

EXT: Display Controller

Extension Key: displaycontroller

Language: en

Keywords: forEditors, forIntermediates

Copyright 2000-2013, François 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

EXT: Display Controller.....	1	Redirection mechanism.....	11
Introduction.....	3	Page TSconfig.....	12
What does it do?.....	3	Hooks.....	13
Screenshots.....	3	RealURL.....	13
Questions?.....	3	Tutorial.....	16
Keeping the developer happy.....	4	Scenario.....	16
Installation.....	5	Preparing for work.....	16
Compatibility.....	5	Querying the database.....	16
Configuration.....	6	Preparing a template for display.....	17
Checking the installation.....	7	Fitting it all together.....	18
Users manual.....	8	Improving the display.....	20
The Display Controller.....	8	Next steps.....	22
Options.....	9	Frequently Asked Questions.....	23
Cached or not cached?.....	9	Troubleshooting.....	23
Query parameters.....	9	Known problems.....	24
Configuration.....	11	To-Do list.....	25
TypoScript.....	11	Appendix A – Tutorial answers.....	26

Introduction

What does it do?

This extension is a FE plugin for the Tesseract Project. It connects a Data Consumer with one or more Data Providers and Data Filters in order to display data in the TYPO3 FE.

If this sounds like gibberish please go straight to the Tutorial chapter of this manual which explains all the basic concepts. For an even quicker introduction you can refer to the tutorial screencast found on the Tesseract web site (<http://...>).

Screenshots

An example of the power of the Tesseract Project: a SQL-based query, a HTML template with a point-and-click mapping interface, the Display Controller to relate these two components together, and a result in the frontend.

The screenshots illustrate the workflow of the Display Controller extension:

- General Configuration:** The 'SQL Query' field is set to `SELECT realName,email FROM be_users`.
- Mappings:** The 'Mapping' tab shows an HTML template using TESSERACT tags to map the query results to an email field. The template is:


```
<u1>
  <!--LOOP(be_users)-->
  <li>###FIELD.realName### (###FIELD.email###)
  <!--ENDLOOP-->
</u1>
```
- Data Objects:** The 'Data Consumer' is set to 'List of BE users' and the 'Primary Data Provider' is 'BE Users'. The 'Type of filter' is set to 'None'.
- Frontend Output:** The frontend displays a list of administrators under the heading 'Your administrators':
 - admin
 - Simple McEditor ([username@example.com](#))
 - Advanced McEditor ([username@example.com](#))
 - News McEditor ([username@example.com](#))

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 make some buzz about it.

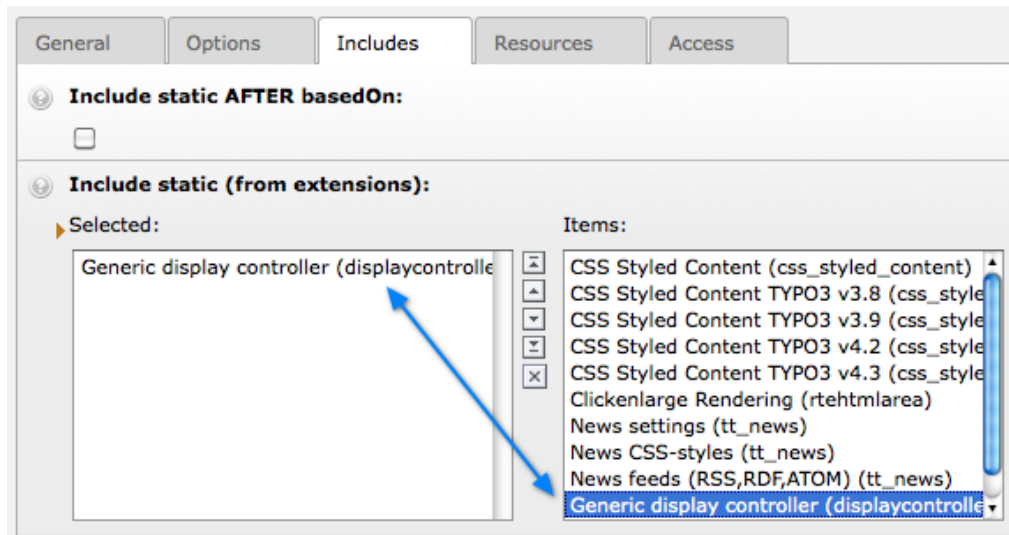
You may also take a step back and reflect about the beauty of sharing. Think about how much you are benefiting and how much yourself is giving back to the community.

Installation

The Display Controller must be installed as part of the Tesseract suite of extensions. It is useless outside of this scope.

The extension will add a number of fields to the tt_content table and creates a new table used to store the relations between the various Tesseract components.

After installation the static TypoScript template provided by the extension should be added to the site's template.



Compatibility

As of version 1.4.0, TYPO3 4.5 or more is required.

Configuration

Here are the configuration options available after installing the extension:

- **Debug:** Choose the type of debugging that you want. Debugging to the output will display status messages right before the Display Controller's normal output in the frontend (see screenshot below). Debugging to the Developer's Log will log messages provided you have an extension enabling that log (such as "devlog"). It's also possible to debug to both. Choosing "None" will turn all debugging off. Debugging can also be activated on a content element basis (see User Manual below).

The icons show the status. Rolling over displays the debug title and message (including the key of the extension which registered the message). Clicking on an icon will dump any extra debug data to the JavaScript console.

The screenshot shows a web page with a sidebar containing Lorem Ipsum text. The main content area features a section titled "Dernière nouvelles" with a sub-header "sample 1". A tooltip is visible over the "sample 1" header, displaying a debug message: "[templatedisplay]: Image not found for marker: FIELD.image". Below the tooltip, the text "Lire la suite..." is visible. The bottom of the screenshot shows a JavaScript console log with the following message:

```
[templatedisplay]: Image not found for marker: FIELD.image
▼ Object {file.: Object, typolink.: Object, file: "sample_1.jpeg", altText: "sample_1", titleText: "sample_1"}
  altText: "sample_1"
  file: "sample_1.jpeg"
  file.: Object
    import: "uploads/tx_news/"
    import.: Object
    maxHeight: "90"
    width: "120"
    __proto__: Object
  titleText: "sample_1"
  typolink.: Object
  __proto__: Object
```

The console log is located at the bottom of the page, with the file path "index.php:301" visible.

- **Minimum level for debugging:** only message with the chosen level of severity or above will be logged for debugging. All others are ignored. This makes for a less verbose output.
- **Debug output class:** this makes it possible to create a custom class for rendering the debug output. This class will be used instead of `tx_displaycontroller_debugger`. It should be properly declared in some extension's `ext_autoload.php` file. It **must** extend `tx_displaycontroller_debugger` and override the `render()` method.

Checking the installation

The loading order of the various Tesseract components is of importance. The best way to check if everything is properly installed is to create a new Display Controller content element and check the "Data Objects" tab.

The screenshot shows the 'Data Objects' tab of a configuration interface. It contains two main sections: 'Data Consumer' and 'Primary Data Provider'. Both sections have a selection field with a red circle around it. Below the 'Data Consumer' section, there are two options: 'Template-based Display' and 'Fluid based Display'. Below the 'Primary Data Provider' section, there is a 'Type of filter' section with four radio buttons: 'None' (selected), 'Detail view', 'List view', and 'Advanced filter (choose Data Filter below)'. To the right of this is the 'Advanced Data Filter (primary)' section, which has a selection field with a red circle around it. Below this is a 'Data Filter' section with a selection field. To the right of the 'Data Filter' section is the 'If filter empty' section with two radio buttons: 'Display nothing' (selected) and 'Display everything'. The 'Display nothing' radio button is circled in red.

If no wizards icons appear for adding new Tesseract components, it means that the "displaycontroller" extension is loaded **after** the others (like "dataquery", "datafilter", etc.). You should change the loading order of the extensions so that "displaycontroller" comes **before** such components (but still after "tesseract" itself).

The safest way is to uninstall everything and re-install all extensions following the order advised by the configuration options of extension "tesseract". If you know what you're doing, you can also directly edit your "localconf.php" file.

Users manual

If you have never used a Display Controller before, please refer to the tutorial further down in this manual.

The Display Controller

This section describes all the options available when creating a new Display Controller content element.

The screenshot shows the 'Data Objects' configuration tab. It contains three main sections:

- Data Consumer:** A dropdown menu with a warning icon, and two options: 'Template-based Display' and 'Fluid based Display'.
- Primary Data Provider:** A dropdown menu with a warning icon, and two options: 'Data Query' and 'Google Queries'. Below this is a 'Type of filter' section with radio buttons for 'None', 'Detail view', 'List view', and 'Advanced filter (choose Data Filter below)'. To the right is an 'Advanced Data Filter (primary)' dropdown with a 'Data Filter' icon. Further right is an 'If filter empty' section with radio buttons for 'Display nothing' and 'Display everything'.
- Secondary Data Provider:** A dropdown menu with a warning icon, and three options: 'Data Query', 'Google Queries', and 'MM-tables selection'. Below this is an 'Advanced Data Filter (secondary)' dropdown with a 'Data Filter' icon. To the right is an 'If filter empty' section with radio buttons for 'Display nothing' and 'Display everything'.

At the bottom, there is a section titled 'If secondary provider returns nothing' with radio buttons for 'Display nothing' and 'Display everything'.

Data Consumer

This is a reference to a Data Consumer record. The field provides wizards for selecting a record, edit the current record or create a new Data Consumer (one such wizard appears per available Data Consumer type). This field is required as nothing will be display otherwise.

Primary Data Provider

This is a reference to the Data Provider that will fetch the data to display. The field provides wizards for selecting a record, edit the current record or create a new Data Provider (one such wizard appears per compatible Data Provider type). This field is required as nothing will be display otherwise.

Type of filter

Choose the type of filter to apply to the Data Provider. The "Detail view" and "List view" options will tell the Display Controller to define a filter using the default GET/POST variable names for both views. Check the "Advanced filter" option if you want to apply a Data Filter record, which has to be selected separately (see below).

Advanced Data Filter (primary)

This is a reference to a Data Filter record that will apply to the Primary Data Provider. The field provides wizards for selecting a record, edit the current record or create a new Data Filter (one such wizard appears per available Data Filter type). *In order for this filter to be active, it is also necessary to select "Advanced filter" from the "Type of filter" field (see above).*

If filter empty

With this field you can choose the behavior of the Display Controller when the filter results in no conditions at all. This may happen with a search field, for example: when the user submits an empty search, the filter will also be empty. You must then choose whether you want to display everything or nothing in such a case.

Secondary Data Provider

This is a reference to the Data Provider that feeds into the Primary Provider. The field provides wizards for selecting a record, edit the current record or create a new Data Provider (one such wizard appears per compatible Data Provider type).

The Secondary Data Provider is meant to return lists of primary keys which will then restrict the list of records returned by the Primary Data Provider.

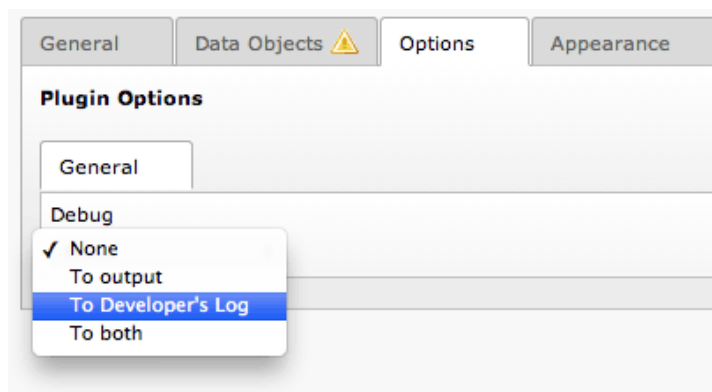
A Data Filter can be applied to the Secondary Data Provider too. The same choice as described above must be made in case that filter ends up being empty.

If secondary provider returns nothing

As with Data Filters it is necessary to define the behavior of the Display Controller should the Secondary Data Provider (if defined) return nothing. Here again the choice is to display either everything or nothing.

Options

The "Options" tab contains additional, minor settings. For now it contains only a debug flag. This is the same as the flag from the extension configuration, but for the current content element only. This makes it easier, for example, to find information in the devlog, instead of having all Display Controllers produce debug output.



Cached or not cached?

The Display Controller fronted plug-in comes in two versions: one cached and one not cached. How to choose which one to use?

The general rule is that the cached version should be preferred whenever possible for obvious performance reasons. However in some circumstances using the non-cached version cannot be avoided. A typical use of the non-cached version is when performing searches: if there are a lot of possible search criteria, it is not desirable or even possible to store every combination of search values into cache.

Query parameters

Since the Display Controller exists in two versions (pi1 and pi2), it uses a common naming for its GET/POST variables, i.e. `tx_displaycontroller[xxx]`, so that both plug-ins use the same syntax. Any variable named according to that scheme will be available in the piVars of the Display Controller. This also means that they will be available as "vars:" in the "datafilter" extension, since it relies on "expressions".

Furthermore the Display Controller recognizes a number default GET/POST variable names with which it builds the "Detail view" and "List view" described above:

- `tx_displaycontroller[table]`: table used for the detail view
- `tx_displaycontroller[showUid]`: primary key of the record to display in the detail view. Used in conjunction with the "table" parameter above, this uniquely defines a given record
- `tx_displaycontroller[max]`: for the list view with page browsing, how many records to display at a time

- `tx_displaycontroller[page]`: for the list view with page browsing, number of the current page (expected to start at 0)
- `tx_displaycontroller[sort]`: for the list view, field on which to sort the records (may use the syntax "table.field")
- `tx_displaycontroller[order]`: order for sorting ("asc" or "desc", defaults to "asc").

Extension "templatedisplay" has an object type that builds links to detail views using the variables names described above. It also uses the proper variables names when creating a page browser. Note that these variables' names are not hard-coded, but are provided by the controllers themselves via an API.

Configuration

TypoScript

Property:	Data type:	Description:	Default:
redirect	-> redirectConfig	Use this configuration to redirect the process to some other page, based on a condition.	
listView	-> listViewConfig	Configuration for the list view. This is generally default values that may be superseded by GET/POST parameters.	
detailView	-> detailViewConfig	Configuration for the detail. Currently this is just about setting a different keyword for the RealURL postVarSets (see "RealURL" below).	

[tsref:plugin.tx_displaycontroller]

-> redirectConfig

Property:	Data type:	Description:	Default:
enable	boolean / stdWrap	Enable the redirection mechanism at all (redirection still depends on the condition below)	
condition	-> if	Condition for the redirection to take place. Note that you have the following data available: <ul style="list-style-type: none"> the total count of records in the SDS in a register called "sds.totalCount" the count of records in the SDS in a register called "sds.count" the data in the first record of the SDS, loaded in the content object (and thus available with the "field" syntax in the getText function) 	
url	-> typolink	URL to redirect to. Note that data from the first record of the SDS is available here too, as described above.	

[tsref:plugin.tx_displaycontroller.redirectConfig]

-> listViewConfig

Property:	Data type:	Description:	Default:
limit	int	Maximum number of records to display	10
sort	string	Field name for default sorting of results. Syntax is similar to SQL, i.e. <code>tablename.fieldname</code> (table name can be omitted)	
order	string	Default ordering of records. Acceptable values are asc or desc .	

[tsref:plugin.tx_displaycontroller.listViewConfig]

-> detailViewConfig

Property:	Data type:	Description:	Default:
postVarSets	string	Name of the key that defines the postVarSets configuration for RealURL.	item

[tsref:plugin.tx_displaycontroller.listViewConfig]

Redirection mechanism

The Display Controller comes with a flexible redirection mechanism whose properties described above. Here is an example configuration:

```
plugin.tx_displaycontroller.redirect {
    enable = 1
    condition {
        value.data = register:sds.totalCount
        equals = 1
    }
    url {
        parameter = 15
    }
}
```

In this example, the redirection is first enabled. Then the condition for the redirection to actually take place is set. It will happen when the total number of records in the SDS is equals to 1. Finally the url for to redirect to is defined as a typolink to page 15.

Don't forget that TypoScript is inherited along the page tree. So a redirection configuration defined at some point may not be desirable at some other point down the page tree. Either disable it:

```
plugin.tx_displaycontroller.redirect.enable = 0
```

or cancel it altogether:

```
plugin.tx_displaycontroller.redirect >
```

Page TSconfig

The Display Controller comes with a Page TSconfig option that makes it possible to act on the "Save and view" button in the BE and call up a properly configured preview page for any record (assuming that page contains appropriately configured Tesseract components, of course).

The basic syntax is:

```
tx_displaycontroller.tx_foo_bars {
    previewPid = 1498
    parameters = &item=###id###&L=###lang###
}
```

where "tx_foo_bars" is the name of the table. The properties are:

Property:	Data type:	Description:	Default:
previewPid	integer	The id of the page that should be called up for preview.	
parameters	string	<p>The list of query string variables that should be added to the call to the preview page, so that it receives all the information that it needs to display the record being "saved and viewed".</p> <p>Three markers can be used in this string:</p> <ul style="list-style-type: none"> ###id### is replaced by the id of the record that has just been saved ###table### is replaced by the name of the table being handled ###lang### is replaced by the id of the current language (0 for the default language) <p>The Display Controller automatically adds "&no_cache=1".</p>	&tx_displaycontroller[table]=###table# ##&tx_displaycontroller[showUid]=### id###&L=###lang###

Whatever syntax you put in the parameters should obviously match the Data Filter used on the preview page, and whatever Data Consumer exists there should also be able to display a record from the given table.

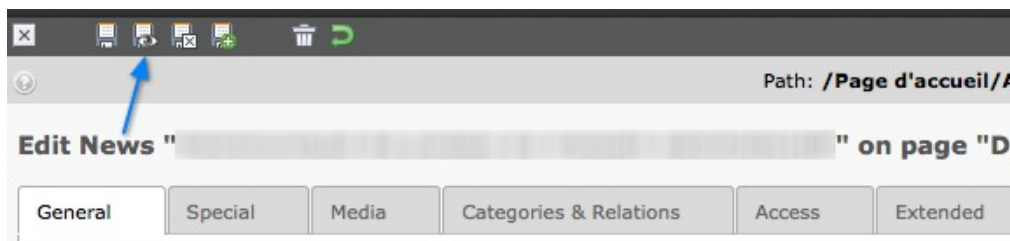
Example

Given the following Page TSconfig:

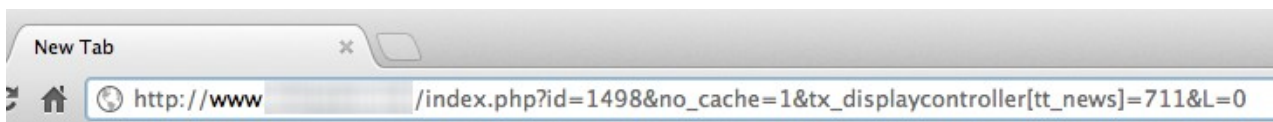
TypoScript Configuration
Page TSConfig

```
tx_displaycontroller.tt_news {
    previewPid = 1498
    parameters = &tx_displaycontroller[tt_news]=###id###&L=###lang###
}
```

when the "Save and view" button is hit



the following URL is called up:



Warning

This preview mechanism may conflict with other such mechanisms. This happens – for example – with extension "linkhandler". The problem with "linkhandler" is that the same config (mod.tx_linkhandler) is used for both creating the extra tabs and previewing. So you can't disable one while keeping the other. If you encounter this problem, the simplest solution is probably to deactivate the hook registered by "linkhandler". Edit file "ext_localconf.php" and comment out the following lines:

```
include_once t3lib_extMgm::extPath($_EXTKEY) . 'service/class.tx_linkhandler_tcemain.php';
$GLOBALS['TYPO3_CONF_VARS']['SC_OPTIONS']['t3lib/class.t3lib_tcemain.php']['processDatamapClass'][] =
'EXT:' . $_EXTKEY . '/service/class.tx_linkhandler_tcemain.php:tx_linkhandler_tcemain';
```

Hooks

There's one hook defined in the Display Controller. It can be used to manipulate filters after their initialization. At this point a filter may either be empty or already have some structure read from the session cache.

The hook must be registered that way:

```
$GLOBALS['TYPO3_CONF_VARS']['EXTCONF']['displaycontroller']['extendInitFilter'][] =
'myextension/class.tx_myextension_hook.php:&tx_myextension_hook';
```

and a typical implementation might look like:

```
class tx_myextension_hook {
    public function extendInitFilter($filter, $parentObject) {
        // Do some changes to the filter
        return $filter;
    }
}
```

The method to implement is called `extendInitFilter` and it receives 2 arguments. The first one is the array containing the filter data and the second one is a reference to the Display Controller itself. The method **must** return the modified filter, even if no changes were made to it.

RealURL

The Display Controller provides a user object for creating speaking URLs for any link to a single record in conjunction with RealURL. A typical configuration will look like:

```
$TYPO3_CONF_VARS['EXTCONF']['realurl']['_DEFAULT']['postVarSets']['_DEFAULT'] = array(
    'item' => array(
        array(
            'GETvar' => 'tx_displaycontroller[table]',
            'valueMap' => array(
                'news' => 'tt_news'
            )
        ),
        array(
            'GETvar' => 'tx_displaycontroller[showUid]',
            'userFunc' =>
'EXT:displaycontroller/class.tx_displaycontroller_realurl.php:&tx_displaycontroller_realurl->main',
            'userFunc.' => array(
                'tt_news' => array(
                    'alias_field' => 'title',
                )
            )
        )
    ),
);
```

```
);
```

Note how the `postVarSets` configuration uses the `"item"` key (in bold above). This is the default key expected by the Display Controller's `RealURL` user-function. It's possible to use another key, but it must be defined in the `TypoScript` configuration:

```
plugin.tx_displaycontroller.detailView.postVarSets = foo
```

When creating a link to a single record, the Tesseract requires the link to be built using `tx_displaycontroller[table]` and `tx_displaycontroller[showUid]` query parameters, the first one containing the table's name and the second the primary key of the record. The Display Controller provides `RealURL` with a user object to transform these parameters into a speaking URL.

In the above configuration the first parameter (`tx_displaycontroller[table]`) is mapped to a list of tables. The key used in the array will be the name of the table as a segment in the speaking URL. The second parameter (`tx_displaycontroller[showUid]`) refers to a user function for generating the alias with the proper field (of the proper table, since it will refer to `tx_displaycontroller[table]`).

It is very important to keep these two configurations (for `"table"` and for `"showUid"`) exactly in that order because it is expected to be this way. Setting a different order or using other indices in the configuration array will break the encoding/decoding process.

If the `"alias_field"` property is missing, the Display Controller will use the `"uid"` field as a fall-back.

The `"alias_field"` property in the configuration can be more than a simple string. It can contain expressions that will be interpreted by the expressions parser. Example:

```
'alias_field' => 'header_{date:Y}'
```

Another feature is to use a marker called `###LANG###` which contains the 2-letter ISO code associated with the language in which the URL is being generated. The code is deducted from the language value map entered in the `RealURL` configuration. Given the following configuration:

```
$TYPO3_CONF_VARS['EXTCONF']['realurl']['_DEFAULT']['preVars'] = array(
    array(
        'GETvar' => 'L',
        'valueMap' => array(
            'fr' => '0',
            'en' => '1',
            'de' => '2',
            'it' => '3'
        ),
        'valueDefault' => 'fr',
        'noMatch' => 'bypass',
    ),
);
```

the `###LANG###` marker will contain `"fr"` when language is 0 or undefined (default), `"en"` when language is 1, etc. The `"alias_field"` configuration would then look like:

```
'alias_field' => 'header_###LANG###'
```

Note that to make the language configuration more easily accessible for the Display Controller, it is possible to use the `"lang"` key as index to the language configuration. So the above configuration would be written as:

```
$TYPO3_CONF_VARS['EXTCONF']['realurl']['_DEFAULT']['preVars'] = array(
    'lang' => array(
        'GETvar' => 'L',
        'valueMap' => array(
            'fr' => '0',
            'en' => '1',
            'de' => '2',
            'it' => '3'
        ),
        'valueDefault' => 'fr',
        'noMatch' => 'bypass',
    ),
);
```

Note the part in bold.

Troubleshooting

If anything seems wrong with the speaking URLs, don't forget to check both ends of the process, i.e.:

- if the link is assembled with properly named variables (i.e. `tx_displaycontroller[table]` and `(tx_displaycontroller[showUId])`)
- if the RealURL configuration is correct (i.e. all tables are listed and an alias field is defined for each)

Activating the “debug” flag in the extension's configuration will log information to the Developer's Log, which may help you track errors in the generation of speaking URLs.

Tutorial

This tutorial is designed not only to explain how the Display Controller itself works, but – more generally – what the whole Tesseract Project is about and what advantages it brings compared to more usual ways of working with TYPO3.

The whole point of the Tesseract Project is to provide a set of tools (extensions) that make it easy to retrieve data from a variety of sources (but in particular the TYPO3 database) and display them in the frontend without having to develop custom extensions every time.

In order to highlight the benefits of the Tesseract Project, this tutorial was designed to show in the frontend a very unusual piece of information: a list of BE users.

Scenario

Here's what we want to achieve in this tutorial: imagine a web site – possibly an intranet – where we want to have a list of BE users somewhere in the frontend, so that visitors can easily contact the web site administrators.

This tutorial will show that this can be achieved with a minimum of efforts and a minimum of redundant components. Future tutorials on the Tesseract web site (<http://www.tesseract-typo3.com/tutorials/>) will take this example further, e.g. showing how to search, sort and filter this list.

Preparing for work

Before starting the actual work, create a new page somewhere in your page tree so that we can start with a blank slate. Call it "Web site administrators".

Place yourself on this new page, in Web > List mode.

Querying the database

As a first step we will use Data Query to get a list of all BE users. We want to display their names and email addresses so that the web site users can contact them easily. Create a new "Data Queries" record and type in the SQL query:

The screenshot shows the 'Data Query' configuration window in TYPO3. It has two tabs: 'General' and 'Advanced'. Under 'General', there is a 'Hide' checkbox which is unchecked. Below that is the 'Title' field with the value 'BE Users'. Underneath is the 'Description' field with the value 'Get list of all BE users'. At the bottom is the 'SQL Query' field, which is highlighted with a blue rectangular box and contains the text 'SELECT realName,email FROM be_users'. To the right of the SQL field is a button labeled 'Validate Query'.

Data Query takes care of all "special" fields automatically, so you don't need to care about them. In this case it will transparently handle the "disabled" field meaning that disabled BE users will not be listed, although we didn't specify this condition explicitly in the SQL query.

WARNING: be sure to type the query as it appears in the above screenshots, with the SQL keywords in uppercase. Data Query requires SQL keywords to be in uppercase.

Let's not worry about the other properties for now. Just save and close the Data Query record.

Preparing a template for display

The next step is to start preparing the display for the results of our query. To achieve this, create a new record of type Template-based Display. Just enter a title and save. You should have a screen that looks like this:

The main action happens in the mapping "Fields". At the moment there's nothing we can do in the "Mapping" tab, but we can switch to the "HTML" tab and start defining a template for displaying the list of BE users. Enter the following HTML:

The basic idea is to display the BE users as a bulleted list. So the first step is to open and close a `` tag. Inside that tag we define a loop on the "be_users" table, using the syntax which is explained in more details in the manual of the Template Display extension. Inside the loop, we create a `` tag for each user and decided to display the name and the email side-by-side, the email being wrapped in brackets.

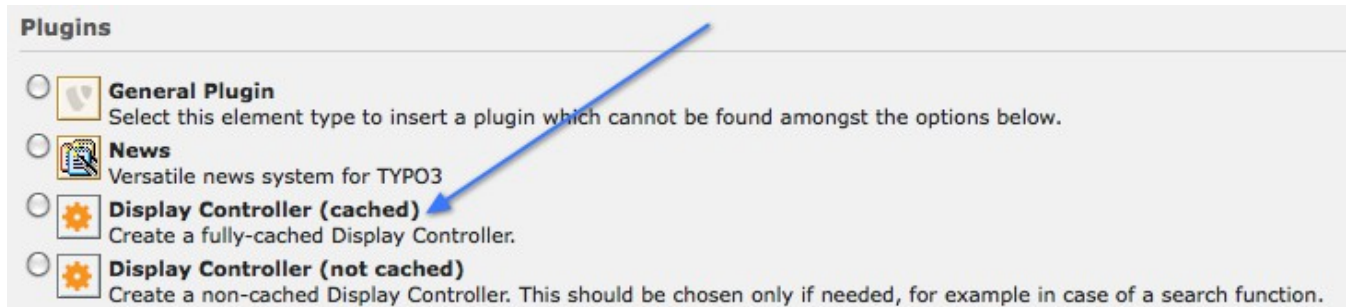
We can move back the the "Mapping" tab. The HTML that we just entered is saved in the background. You should now have the following view:

As you can see that HTML was parsed and some elements were recognized and highlighted: the loop and the "field markers". Those markers are where the data from the database will come. They currently have warning icons over them because they are not yet related to a database field, even though they use the same name. In the current situation, the "Data Queries" record that we created in the previous step and the template are not related in any way.

In the next step we will define the actual content element and make the relation between the query and the template. For now you can save and close the "Template-based Display" record.

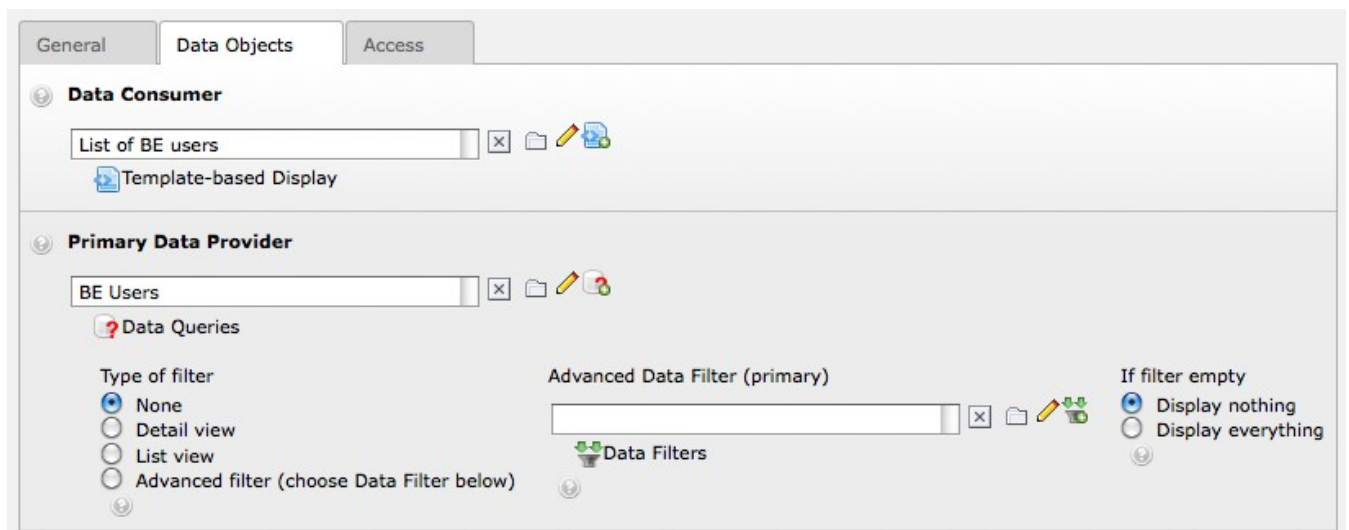
Fitting it all together

It is finally time to use the Display Controller itself. Move to the Web > Page module and create a new "Display Controller (cached)" by choosing from the new content element wizard:

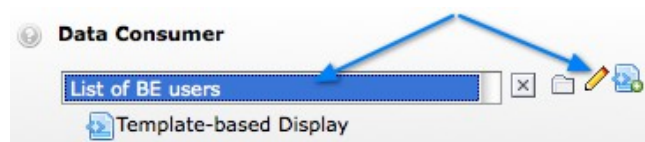


See the "User's manual" chapter for a discussion on the difference between the cached and non-cached versions.

Switch to the "Data Objects" tab to create the relations between the components we defined earlier. For the "Data Consumer" select "Template-based Display" record that we just created. For the "Primary Data Provider" select the "Data Queries" record created in the first step. Click on the "Save" button. Your screen should look like this:

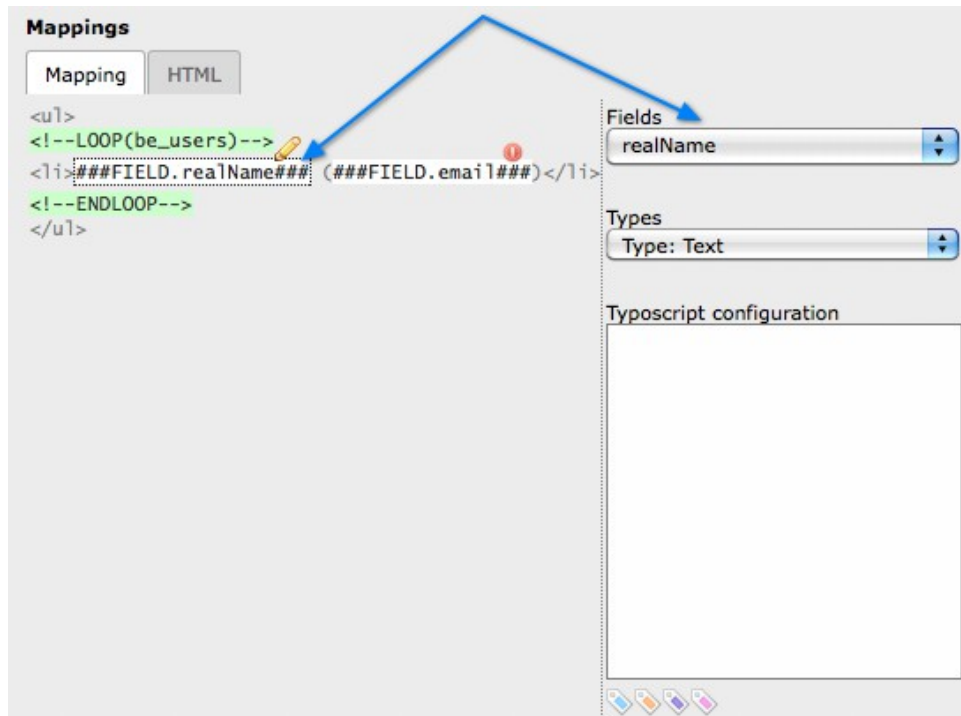


Now the "Template-based Display" record has been set in relation with the "Data Queries" record via the Display Controller. It's time to edit the "Template-based Display" record and define the relations between the database fields and the template markers. Select the "Template-based Display" record and click on the pencil icon:

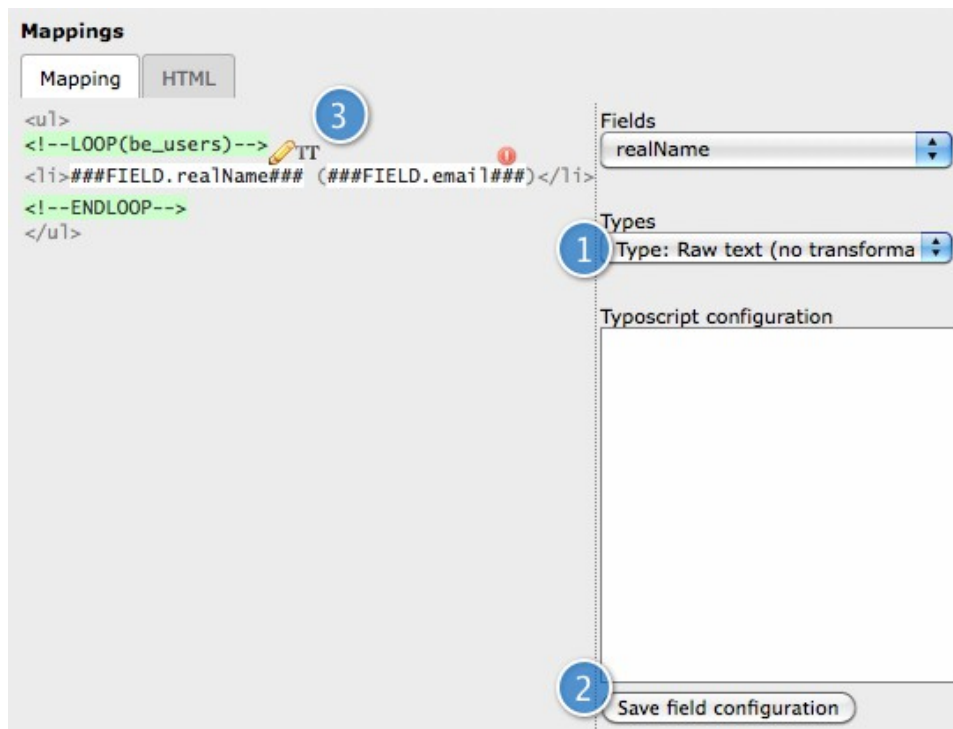


In the new window, click on the `###FIELD.realName###` marker. Notice the drop-down menu that has appeared on the right side. Also note that the "realName" field is preselected in this drop-down menu. That's because the marker used the exact same name as the database field.

Below the "fields" drop-down menu is another selector called "Types". This defines how the field should be rendered. You will notice that some items in this selector bear names that resemble TypeScript content objects. This is no coincidence. For example a field of type "text" will be rendered using a TEXT object. The box below the selector allows you to enter TypeScript corresponding to the type that you selected. The Template Display manual has a list of all types and their corresponding TypeScript objects or functions.



In this case we don't want to do anything special with the administrator's name, so we don't need any TypoScript rendering. Hence we choose the "Raw text (no transformation)" type and click the "Save field configuration" button below the TypoScript configuration field.



Notice how the little icon next to the pencil changed after clicking on "Save field configuration". This is a quick visual clue as to the type of the field.

Let's move on to the next field where we will use some TypoScript. Indeed we want the e-mail address to be clickable. So the steps to take are:

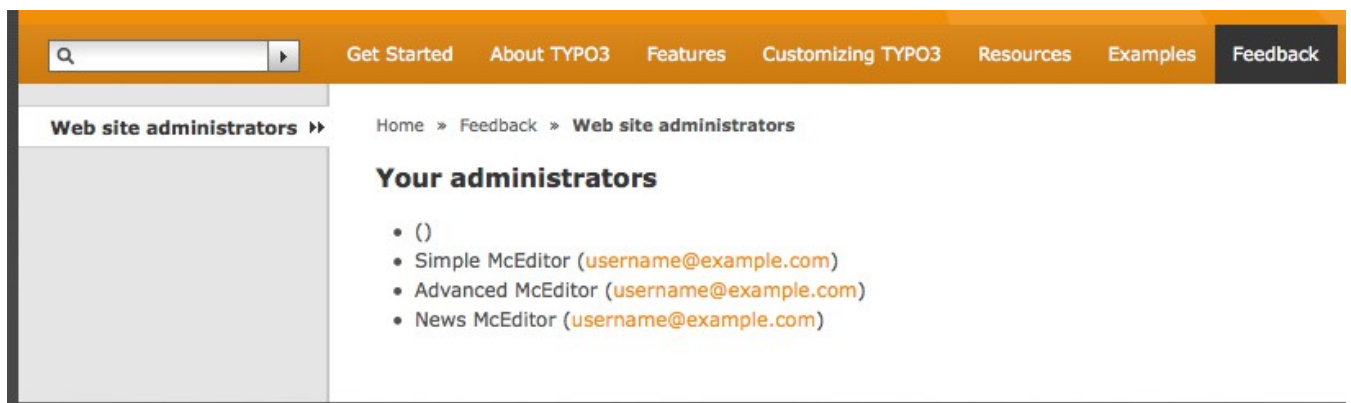
1. click on the `###FIELD.email###` marker
2. verify that the "email" field is indeed selected in the "Fields" selector
3. choose the type "Email" from the "Types" selector

4. click on "Save field configuration"

Your display should look like this:

The "Email"-type field corresponds to a "typolink" function. It expects that the field it is mapped to contains an e-mail address, otherwise it will just create a typolink to whatever else.

We are ready for the big jump! Save and close the edit window and then view the page we have been working on. You should see a list of BE users with a clickable e-mail address (the screenshot is based on the Introduction Package):



What's wrong with the first line? That's quite simple: this BE user has no real name and no e-mail address defined. This gives us a good opportunity to improve our example.

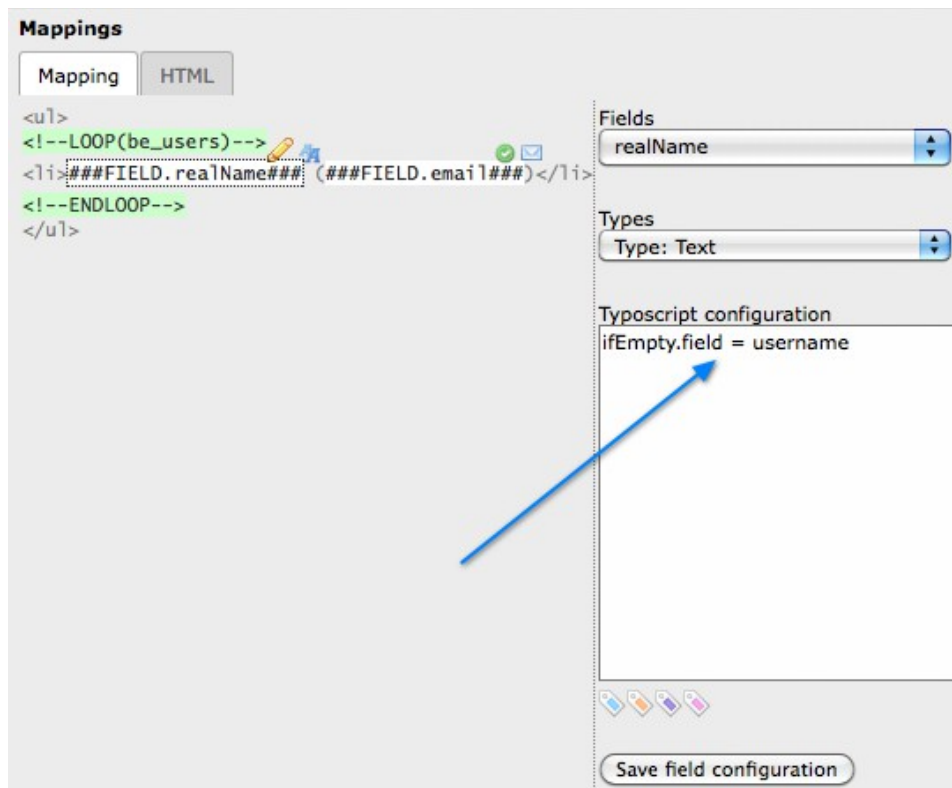
Improving the display

A first approach could to just filter out users that have no real name or e-mail address in the SQL query itself. This is a bit rough. We can do better with some more TypoScript in the display.

First of all let's edit the SQL query so that we have the username at disposal too. Edit the "Data Queries" record and add the "username" field to the list of selected fields:

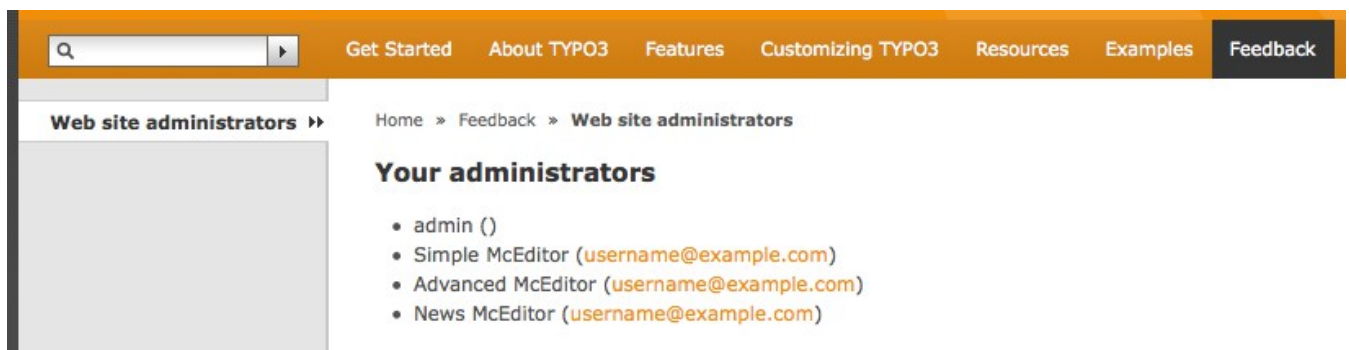
Now let's edit the "Template-based Display" record again. We want to achieve the following: if the real name is not defined, we want to display the username instead. This will be possible only with TypoScript so we have to change the type of the "realName" field to "text" and then enter the following TypoScript:

```
ifEmpty.field = username
```



NOTE: don't forget to click the "Save field configuration" button every time you make a change either to the "Fields" selector, the "Types" selector or the "TypoScript configuration" field.

Let's look at the result in the frontend:



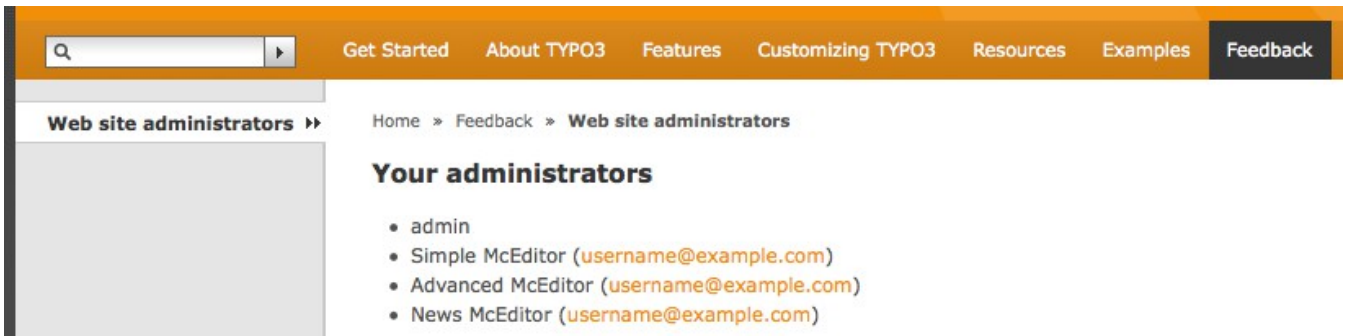
The username has indeed replaced the real name for the first user.

How was that possible? For each record handled inside the loop, Template Display loads all fields from the record into the local content object used for rendering. Thus every field is available in the "field" property of stdWrap. This makes it very easy to use the data from the database to create sophisticated renderings. It is this case we are using this together with the "ifEmpty" property, so that the username gets displayed when the real name is blank.

The second improvement would be to avoid displaying empty brackets when no e-mail address is defined. This is left as an exercise to the reader. The answer can be found in "Appendix A – Tutorial answers". Just a hint:

- the "Email"-type field corresponds to a typolink function. This implies that we will not have the TypoScript available to avoid displaying empty brackets. The type of the field will have to be changed and the "mailto:" link built differently. And the HTML template has to be changed too, of course.

The result should be as follows:



Web site administrators >>

Home » Feedback » Web site administrators

Your administrators

- admin
- Simple McEditor ([username@example.com](#))
- Advanced McEditor ([username@example.com](#))
- News McEditor ([username@example.com](#))

Next steps

There are more tutorials on the Tesseract Project web site (<http://www.typo3-tesseract.com/>). The first tutorial is equivalent to the one above. The second tutorial builds on it and introduces the use of the third type of components: Data Filters.

On top of tutorials the web site offers a number of tips & tricks highlighting some advanced features.

Frequently Asked Questions

Troubleshooting

Q: When I save my Display Controller, I lose the data provider and the data consumer. What's going wrong?

A: This is generally due to the order in which the extensions are installed: "displaycontroller" must be loaded before any of the data providers, data consumers or data filters. You should normally see some information about allowed record types in the Display Controller input form (see screenshot below). If you don't, there's an installation problem. Try changing the order of the extensions (this can be simply done by editing the localconf.php file).

The screenshot shows the TYPO3 administration interface for the Display Controller. The top navigation bar includes 'General', 'Data Objects', 'Access', and 'Tags'. The 'Data Objects' tab is active, showing two main sections: 'Data Consumer' and 'Primary Data Provider'.

Data Consumer: Contains a dropdown menu with 'BE Users list' selected. Below it is a 'Template-based Display' button. A blue arrow points from this button to the 'BE Users' provider.

Primary Data Provider: Contains a dropdown menu with 'BE Users' selected. Below it is a 'Data Query' button. A blue arrow points from this button to the 'BE Users list' consumer.

Advanced Data Filter (primary): Contains a dropdown menu with 'Dummy filter' selected. A blue arrow points from this dropdown to the 'Data Query' button.

Type of filter: Includes radio buttons for 'None', 'Detail view', 'List view', and 'Advanced filter (choose Data Filter below)'. The 'Advanced filter' option is selected.

If filter empty: Includes radio buttons for 'Display nothing' and 'Display everything'. The 'Display everything' option is selected.

Known problems

If you have any issues, please refer to the Tesseract Project web site (<http://www.typo3-tesseract.com/>). You may also post your problems to the TYPO3 English mailing list (typo3.english), so that others may benefit from the answers too. For bugs or feature requests, please open an entry in the extension's bug tracker on Forge (<http://forge.typo3.org/projects/extension-displaycontroller/issues>).

To-Do list

The roadmap for the evolution of this extension can be found on Forge: <http://forge.typo3.org/projects/roadmap/extension-displaycontroller>

For feature requests, please open a report on Forge issue tracker: <http://forge.typo3.org/projects/extension-displaycontroller/issues>

Appendix A – Tutorial answers

Here are the steps that were needed to improve the display of the e-mail address:

1. edit the HTML so that the `###FIELD.email###` marker is just next to the `###FIELD.realName###` marker
2. change the type of the field to "Text"
3. enter the following TypoScript configuration:

```
typolink.parameter.field = email
required = 1
noTrimWrap = | (|)|
```

The "mailto:" link is now built using the `typolink` property of the `TEXT` object. We use the fact that the record's data is loaded into the content object to retrieve the e-mail address using the "field" property. We also make the field value required, so that nothing gets displayed when it's empty. The brackets are added using the "noTrimWrap" property in order to also include a blank space before the opening bracket.