EXT: Templatedisplay

Extension Key: templatedisplay
Language: en
Keywords: forAdmins, forIntermediates

Table of Contents

Introduction

Templatedisplay is a kind of "mappable" template engine for TYPO3. The extension is part of the Tesseract extensions family and deals with rendering content onto the Frontend.

In short words, it enables to do a mapping between markers and databases fields. A marker is a pattern that will be replaced dynamically by a value coming from the database. This value can be formatted according to some TypoScript configuration. It is also possible to incorporate user defined markers within a hook.

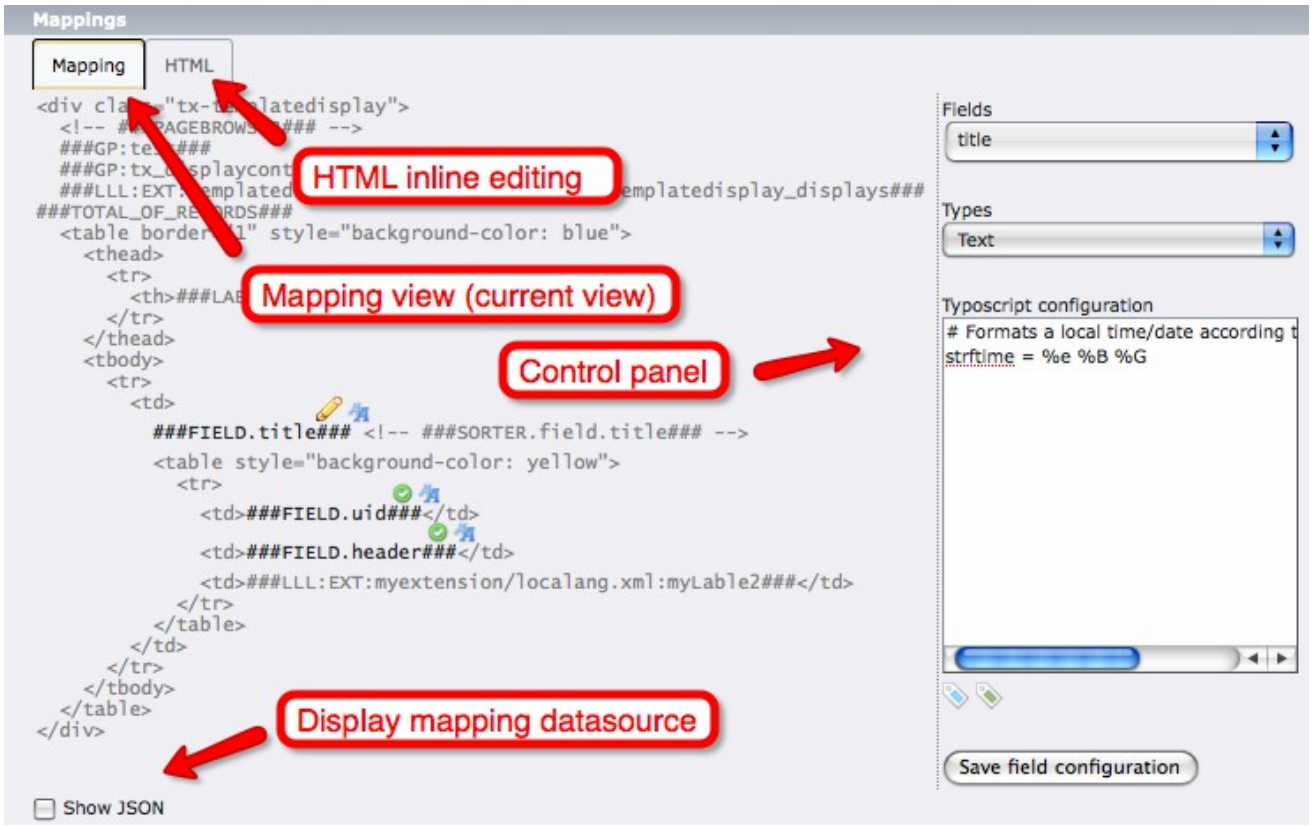
Templatedisplay is well designed for rendering lists with advanced features like sorting, filtering, page browsing. It offers a simple syntax for looping on record set, testing condition, counting records.

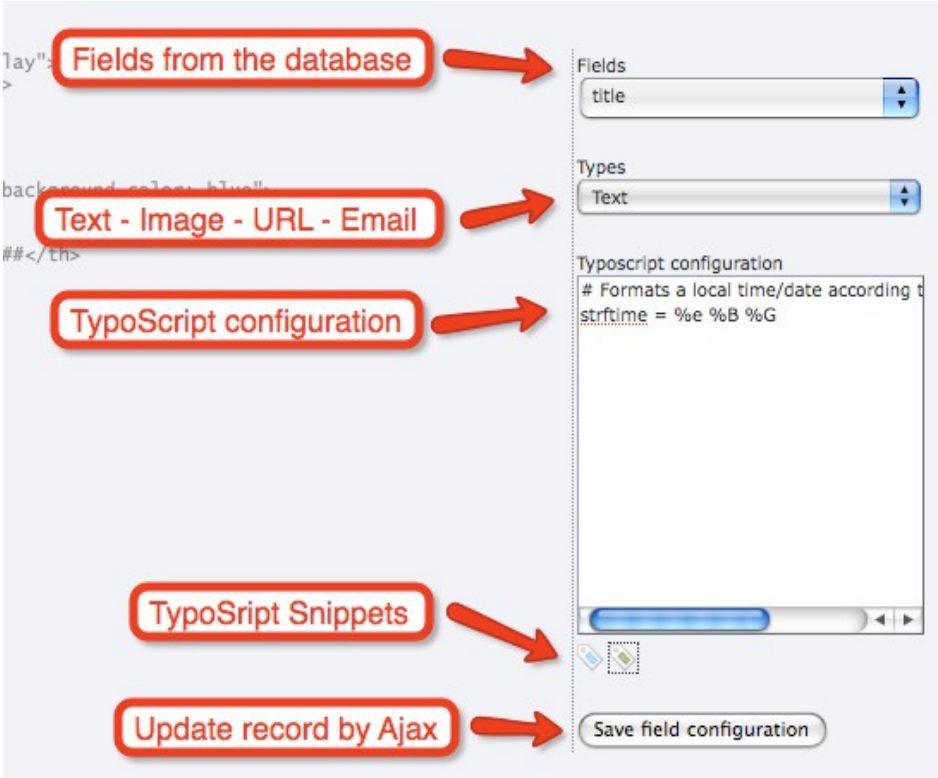
Screenshots

This is basically the default view where it is possible to map any markers to a database's field. The mapping is done by

1. clicking on a marker (e.g. ###FIELD.title###),
2. selecting a field in the drop-down menu,
3. selecting a type within the list,
4. adding some possible additional configuration,
5. clicking the "save field configuration" button.

By using the name of the marker, Templatedisplay will tries to identify the field in the dropdown menu "Fields". If the correspondence fails, the filed must be selected manually in the drop-down menu.





Questions?

If you have any questions about this extension, you may want to refer to the Tesseract Project web site (http://www.typo3-tesseract.com/) for support and tutorials. You may also ask questions in the TYPO3 English mailing list (typo3.english).

Keeping the developer happy

If you like this extension, do not hesitate to rate it. Go the Extension Repository, search for this extension, click on its title to go to the details view, then click on the "Ratings" tab and vote (you need to be logged in). Every new vote keeps the developer ticking. So just do it!

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

Extension "templatedisplay" is part of the Tesseract framework. It will not do anything if installed alone.

After installation you must load the static TypoScript template for proper rendering.

Templatedisplay can easily display a page browser, but this requires extension "pagebrowse" to be installed too.

Requirements

Extension "templatedisplay" requires the PHP Simple XML library.

Versions 1.4.0 and above require TYPO3 4.5 or more.

Upgrading

Please read the sections below carefully to know if you are impacted by changes in some versions.

Upgrading to 1.3.0

In version 1.3.0, the static TypoScript setup was changed to use a reference to lib.parseFunc_RTE, instead of making a copy. This was made so that plugin.tx_templatedisplay.richtext.parseFunc stays in sync with lib.parseFunc_RTE. The drawback is that you cannot make changes like:

```
plugin.tx_templatedisplay.richtext.parseFunc.foo = bar
```

anymore. If you did such changes before, you should first override the reference by a copy and make your change again, e.g.

```
plugin.tx_templatedisplay.richtext.parseFunc >
plugin.tx_templatedisplay.richtext.parseFunc < lib.parseFunc_RTE
plugin.tx_templatedisplay.richtext.parseFunc.foo = bar
```

Configuration

HTML Template setup

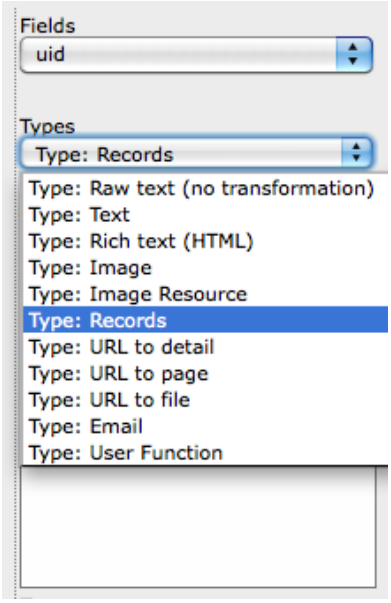
HTML Template can be defined of two manners.

- insert inline HTM directly in the text box
- external file loaded with following syntax: FILE:fileadmin/templates/plugins/tesseract/news/list_of_news.html

External files would have the benefit to make use of an external editor which is more convenient when editing large templates

Element types

There are various elements types that can be chosen according to your needs:



Type	Description	TypoScript Object / function	Default parameters
Raw text	This will display the data from the database field as is, without any transformation. Use this whenever possible, as it gives much better performance.	-	
Text	The local TypoScript is the same as for Object TEXT. The value of the TEXT object will be loaded with the value from the mapped database field.	TEXT	
Rich text	This is actually the same as the "Text"-type element, but it is designed to handle fields that use the RTE. Such fields need special rendering, so that RTE data is interpreted before display in the front-end. As this is not necessary with every text field, a separate element type exists.	TEXT	
Image	The value from the database field will be automatically stored **in the "file" property** of the object.	IMAGE	altText = file_name titleText = file_name
Image Resource	Returns only the image-reference, possibly wrapped with stdWrap. The value from the database field will be automatically stored **in the "file" property** of the object.	IMG_RESOURCE	
Media	Uses the MEDIA content object for rendering, so it can be used for displaying video or audio files. The value from the database field is used automatically for the "file" property.	MEDIA	type = video renderType = auto

Records	Will render records using the TypoScript RECORDS object. The value is expected to be a uid or have the structure "tablename_uid" (e.g. tt_content_38). The mapped value is automatically stored **in the "source" property** of the object.	RECORDS	
URL to detail	This is designed to create a link and corresponds to the typolink TS function. It is automatically loaded with a configuration to create a link to a detail view as expected by the Display Controller (extension: displaycontroller). It is also configured to return only the URL, but this can be overridden in the local TS. The value from the database field will be automatically stored **in the "additionalParams" property** of the object.	typolink	useCacheHash = 1 returnLast = url additionalParams = &tx_displaycontroller[table]=xyz&tx_displaycontrolle[showUid]=$value
URL to page	It should be used only for database fields who contain page id's, as it will create a link to said page. The value from the database field will be automatically stored **in the "parameter" property** of the object.	typolink	returnLast = url useCacheHash = 1
URL to file	Meant for links to files. The value from the database field will be automatically stored **in the "parameter" property** of the object.	typolink	returnLast = url useCacheHash = 1
Email	Meant for emails. The value from the database field will be automatically stored **in the "parameter" property** of the object.	typolink	
User function	It is preconfigured with a property called "parameter" which will contain the value from the database field. The value from the database field will be automatically stored **in the "parameter" property** of the object.	USER	

List of available markers

Content markers

Name	Description
###FIELD.myField###	This is the most common marker that deals with content of the database. When possible, try to make correspond the name of the marker with the name of the field. Templatedisplay will be able to guess automatically the mapping. Click on it to start the mapping process.
###OBJECT.myValue###	Attach some TypoScript to this marker. Same configuration options than FIELD markers but no field associated with.
###LABEL.myField###	The label of the field is translated according to the language of the website. To have a correct translation, the LABEL must have a proper TCA.
###LLL:EXT:myExtension/locallang.xml:myKey###	When no TCA is provided or an external string must be translated, use this syntax for translating a chain of character.
###EXPRESSION.key:var1|var2###	Calls on the expression parser of extension "expressions" to resolve any well-formed expression. **Example:** `###EXPRESSION.gp:clear_cache###` will retrieve the value of a GET/POST variable called "clear_cache".
###FILTER.myTable.myField###	Value of a filter. MyTable is optional and depend of the filter naming.
###SORT.sort###	Value of the sort. The most probably a field name.
###SORT.order###	Value of the order. Can be "ASC" or "DESC"
###SESSION.sessionName.order###	Access information stored in the session.

Name	Description
###COUNTER###	The counter is automatically incremented by 1. This syntax makes sense inside a LOOP and can be used for styling odd / even rows of a table for example. The syntax may looks like this: `<!--IF(###COUNTER### % 2 == 0)-->class="even"<!--ELSE--> >class="odd"<!--ENDIF-->` In the case of a LOOP in a LOOP the second COUNTER remains independent. `<!--LOOP(pages)--> <div>counter 1 : ###COUNTER###</div> <!--LOOP(tt_content)--> <div>counter 2 : ###COUNTER###</div> <!--ENDLOOP--> <!--ENDLOOP-->`
###COUNTER(loop_name)###	This kind of counter is handy in case of LOOP in a LOOP. Let's assume, we need to access the value of the parent COUNTER in a child's LOOP. `<!--LOOP(pages)--> <div>Some value</div> <!--LOOP(tt_content)--> <div>counter pages: ###COUNTER(pages)###</div> <!--ENDLOOP--> <!--ENDLOOP-->`
###PAGE_BROWSER###	If extension "pagebrowse" is installed and correctly loaded, displays a universal page browser. Other page browsers are possible but must be handled with a Hook.
###RECORD(tt_content, 12)###	Call in the template it self an external record. Very handy for including records in a records.
###HOOK.myHook###	See section Hooks
###TOTAL_RECORDS###	Returns the total number in the main of records **without considering a possible limit**. To have a glimpse on the data structure, add the parameter "debug[structure]" in the URL. The value ###TOTAL_RECORDS### corresponds to the cell "**totalCount**" of the main structure (level 1). Make sure you have a backend login to see the table.
###SUBTOTAL_RECORDS###	Returns the total of records in the main data structure **considering a possible limit**. To have a glimpse on the data structure, add the parameter "debug[structure]" in the URL. The value ###SUBTOTAL_RECORDS### corresponds to the cell "**count**" of the main structure (level 1). Make sure you have a backend login to see the table.
###TOTAL_RECORDS(tablename)###	Returns the total of records corresponding to a table name **without considering a possible limit**.
###SUBTOTAL_RECORDS(tablename)###	Returns the total of records corresponding to a table name **considering a possible limit**.
###RECORD_OFFSET###	Return the page offset. The page offset corresponds to the current position inside a global record set. This marker is useful when displaying a page browser. See marker ###PAGE_BROWSER###. You can have something like this: ###RECORD_OFFSET### / ###TOTAL_RECORDS### which will display the current position among the total number of records.

###START_AT###	Return the position of the first record returned by a subset, considering a possible limit.
	
	This marker is useful when displaying a page browser like this one :
	`Records 1 - 10 of 2000 in total`
	
	which would be coded like this in the template:
	`Records ###START_AT### - ###STOP_AT### of ###TOTAL_RECORDS### in total`

###STOP_AT###	Return the position of the last record returned by a subset, considering a possible limit. See ###START_AT### above.

Structure Markers

Name	Description
<!--LOOP(loop_name)--> <!--ENDLOOP-->	Where loop_name is a table name.
<!--IF(###FIELD.maker### == 'value')--> <!--ENDIF-->	Allows to display conditional content. Be careful to use parentheses around the condition. **Warning**: it is not possible to nest IF markers.
<!--EMPTY--> <!--ENDEMPTY-->	This part is displayed only if the Data Structure is empty. Please mind that the rest of the template is still displayed.

Functions

Name	Description
FUNCTION:php_function("###MARKER###" ,parameter1,...)	A PHP function. No simple / double quote required. Examples: FUNCTION:str_replace(P ,X, ###LABEL.title###) FUNCTION:str_repeat(###LABEL.title###,2) FUNCTION:md5(###LABEL.title###)
LIMIT(###MARKER###, 4)	Limit the number of words in a marker E.g. LIMIT(###FIELD.description###, 4) will return the first 4 words of field description
COUNT(tableName)	Return the number of records from the Data Structure. Add parameter debug[structure] in the URL to see the Data Structure. (Works with a BE login) E.g. COUNT(tt_content) will return the number of records in table tt_content
PAGE_STATUS(404) PAGE_STATUS(404, page/404/) PAGE_STATUS(301, new/page/)	If the Data Structure is empty, send the appropriate header and redirect link when needed. For the 404 status, leaving the redirect URL empty will make it fall back on the internal page not found handling of TYPO3.

TypoScript configuration

Default rendering

A default rendering can be defined for each element type. The static template provided with the extension contains the following:

```
plugin.tx_templatedisplay {
      defaultRendering {
            richtext.parseFunc < lib.parseFunc_RTE
      }
}
```

This configuration copies the RTE parseFunc into the parseFunc for the rich text-type element, making possible to render correctly RTE-enabled fields. Here's an example configuration:

```
plugin.tx_templatedisplay {
      defaultRendering {
            text.wrap = <span class="text">|</span>
      }
}
```

This would wrap a span tag with a "text" class around **every** text-type element rendered by Template Display.

Other examples

Example 1: defining the page title according to a field value, useful for a detail view. **Make sure, "Display Controller (cached)" is defined.** Otherwise, "substitutePageTitle" will have no effect.

```
plugin.tx_templatedisplay {
    substitutePageTitle = {title} - {field_custom}
}
```

Example 2: setting the pagebrowse parameters

```
plugin.tx_templatedisplay {
      pagebrowse {
            templateFile = fileadmin/templates/plugins/pagebrowse/template.html
            enableMorePages = 1
            enableLessPages = 1
            pagesBefore = 3
            pagesAfter = 3
      }
}
```

Reference

Name	Value	Description
defaultRendering	Rendering configuration	Default TS configuration for each element type
substitutePageTitle	dataWrap	Substitute page title with values from the datastructure. Instead of having the default page title, it is possible to set an other value
pagebrowse	pagebrowse configuration	See extension pagebrowse

[tsref:plugin.tx_templatedisplay]

Developer's Guide

Hooks

Hooks offer an opportunity to step into the process at various points. They offer the possibility to manipulate data and influence the final output. Hooks can be used to replace personalized markers, introduced previously in the HTML template. There is a convention in templatedisplay to name Hook like ###HOOK.myHook###.

In templatedisplay, there are 2 available hooks:

- preProcessResult (for pre-processing the HTML template)
- postProcessResult (for post-processing the HTML content)

To facilitate the implementation of a hook, a skeleton file can be found in EXT:templatedisplay/samples/class.tx_templatedisplay_hook.php

Step 1

In the ext_localconf.php of your extension, register the Hook.

```
$GLOBALS['TYPO3_CONF_VARS']['EXTCONF']['templatedisplay']['postProcessResult']['myHook'][] =
'EXT:templatedisplay/class.tx_templatedisplay_hook.php:&tx_templatedisplay_hook';
```

Remarks:

- "postProcessResult" can be replaced by "preProcessResult".
- "myHook" can be something else but must correspond to the marker ###HOOK.myHook###.
- Make sure the path of the file is correct and suit your environment.
- Don't forget to clear the configuration cache!!

Step 2

Write the PHP method that will transform the content.

```
class tx_templatedisplay_hook {
	public function postProcessResult($content, $marker, &$pObj) {
		$controller = $pObj->getController();
		$data = $controller->cObj->data;
		if ($data['uid'] == 11399) {
			$_content = '';
			$content = str_replace('###HOOK.myHook###', $_content, $content);
		}
		return $content;
	}
}
```

Custom element types

It is possible to define custom element types. Such types will be added to the list of available types in the mapping interface, which makes them easier to use for users than the user-function type.

As for hooks this is a two-step process.

Step 1

Register the custom type in ext_localconf.php file of your extension. The syntax is as follows:

```
$GLOBALS['TYPO3_CONF_VARS']['EXTCONF']['templatedisplay']['types']['tx_test_mytype'] = array(
	'label'		=> 'LLL:EXT:' . $_EXTKEY . '/locallang.xml:mytype',
	'icon'		=> 'EXT:' . $_EXTKEY . '/mytype.png',
	'class'		=> 'tx_test_templatedisplay'
);
```

The custom type is registered with a specific key (e.g. "tx_test_mytype") and with the following information:

- a label that will appear in the drop-down list of available element types (as well as alt text for the icon)
- an icon that will appear in the mapping interface when that type has been selected
- a class to do the processing of that custom type. The class must implement the tx_templatedisplay_CustomType interface (more below).

You also need to include the class that will do the processing. As of TYPO3 4.3 you can register it with the autoloader instead (this is the preferred way).

Step 2

The method itself is expected to do the rendering. It receives the following parameters:

Parameter	Type	Description
$value	mixed	The current value of the field that was mapped.
$conf	array	TypoScript configuration for the rendering (this may be ignore if you don't need TypoScript).
$pObj	object	A reference to the calling tx_templatedisplay object.

A sample implementation is provided in the "samples/class.tx_templatedisplay_phonetype.php" file. The code looks like this (without comments):

```
class tx_templatedisplay_PhoneType implements tx_templatedisplay_CustomType {
	function render($value, $conf, tx_templatedisplay $pObj) {
		$rendering = '<a href="callto://' . rawurlencode($value) . '">' . $value . '</a>';
		return $rendering;
	}
}
```

In this simple example the class just does some minor processing with the value it receives and returns the result.

As of TYPO3 4.3, it is recommended that such classes also implement the t3lib_Singleton interface so that only one instance of it is created (otherwise one instance is created for each field using this custom type on each pass in the loop). This will save memory.

Debugging

Debugging is provided in form of parameters added in the URL. A backend login is mandatory in order to see the output.

Name	Description
debug[structure]	Display the current data structure. Useful to see which data are given to templatedisplay.
debug[template]	Display the structure of the template. The template is cut in small pieces for processing according the LOOP and SUBLOOP.
debug[filter]	Display the active filter.
debug[markers]	Display the list of markers and their values.
debug[display]	Display the untranslated markers. By default, the production mode is active. It means when a translation does not succeed, the marker is erased from the final output. For debug purpose, it might be useful to identify these markers.
debug[pagebrowse]	Display the parameter that are transmitted to the pagebrowser. Extension pagebrowse is expected here.

Whenever the extension "devlog" is installed, it is also possible to monitor some information. In a templatedisplay record, check the following options as desired. It may be convenient, in (pre) production mode to check some information produced by visitor.

Known issues

Nested IF markers don't work

```
<--IF()-->
  ...
  <--IF()-->
    ...
  <!--ENDIF-->
  ...
<!--ENDIF-->
```

This feature seems to be obvious, but is quite difficult to implement, though (at least, with the actual code base). It comes from the way the template engine works and particularly the general use of regular expressions to handle the HTML. It would require a complex analysis of the template to cut up in right parts and supbarts the "IF" markers.

Experience has shown that it's possible to live without this feature. If the need of nested "IF" markers is required, you many want to have a look at the "phpdisplay" or "fluiddisplay" Data Consumers.

http://forge.typo3.org/issues/show/1954

Multiple edition is not possible (module web > list)

This would require too much effort for very small benefit. The cases of multiple edition in templatedisplay are very rare.

http://forge.typo3.org/issues/show/1982