EXT: A Better Tag Cloud

Extension Key: vge_tagcloud

Language: en

Keywords: forEditors, forBeginners

Copyright 2009-2013, Francois Suter, <typo3@cobweb.ch>

This document is published under the Open Content License

available from http://www.opencontent.org/opl.shtml

The content of this document is related to TYPO3

- a GNU/GPL CMS/Framework available from www.typo3.org

Table of Contents

Introduction

What does it do?

This extension can take content from (nearly) any table in the TYPO3 database, split it into words and create a tag cloud by counting the occurrences of each word. It offers a lot of flexibility for sorting, filtering and styling the results. It is also capable of handling multiple languages and workspaces.

The extension also provides a second plug-in to display a list of pages based on a list of page numbers associated with each keyword. The content of both plug-ins is cached properly.

Screenshots

Here is a screenshot of the result. A tag cloud...

Credits

The development of this extension was funded by the Geneva City Council. It was part of a workshop on TYPO3 extension development. Thus you will find that the source code is very clean and extensively commented. As such this extension may be useful as an example for other extension developers.

Note that since the initial development the TYPO3 Coding Guidelines have evolved and though this plugin is still pretty clean, it doesn't represent state of the art code anymore.

Questions?

If you have any questions about this extension, please ask them in the TYPO3 English mailing list (typo3.english), so that others can benefit of the answers.

Keeping the developer happy

It's unfortunately not possible to rate extensions anymore (maybe that will change again), so feel free to send thanks to the developers or praise the extension in general. If you really want to give something back, you may consider my Amazon wish list: http://www.amazon.co.uk/registry/wishlist/G7DI2AN99Y4F

You may also take a step back and reflect about the beauty of sharing. Think about how much you are benefiting and how much yourself is giving back to the community.

Installation

Installing the extension

The installation is a pretty straightforward business:

1. download and install the extension using the Extension Manager

2. include the extension's static TS template into your template
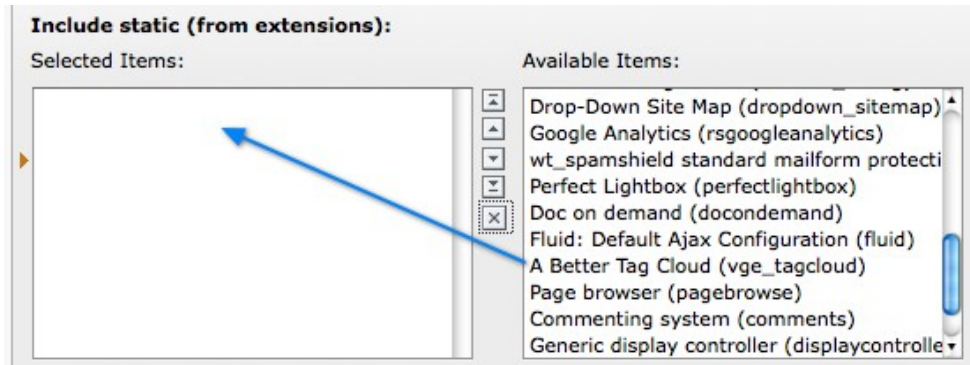
3. insert the tag cloud as an element on your page or via TypoScript to have it appear on a bunch of pages.

Upgrading

There are some minor issues to pay attention to when upgrading from older versions.

Upgrading to version 1.7.0

A tiny security improvement in version 1.7.0 introduced a change of TS configuration. The PHP function `htmlspecialchars()` is now called when displaying the search keyword in the pi2 plugin. Before this was achieved with the TS property "htmlSpecialChars" being activated on the `plugin.tx_vgetagcloud_pi2.keyword` property.

If you have defined your own setup for the `plugin.tx_vgetagcloud_pi2.keyword` property, you should make sure to disable "htmlSpecialChars", otherwise the keyword will be escaped twice.

Upgrading from below version 1.4.0

As of version 1.4.0 a separate stdWrap property was introduced in the TypoScript setup when rendering the tag cloud using styles. It is called "tagWrapStyles" and it is used instead of "tagWrap" if the rendering type is set to "style".

This may lead to some surprises if you are using the styles rendering and had changed the "tagWrap" property. You need to switch to using the "tagWrapStyles" property instead.

Furthermore the default TS for "tagWrap" was quite heavily changed to account for better sanitizing of the output. You should check if your TS changes still match the new setup.

Users Manual

General considerations

Before diving into the plug-in's options, it is important to understand how the data for the tag cloud is gathered and compiled, and what is done afterwards with it for rendering the actual tag cloud.

Data is fetched from one or more fields from a table inside the TYPO3 database. Only tables that are referenced in the TCA are available for selection. Similarly only columns (fields) form those tables that have a TCA description can be selected. The data gathered can be limited by selecting a starting point in the TYPO3 page tree.

Depending on which field is chosen, the data may either be single keywords or complete texts from which keywords must be extracted. So the plug-in's next work is to split the data into individual words. This list is then parsed and every occurrence of the same word is added to a running total. The result is a list of unique words with a number of occurrences for each. This is referred to as "weight" in the rest of this manual.

This weight is of course of utmost importance since the whole point of tag clouds is to highlight words according to their importance. The weight is thus used in the styling process when rendering the tag cloud. Two styling processes are available. The first one uses relative weight boundaries (defaults are 100% and 200%) which are directly used as font size attributes. According to its weight, a word receives a relative size somewhere between the minimum and maximum weight boundaries.

The other display possibility is using a list of predefined styles. For example if you define four styles, the words will be split into four weight groups and attributed the corresponding style.

How those weights and styles are actually used all depends on the TypoScript configuration.

Inside the tag cloud, each keyword is actually linked to a page. The same page is used for all keywords, but of course query parameters may vary. The default setup of the first plug-in (pi1) makes it ready to link to a page containing the second plug-in (pi2).

Usage as a content element

A Better Tag Cloud can be inserted as a traditional content element on a page, just like any other plug-in. Just choose the right type either from the plug-ins list in the content element form or from the new content wizard.
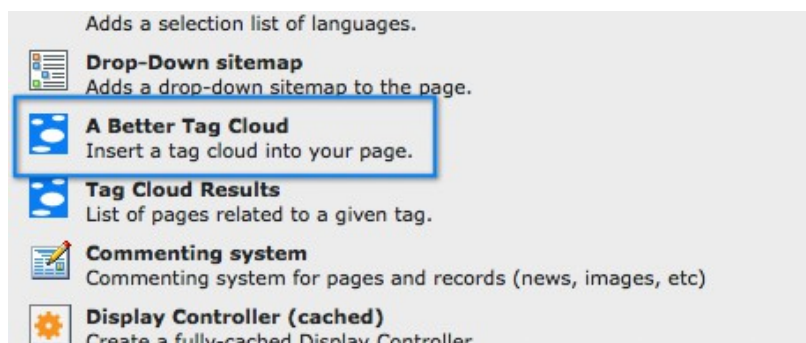
Most of the parameters can then be set using the provided flexform. A few special parameters can only be set using TypoScript (see Configuration). The flexform is split into three tabs. The properties that can be set in each tab are described below.

The only exceptions are the starting page and the level of recursing, which are set in the "Behaviour" tab of the content element.

The tag cloud will be assembled based on the selected pages and any children page encompassed in the "recursive" setting. If no starting point is defined, the default TypoScript setup will make it start from the root page of the web site. In case no TypoScript is defined at all, the tag cloud will take the current page as a starting point.

General Setup

- **Reference table:** choose the table from which you want to draw keywords.

- **Reference fields:** choose one or more fields from the table selected above.

- **Include not in menu pages:** choose whether to includes pages that are not in menu (when querying the pages table) or elements in pages that are not in menu (when querying the tt_content table, for example, on any other record that has a pid field).

- **Character for splitting field(s) value:** define which character to use to split the values found in the selected fields into individual keywords. For example, if you choose the "keywords" field of the "pages" table, you have to define the comma as a separator. By default, no separator is defined, which means the field(s) will be split along word boundaries (however see "A note about word boundaries" in the Configuration section).

- **Unique keywords for each item:** if this option is checked, a given keyword is counted only once in each item considered by the tag cloud. Example: you are building the tag cloud based on news title and have two news entitled "It's so good I could cry" and "Cry baby cry". By default "cry" would have a count of 3 (1 in the first title and 2 in the second). By checking this box, "cry" counts only once in the second title and its total count is now 2.

- **Case handling:** define whether keywords should be left as is or converted to lower or upper case. Note that if keywords are left as is, they will be differentiated based on case. For example, "typo3" and "TYPO3" will appear as two different tags in the cloud. The default behavior is to convert all words to lower case.

- **Minimum word length:** a minimum length for words to be included in the tag cloud. Any word shorter that the given number will be left out.

- **Target page for tag links:** choose the page the links on each tag should point to. This may be, for example, the page with the second plug-in (pi2) or the search results. Note that choosing a page is only part of the task. The actual links are built using TypoScript (see Configuration). The target page can also be defined globally by editing the plug-in's constants.

- **Sorting (before capping):** you may limit the number of words that appear in the tag cloud (see below). However before cutting off the list of keywords you may want to sort them using this property. Possible values are natural (i.e. nothing), alphabetical or by weight (i.e. the number of occurrences of each keyword). The default is by weight.

- **Sort order:** this property complements the previous one by allowing to choose an order for sorting. The default is descending (so highest goes first).

- **Maximum number of tags:** set the maximum number of keywords that should appear in the tag cloud. An empty field (or a value of zero) means that all keywords get displayed.

Format & Style

- **Type of styling:** choose to use styles or weights (if this is not clear, please refer to the General considerations above).

- **Minimum weight (if using weights)/Maximum weight (if using weights):** define limits for the weights to be

used. These fields support only numbers. Do not enter units here. Units belong to the TypoScript configuration. However those units condition what kind of value you need to enter here. If using percentages, you will want to enter values like "100" and "200". If you use "em", you will want to enter something like "1" and "2".

- **Scaling (if using weights):** choose a scaling method other than "linear" to smoothen or – on the contrary – enhance the differences between the weights of the various tags. This is useful when there are wide differences in weight or when a given word has an extraordinary weight compared to others. The picture below gives an idea of

Default (Linear)

High weights flattened

Low weights flattened

High and low weights flattened

Middle weights flattened

the effect of each setting.

- **List of styles, comma-separated (if using styles):** list the name of the styles you have prepared in your CSS.

- **Separator:** define some characters that you may want to insert between each keyword.

- **Sorting (for display):** define the sorting of the keywords for the rendering. The default is alphabetical.

- **Sort order:** this property complements the previous one by allowing to choose an order for sorting. The default is ascending (so lowest goes first).

TypoScript
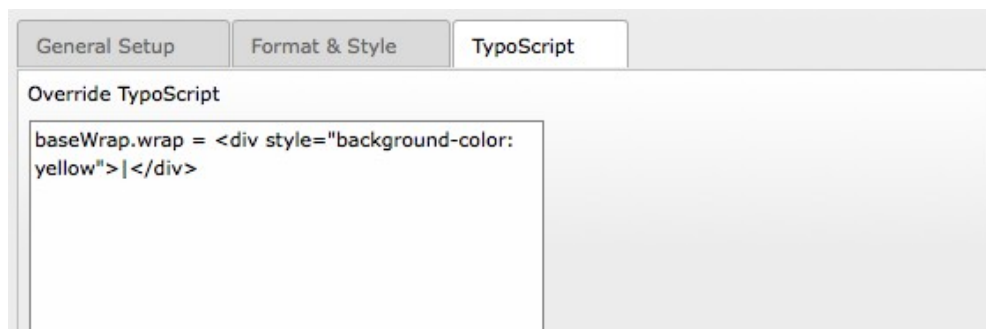
The last tab contains a single text field in which some TypoScript can be entered. This can be any TypoScript configuration that is described in the "Configuration" section below. It is entered without the `plugin.tx_vgetagcloud_pi1` prefix (see example in screenshot above).

This TypoScript will override any values coming from the main TS template and also values defined in the other tabs of the FlexForm.

There are limits to what can be done inside this field. Actually the TS entered here is only parsed and merged with the main TS. This means that operator like >, <, =< and := will **not** work. This means that it is not possible to unset properties. They must actually be overridden by some other value. Empty values will override non-empty values.

Also note that conditions and constants **cannot** be used inside this field, although include files can.

Usage in a template

It is possible to include the tag cloud in a template to have it displayed on a bunch of or even all pages of your web site. Simply include it as appropriate for the templating system that you use.

Here is an example for the traditional templates:

```
temp.cloud < plugin.tx_vgetagcloud_pi1
temp.cloud {
     startPage.data = leveluid: -2
     recursive = 2
}
page.subparts.TAGCLOUD < temp.cloud
```

The same for TemplaVoilà:

```
lib.cloud < plugin.tx_vgetagcloud_pi1
lib.cloud {
     startPage.data = leveluid: -2
     recursive = 2
}
```

then define a TemplaVoilà element as a TypoScript Object Path and set it to point at lib.cloud.

Setting up a results page

Although it is possible to imagine the tag cloud existing all by itself, the general use of such a tool is to be able to click on a given keyword and get a sensible result. This means that the links on each keyword must point to a page which knows what to do with the information that is passed to it. There are many ways to achieve that:

- pass the keyword in a "tx_indexedsearch[sword]" variable and point to a page with the indexed search engine. This will trigger a search using the keyword;

- something similar can be done with news, by passing the keyword inside a "tx_ttnews[swords]" variable and pointing to a page with the tt_news search engine;

- pass the list of pages on which the keyword was found to a page containing the second plug-in. This will display those pages as a list.

- using the TypoScript setup or some of the hooks it is possible to link to many other extensions. See the chapter on hooks for an example of how to link to a glossary extension.

The various TypoScript setups for such results pages are detailed in the "Configuration" chapter.

To set up a page with the indexed search or the tt_news search engines, please refer to the manual of those extensions. To use this extension's second plug-in (pi2), just create a new content element and select the appropriate plug-in from the wizard:
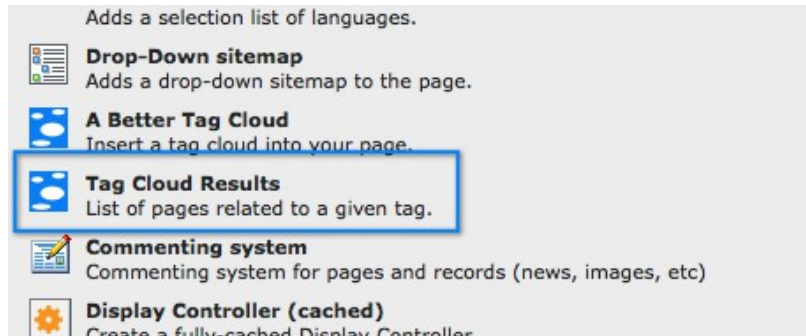
Done. There's nothing more to do here. Configuration is done using TypoScript.

Configuration

TypoScript Constants

Property:	Data type:	Description:	Default:
pageID	int	uid of the page used for the links on each tag	0

TypoScript Setup

plug-in 1

Property:	Data type:	Description:	Default:
startPage	int/stdWrap	Comma-separated list of page uid's or a stdWrap The default value simply points to the root page of the site.	startPage.data = leveluid:0
recursive	int	Number of levels to explore below each page defined in the startPage property. For infinite recursing just set a very large value (e.g. 255)	255
referenceTable	string	Name of the table to fetch the data from	pages
referenceFields	string	Comma-separated list of fields to fetch the data from	keywords
addWhere	string	Additional SQL WHERE clause that will be applied to the query of the reference table. It must **not** start with the "WHERE" nor the "AND" keywords.	
references	indexed array of reference configurations -> referenceConfig	This property makes it possible to query several tables instead of just one. For each reference you can indicate a table, a list of fields and an additional SQL where clause. See example of use in the "TypoScript examples" section below.	
exclude	special	This property makes it possible to define some criteria of exclusion for the data being fetched. This is defined for the reference table and for any field of that table. Values that match are excluded. Several values can be set, separated with commas. If you look at the default value, it means that we put conditions on both the tt_content and the pages tables. Obviously these conditions will be used only if we are indeed querying either tt_content or pages. In the case of tt_content the criteria state that we must exclude content elements of Ctype "mailform" and any instance of the tag cloud plug-in itself. As for pages we exclude several types (spacer, sysFolder and recycler) and those not in menu.	exclude.tt_content.CType = mailform exclude.tt_content.list_type = vge_tagcloud_pi1 exclude.pages.doktype = 199,254,255
includeNotInMenu	boolean	Includes pages that are not in menu (when querying the pages table) or elements in pages that are not in menu (when querying the tt_content table, for example, on any other record that has a pid field). Default is to not include.	0
splitChar	string/stdWrap	Character to use to split the values found in the reference fields. If left empty, values are split along word boundaries.	
uniqueKeywordsPerItem	boolean	If checked, each keyword will be counted only once inside each item considered by the tag cloud (see more details in the description of the "General Setup" tab above)	0
splitWords	string	Regular expression for splitting the words along their boundaries Please refer to "A note about word boundaries" below	[-.,;:"&%()<>!?/'\s]

11

Property:	Data type:	Description:	Default:
splitWordsWrap	wrap	This wrap is applied on the splitWords property described above to complete the regular expression syntax. Change it in case you want to use other wrappers than "/" around the regular expression or other modifiers. Its default value (with modifier "u") makes it able to handle UTF-8 content properly. **NOTE**: it was kept separate from splitWords for backwards-compatibility reasons.	/|/u
caseHandling	string	How to handle the case of keywords. Leave empty to leave keywords as is. Set to **lower** to force to lower case, or **upper** to force to uppercase.	lower
targetPage	int	uid of the page the links around each keyword should point to	{$plugin.tx_vgetagcloud_pi1.pageID}
minLength	int	A minimum length for words to be included in the tag cloud. Any word shorter that the given number will be left out.	
sorting	string	Criterion by which to sort the keywords before applying a limit to the number of keywords. Possible values are **natural** to avoid sorting the keywords, **alpha** (for alphabetical sorting) or **weight**.	weight
sortOrder	string	Complements the above parameter with an order, either **asc** (ascending) or **desc** (descending)	desc
maxWords	int	Maximum number of words to include in the tag cloud. Leave empty to include all words.	
renderingType	string	Type of rendering to use. Select **weight** for relative weights or **style** for using a list of styles.	weight
minWeight	int	Minimum weight to use for the relative weights type of rendering	100
maxWeight	int	Maximum weight to use for the relative weights type of rendering	200
scale	string	Scaling to apply to the weight of each word. This makes it possible to flatten out or – on the contrary – enhance the differences between weights at each end of the scale. The possible values are: • **linear**: this is the default value. It does nothing, all weight are used as is. • **flatTop**: this reduces the highest weights. This is useful when some words are used very frequently and this dwarf all other words inside the tag cloud. • **flatBottom**: this reduces the differences for the lowest weights. • **flatTopAndBottom**: combines the effects of both of the above. Both high and low values will be evened out. • **flatMiddle**: this evens out the differences in the middle range of weight. See the picture in the Users Manual.	linear
scaleFactor	int+	Related to the "scale" property above. This value modified the behavior of the "flatTop" and "flatBottom" scales. A higher value will enhance the effect, a lower value will reduce it.	3
styles	string	Comma-separated list of styles to use when rendering with styles	
separator	string	Characters to insert between each keyword	
sortingForDisplay	string	Criterion by which to sort the keywords for display. Possible values are **natural** to avoid sorting the keywords, **alpha** (for alphabetical sorting) or **weight**.	alpha
sortOrderForDisplay	string	Complements the above parameter with an order, either **asc** (ascending) or **desc** (descending)	asc

Property:	Data type:	Description:	Default:
tagWrap	stdWrap	stdWrap property for each keyword in the tag cloud when rendering using the **weights**. The link placed on the keyword is defined in the stdWrap itself.	See "TypoScript examples" below
tagWrapStyles	stdWrap	Same as tagWrap but used when rendering with **styles**.	See "TypoScript examples" below
cloudWrap	stdWrap	stdWrap for the complete tag cloud Note that the default value takes care of excluding the tag cloud's content from the indexed search engine.	See "TypoScript examples" below
baseWrap	stdWrap	stdWrap for the whole of the plug-in's content. Use that if you don't want the content to be wrapped using pi_wrapInBaseClass().	
_CSS_DEFAULT_ STYLE	string	The plug-in's configuration includes some basic styles that make the tag cloud display correctly. Use as a basis for your own styling.	

[tsref:plugin.tx_vgetagcloud_pi1]

-> referenceConfigs

Property:	Data type:	Description:	Default:
table	string	Name of the table	
fields	string	Comma-separated list of fields	
where	string	Additional conditions to add to the SQL WHERE clause (do **not** start with "AND")	

[tsref:plugin.tx_vgetagcloud_pi1.referenceConfigs]

plug-in 2

Property:	Data type:	Description:	Default:
keyword	stdWrap	stdWrap for the keyword that was passed to the page in the "keyword" variable	keyword.wrap = |
message	stdWrap	stdWrap for the result message displayed by the plug-in	message.wrap = |

results	cObj	Content object for the rendering of the list of pages. The default is a simple hierarchical menu.	results = HMENU
			results {
			    special = list
			    special.value.field = tag_pages
			    1 = TMENU
			    1 {
			        NO.allWrap = |
			    }
			    wrap = |
			}

baseWrap	stdWrap	stdWrap for the whole of the plug-in's content. Use that if you don't want the content to be wrapped using pi_wrapInBaseClass().	

[tsref:plugin.tx_vgetagcloud_pi2]

TypoScript examples

Several values are available for use inside the tagWrap property. They are dynamically defined for each keyword. They are:

- tag_keyword: the keyword itself;

- tag_link: the id of the page to link to. Note: this value is the same for all keywords;

- tag_weight: the absolute weight of the keyword (i.e. the number of occurrences);

- tag_style: the relative weight or style from the list that apply to this keyword;

- tag_id: an incremented unique number that can be used if you want to place a "id" attribute on each keyword;

- tag_pages: a list of page numbers the keyword was found in, separated by underscores;

- tag_startpage: the id of the page chosen as a start page for collecting keywords (corresponds to the "Startingpoint" field or to the "startPage" TypoScript property). Note: this value is the same for all keywords.

Below are examples of use.

Default setup

The default value of the "tagWrap" property makes use of most of the special values introduced above:

```
plugin.tx_vgetagcloud_pi1 {
    tagWrap {
        htmlSpecialChars = 1
        typolink {
            parameter.data = field:tag_link
            additionalParams.cObject = TEXT
            additionalParams.cObject {
                field = tag_keyword
                rawUrlEncode = 1
                dataWrap = &tx_vgetagcloud_pi2[pages]={field:tag_pages}&tx_vgetagcloud_pi2[keyword]=|
            }
            ATagParams.cObject = TEXT
            ATagParams.cObject {
                field = tag_keyword
                htmlSpecialChars = 1
                dataWrap = id="tag{field:tag_id}" title="|"
            }
            useCacheHash = 1
        }
        dataWrap = <li style="font-size: {field:tag_style}%">|</li>
    }
}
```

Let's look at the above code in detail. The "tagWrap" property is a stdWrap. The value is the current keyword. This will be put through `htmlspecialchars()` so that any special characters are properly rendered.

Then the typolink is built around the keyword. The link's destination if defined using the `tag_link` special value. Additional parameters are defined so that the second plug-in (pi2) can receive all the information it needs. This means the list of pages on which the keyword was found (stored in the special value `tag_pages`) and the keyword itself (stored in the special value `tag_keyword`). The latter is properly encoded for usage in a URL.

Next some parameters are added to the hyperlink itself. The keyword (again using `tag_keyword`) is used for the title attribute and a unique id (stored in the special value `tag_id`) is added to the link, making it possible to target every individual link, if needed. Again the title is put through `htmlspecialchars()`.

Note the activation of `useCacheHash` in the typolink setup. This is necessary because the link refers to the pi2 plugin which is cached (its type is USER). If the `useCacheHash` is not enabled you risk running into the empty cHash problem (if you don't understand what this is all about, please refer to the article, "The Mysteries of &cHash", http://typo3.org/development/articles/the-mysteries-of-chash/).

Back from the typolink, the result is wrapped inside a list item, on which the relative weight is applied as a percentage of font size, using the `tag_style` special value.

The wrap around the cloud completes the setup:

```
plugin.tx_vgetagcloud_pi1.cloudWrap.wrap = <!--TYPO3SEARCH_end--><ul>|</ul><div
style="clear:both;"></div><!--TYPO3SEARCH_begin-->
```

All tags are wrapped inside an unordered list and search engine markers are set so that the tag cloud is not indexed. The default styles provided by the extension ensure that the tags are displayed as floating elements and not as actual list items.

Linking to the indexed search engine

Using the individual keywords from the tag cloud to launch a search is very easy. Little configuration needs to be changed from the default setup.

```
plugin.tx_vgetagcloud_pi1 {
    tagWrap {
        typolink {
            additionalParams >
            additionalParams.field = tag_keyword
            additionalParams.rawUrlEncode = 1
            additionalParams.wrap = &tx_indexedsearch[sword]=|
        }
    }
```

```
}
```

Of course the destination of the typolinks must be a page containing the indexed search plug-in.

Linking to tt_news and TIMTAB

This example shows how to build a tag cloud based on tt_news. This means this method can also be used for basing a tag cloud on TIMTAB entries. Very little needs to be changed from the default setup.

```
plugin.tx_vgetagcloud_pi1 {
     referenceTable = tt_news
     referenceFields = keywords
     tagWrap {
          typolink {
               additionalParams >
               additionalParams.field = tag_keyword
               additionalParams.rawUrlEncode = 1
               additionalParams.wrap = &tx_ttnews[swords]=|
          }
     }
}
```

With this each keyword will start a tt_news search. Of course, the links on the keywords must point to a page containing the tt_news plug-in with code SEARCH.

Querying multiple tables

The tag cloud can be built using data from multiple tables. This can only be achieved using TypoScript and not with the FlexForm configuration. Here is an example of such setup:

```
plugin.tx_vgetagcloud_pi1 {
     references {
          1 {
               table = pages
               fields = title,keywords
               where = doktype = 1
          }
          2 {
               table = tt_content
               fields = header
          }
          3 {
               table = tt_news
               fields = title
          }
     }
}
```

In this example code we decide to query three tables: the pages, the content elements and the news items. In the pages table, we will use the title and keywords fields, the header field of content elements and the title field of the news items. Furthermore the pages are restricted to being only of `doktype = 1`. This will be added to the WHERE clause of the SQL query that will run on the pages table.

Precedence of FlexForm and TypoScript

FlexForm settings always override TypoScript set up. The only exception to that is when querying multiple tables. In this case, the TypoScript property "references" is used no matter what values are entered into the FlexForm "Reference table" and "Reference fields" selectors.

The TypoScript entered in the "Override TypoScript" of the FlexForm takes precedence over the TS from the main template as well as the FlexForm settings.

Consistency between pi1 and pi2

Depending on your setup you must take care to ensure consistency between the pages found by pi1 and the pages displayed by pi2. In the latter the default rendering for the list of pages uses a HMENU content object. This object takes care of all the hassle of enable fields (hidden, deleted, access restrictions, publication dates), language overlay and versioning. The pi1 code does the same. But the HMENU object is also going to respect the "Not in menu" field. This is why the default pi1 setup excludes those pages (`includeNotInMenu = 0`). If you change this setup, you must also change the HMENU in pi2 to display the pages that are not in menu (`results.includeNotInMenu = 1`).

The "includeNotInMenu" property in the pi1 plug-in is also important, for example, when retrieving your keywords from content elements. With the default settings, the pi1 plug-in will not reference content elements on pages that are not in menu. That way when the pi2 is called, the results are consistent. If you want to take words also from not in menu pages,

you have to set "includeNotInMenu" to 1 in both plug-ins setup.

A note about word boundaries

In theory it is easy to split a block of text into single words. One needs only call the PHP function preg_split() and use the regular expression "\b". In practice it is not quite that simple, at least when using UTF-8 encoding. In this case \b will split words on characters with diacritical marks (like "é" or "ü") thus ruining the whole thing. In such a case it is necessary to be more specific, hence the TypoScript property "splitWords" to define which regular expression to use in preg_split() to find word boundaries.

The default value should cover most cases, but it is possible that you may have to modify it. Then again if you're just using English and no UTF-8 encoding, you may try to revert to simple "\b".

Note that the TypoScript property is included inside a single-quoted string and that single quotes and forward slashes inside the regular expression will be escaped for proper syntax. If you want to split words along backslashes too, you need to escape the backslash yourself (i.e. type "\\").

RealURL configuration

The extension includes a file for RealURL autoconfiguration. If you do not use this mechanism, it is still quite easy to set things up for the pi2 plug-in to work with RealURL. Here is an example configuration, but you may of course change it if you know what you're doing:

```
$TYPO3_CONF_VARS['EXTCONF']['realurl'] = array(
    '_DEFAULT' => array(
        'postVarSets' => array(
            '_DEFAULT' => array(
                'tagcloud' => array(
                  array('GETvar' => 'tx_vgetagcloud_pi2[keyword]'),
                  array('GETvar' => 'tx_vgetagcloud_pi2[pages]')
                )
            )
        )
    )
)
```

This will produce URLs like http://www.mydomain.com/tag-cloud-results/tagcloud/somekeyword/1_2_3 when calling the page where an instance of the pi2 plug-in resides.

Extending the Tag Cloud

As is common in TYPO3, most classes that make up this extension can be modified using an XCLASS. On top of this there a number of hooks that allow various manipulations.

Hooks

All hooks are listed below with their name and purpose. They all belong to the first plug-in (pi1).

- **postProcessPages**: this hook can be used to manipulate the list of pages from which keywords will be collected. It receives the list of pages as an array and is expected to return an array too.

- **extractKeywords**: this hook is used to register methods for extracting keywords from the raw data coming from the database. These methods are used **instead** of the default methods. They are expected to return simple arrays filled with keywords.

- **postProcessRawKeywords**: this hook allows the manipulation of the the full list of keywords before any sorting or capping is applied to it, but after case transformation. Methods called by this hook receive the list of all keywords and are expected to return a simple array with the full list of keywords.

- **postProcessFinalKeywords**: this hook is called just before the keywords are displayed and allows for a list-minute change. It receives the sorted and capped list of keywords, along with their count. It is expected to return an array with the same structure.

- **processTagData**: this hook comes in at an even later point and makes it possible to manipulate the cObj data just before the rendering is done. It receives a reference to the cObj data and can manipulate it at will. It returns nothing.

An example class for most of the hooks is provided in the file `hooks/class.tx_vgetagcloud_hook_example.php`. For the the "extractKeywords" hook, the example method shows how to extract data from a flexform field, if such was chosen in the plugin's setup. For the "postProcessRawKeywords" hook, the example is a function that removes all keywords that are 2-letter long or less. One could also imagine, for example, removing all keywords that appear only once. The example for the "postProcessFinalKeywords" hook introduces a special sorting where keywords are arranged by length (rather useless, but looks nice).

The "processTagData" hook example shows how to link the tag cloud to extension "sg_glossary". In the constructor of the `tx_vgetagcloud_hook_example` class a list of keywords is loaded from the glossary. Then in the hook the keyword (as stored in `tag_keyword`) is matched against this list to retrieve the corresponding uid from the `tx_sgglossary_entries` table. This value is stored in a new field `tag_uid`. To create the link to the glossary a simple modification is done to the TS setup:

```
plugin.tx_vgetagcloud_pi1
     tagWrap {
          typolink {
               additionalParams >
               additionalParams.field = tag_uid
               additionalParams.wrap = &uid=|
          }
     }
}
```

Examples of how to register those hooks are found in the `ext_localconf.php` file (the lines are commented out so that the example functions are not activated by default).

To Do List

Here are some ideas of features that could be implemented:

- Make it possible to use a database table where keyword weights are already stored.

- Try setting an optionSplit property on the tagWrap rather than using the separator property

- In the flexform, put a message along the lines of "Please select a table first" in the referenceFields as long as a reference table has not yet been selected

- Transfer some of the ideas for the use of hooks to the plug-in configuration (minimum length, minimum weight)

- It might be useful to be able to use a stop word service in the tag cloud, but such a service has to be developed first...

The development roadmap can be found on Forge: http://forge.typo3.org/projects/extension-vge_tagcloud/roadmap

Please post feature requests to the issue tracker: http://forge.typo3.org/projects/extension-vge_tagcloud/issues

Known problems

None to date, except word boundaries which may not be quite correct.

If you have any issues, please post your problems to the TYPO3 English mailing list (typo3.english), so that others may benefit of the answers too. For bugs or feature request, please open an entry in the extension's bug tracker on Forge (http://forge.typo3.org/projects/extension-vge_tagcloud/issues).