_Search_Lucene_Search_Query*)
  - A string as you would literally enter it into a search box
- Search term highlighting only happens if this argument expresses some reasonable search terms.
- **crop**: If given, this argument denotes the number of characters the resulting string may have in total (including an optional prefix and / or suffix). A string will preferably be cropped at it's end. If search term highlighting is active and the string has to be cropped by at least one third (33%), then it might also be cropped in it's beginning, keeping the last three words before the first occurrence of the first search term match within the string. This sounds complicated, but the result is pretty much what you expect as a regular human visitor anyway, so don't worry too much about the cropping logic ... The argument is empty by default (no cropping will occur).
- **append**: This is the string that gets appended as a suffix if the string has to be cropped at it's end. Defaults to " ...".
- **prepend**: This is the string that gets prepended as a prefix if the string has to be cropped at it's beginning. Defaults to "... ".
- **bodytext**: This argument tells the highlighter from which index document field the value of the text parameter has been drawn (if no search term highlighting is used, this argument simply doesn't matter). The originating field is important as the internal search definition may differ from field to field. You might e.g. use a fuzzy search for the *bodytext* field, whereas the *title* field requires an exact match, so the highlighter has to work differently for the two fields ... Mostly you will want to highlight search results extracted from the *bodytext* field (which is the default for this argument anyway).

Search term rewrite hooks

There are some situations where it might be necessary (or just reasonable) to pre-process your visitors' search terms, maybe depending on your site's topic. Imagine e.g. the situation that you expect your visitors to search for metric lengths. Some people will enter a space between the number and the unit (like "10 m"), some won't (like "10m"), others will write out the whole unit (like "10 meters"), Germans will use a comma as decimal separator, the rest of the world will use a dot, and so on. It would definitely make sense to do some **normalization** in these cases, internally manipulating the visitors' search terms so that they conform to the general spelling / writing style on your website.

The extension features a search service (*Tx_TwLucenesearch_Service_Lucene*) that offers **two hooks** you can use for manipulating your visitor's search terms prior to running a single search (please visit http://typo3.org/documentation/article/how-to-use-existing-hooks-in-your-own-extension/ for general information on the nature and usage of hooks). The hooks are typically registered in

- either the */typo3conf/localconf.php*
- or any extension's *ext_localconf.php* file

- and are processed in the following order:

Search rewrite hook

The **search rewrite hook** basically operates on the whole raw search string, prior to any parsing by the search service. Changes you do inside this hook are just the same as if the seeker had entered them himself. This is the place for

- **normalizing spelling / writing differences** (like in the metric length example above),
- **converting typical word combinations to phrases** (by encapsulating them in quotes),
- excluding **stop words** from the search (a feature that is not natively implemented yet),
- and so on.

Just always keep in mind that using this rewrite hook will alter the search term before it gets parsed by the search service.

The search rewrite callbacks always get passed two arguments:

1. A list of parameters with (currently) one meaningful element, carrying the key "*searchterm*" and containing the raw search term string. The parameters array gets passed by reference, so you have to modify the value of the "*searchterm*" element in order to impact the current search.
2. A reference to the search service instance itself (you most likely don't need this for anything).

Inside the hook callback you can do whatever you want – just remember to write back your changes to the *$parameters['searchterm']* element in the end.

Example

The following example aims to be as simple and obvious as possible, so we'll simply put everything into the file */typo3conf/localconf.php*.

```
/**
 * Normalize different spellings of metric lengths
 *
 * The method detects and converts several metric length formats (like e.g. "10 m",
 * "10m", "10 meter", "10 metres" etc.) to one common format ("10m") and casts them
 * as phrase.
 *
 * @param array $params                                          Parameters
 * @param Tx_TwLucenesearch_Service_Lucene $service              Lucene Index Service
 * @return void
 */
function user_rewriteMetrics(array &$params, Tx_TwLucenesearch_Service_Lucene $service) {
      $searchterm         = ' '.trim($params['search']).' ';
      $pattern            = "%\s((?:\d+(?:[\.\,]\d+)?)|(?:[\.\,]\d+))\s*m(?:
(?:(?:eter)|(?:etre))s?)?\s%i";

      // Match and rewrite metric lengths
      while(preg_match($pattern, $searchterm, $metric)) {
            $length       = floatval(str_replace(',', '.', $metric[1]));
            $rewritten    = ' "'.$length.'m" ';
            $searchterm   = str_replace($metric[0], $rewritten, $searchterm);
      }

      $params['search']   = trim($searchterm);
}

$GLOBALS['TYPO3_CONF_VARS']['EXTCONF']['tw_lucenesearch']['search-rewrite-hooks'][] =
'user_rewriteMetrics';
```

Of course you could (and maybe even should) outsource the callback to an external PHP class or similar. Always make sure that your callback name (or class name respectively) starts with a valid class prefix (i.e. "tx_", "Tx_", "user_" or "User_"):

```
$GLOBALS['TYPO3_CONF_VARS']['EXTCONF']['tw_lucenesearch']['search-rewrite-hooks'][] =
'EXT:myext/Classes/Utility/Lucene.php:&Tx_Myext_Utility_Lucene->rewriteMetrics';
```

Term rewrite hook

The **term rewrite hook** acts shortly after the search rewrite hook and is applied on every single query token that has been extracted from the original search string. Again, the callback also gets passed two arguments:

1. A list of parameters with (currently) one meaningful element, having the key "*token*" and containing the current query token. A query token might be a single "word" or a "phrase" (quoted part of a search string) and always is an instance of the PHP class *Zend_Search_Lucene_Search_QueryToken*. The parameters array gets passed by reference, so you have to modify (or replace) the value of the "*token*" element in order to impact the current search.
2. A reference to the search service instance itself (you most likely don't need this for anything).

Compared to the search rewrite hook, which can modify the original search string and thus alter the number and structure of the resulting query tokens, the term rewrite hook operates on a single query token only. However, these tokens have already been parsed and typified by the search service. To learn more about the different token types and the query tokens' object properties you will have to study the *Zend_Search_Lucene_Search_QueryToken* source code (or the related articles available on the internet). The term rewrite hook enables you e.g. to modify or even completely replace a query token based on it's type and / or content.

Example

Again, the following example can be put into the file */typo3conf/localconf.php*. It demonstrates how a term rewrite hook can be used to convert non-phrase tokens representing floating point numbers into phrases (while substituting comma based decimal separators with dots at the same time).

```
/**
 * Rewrite single query tokens
 *
 * @param array $params                                          Parameters
 * @param Tx_TwLucenesearch_Service_Lucene $service              Lucene Index Service
 * @return void
 */
function user_rewriteFloats(array $params, Tx_TwLucenesearch_Service_Lucene $service) {
      /* @var $token Zend_Search_Lucene_Search_QueryToken */
      $token            =& $params['token'];
      $term             = $token->text;

      // Convert floating numbers to phrase tokens (and replace commas with dots)
      if (($token->type != Zend_Search_Lucene_Search_QueryToken::TT_PHRASE) &&
            preg_match("%^\d*[\,\.]\d+$%", $term)) {
            $token      = new Zend_Search_Lucene_Search_QueryToken(
                  Zend_Search_Lucene_Search_QueryToken::TT_PHRASE,
                  str_replace(',', '.', $term),
                  $token->position
            );
      }
}

$GLOBALS['TYPO3_CONF_VARS']['EXTCONF']['tw_lucenesearch']['term-rewrite-hooks'][] =
'user_rewriteFloats';
```

Using boost factors

The Lucene Index allows influencing the relevance / order of search results by "boosting" search matches on three levels:

1. **Document level boosting**: Applied during indexing, enhancing whole index documents (currently not supported by this extension).
2. **Field level boosting**: Applied during indexing, enhancing single fields within documents (currently not supported by this extension).
3. **Query level boosting**: Applied during querying, enhancing document relevance if a single term is matched. This is the only boosting level currently supported by this extension.

By default, index documents have no boost – or, rather, they all have the same boost factor of 1.0. Although the boost factor always has to be positive, it can be less then 1 (e.g. 0.2). By changing a document's boost factor, you can instruct the Lucene Index to consider it more or less important with respect to other documents in the index. The same concept applies on the field level: By changing a single field's boost factor, a term match within this field will be considered more or less important compared to matches within fields with different boost factors.

With the TypoScript setting *search_lucene.searchConfig* you can control the query level boosting and thus the importance of query term matches within the searchable document fields. For more information on query level boosting – or the Lucene query syntax in general – please visit http://lucene.apache.org/core/3_6_1/queryparsersyntax.html#N100DA.

Developer tools

Currently there are two developer tools you can use for working on and optimizing your index configuration. Please make sure to enable them in your extension configuration via the Extension Manager (and disable it again for production use).

Display index relevant contents only

Sometimes it is quite handy to see exactly what the indexer sees. For this purpose, simply append the GET parameter *index_content_only* with the value 1 to the frontend URL you want to examine:

Example

```
http://example.com/index.php?id=1&index_content_only=1
```

The indexer will then abort the regular page rendering and output the indexable contents of the page instead (as plain text). This way you can e.g. check if you used the indexing control markers *<!--TYPO3SEARCH_begin-->* and *<!--TYPO3SEARCH_end-->* correctly in your templates.

Force the re-indexing of a document

In case you want a page to be forcibly re-indexed (e.g. because you altered the *search_lucene.searchConfig* settings after the page has been indexed), you can append the GET parameter *index_force_reindex* with the value 1 to the URL.

Example

```
http://example.com/index.php?id=1&index_force_reindex=1
```

The page will be re-indexed, regardless of it's last modification timestamp.

Known problems

The Lucene Index is file based. Depending on the size and structure of your index it might grow until it exists of quite a lot of single files that all have to be kept open while indexing or searching. The number of files that can be opened at the same time is limited by your operating system. Under certain circumstances this limit might get exceeded by your index, resulting in a "**Too many open files**" error. This is not a limitation of the extension or the Lucene Index but rather of the operating system itself. You can probably do something about it by altering the **MergeFactor** value in your extension configuration. To learn more about the meaning of the MergeFactor please visit http://framework.zend.com/manual/en/zend.search.lucene.index-creation.html#zend.search.lucene.index-creation.optimization.mergefactor.

To-Do list

Future improvements could include:

- Indexing of external documents (pdf, doc, txt ...)
- Indexing and searching of something else than frontend pages
- A crawler that gets everything indexed on a regular basis (or maybe an alternative and better approach?)
- An “advanced search” assistant / form
- A dedicated backend module for managing the index (instead of using the update script), e.g. with the ability of deleting / updating single index documents
- Support for stop words

In general, this extension is still very basic and leverages only a small subset of the features supported by the Lucene Index. So far, the functionality has been sufficient for the author, but if there's enough public demand the extension will be further improved. So please let us know what you think and use the GitHub repository for reporting of issues. Thanks!

ChangeLog

Version:	Changes:
2.0.2	Bugfix release: Fixed problem with Umlauts in search terms
2.0.1	Bugfix release: Fixed incompatible view helper declaration
2.0.0	TYPO3 8 release: Uses new Icon API (TYPO3 7.5+)
1.6.0	TYPO3 7 release: Multiple bugfixes, suitable for composer mode TYPO3
1.0.2	Fixed a regression bug with wrong Flexform file name
1.0.1	Fixed a bug causing whitespace errors when indexing HTML5 documents
1.0.0	TYPO3 CMS 6 Release: Switch to PHP namespaces and the new class / sysext structure (no usage of the compatibility layer!) ATTENTION: Support for TYPO3 4.x has been dropped, use the 0x versions instead! Added more German language labels
0.6.1	Fixed significant manual error: The indexing start marker is ***<!--TYPO3SEARCH_begin-->*** (instead of *<!--TYPO3SEARCH_start-->* as erroneously documented before)! Minor manual corrections
0.6.0	Initial public release to the TYPO3 Extension Repository