

## TCA Reference

Extension Key: doc\_core\_tca

Language: en

Version: 4.6.0

Keywords: forAdmins, forDevelopers, forIntermediates

Copyright 2000-2011, Documentation Team, <documentation@typo3.org>

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](http://www.typo3.org)

### Official documentation

This document is included as part of the official TYPO3 documentation. It has been approved by the TYPO3 Documentation Team following a peer-review process. The reader should expect the information in this document to be accurate - please report discrepancies to the Documentation Team ([documentation@typo3.org](mailto:documentation@typo3.org)). Official documents are kept up-to-date to the best of the Documentation Team's abilities.

### Core Manual

This document is a Core Manual. Core Manuals address the built in functionality of TYPO3 and are designed to provide the reader with in-depth information. Each Core Manual addresses a particular process or function and how it is implemented within the TYPO3 source code. These may include information on available APIs, specific configuration options, etc.

Core Manuals are written as reference manuals. The reader should rely on the Table of Contents to identify what particular section will best address the task at hand.

# Table of Contents

<b>TCA Reference.....</b>	<b>1</b>	
<b>Introduction.....</b>	<b>3</b>	
About this document.....	3	
What's new.....	3	
Credits.....	3	
Feedback.....	3	
<b>What is \$TCA?.....</b>	<b>4</b>	
Structure of the \$TCA array.....	4	
Glossary.....	5	
The [ctrl] section vs. the other sections.....	6	
<b>\$TCA array reference.....</b>	<b>7</b>	
[ctrl] section.....	7	
[interface] section.....	22	
[feInterface] section.....	24	
[columns][field name] section.....	24	
[columns][field name][config] / Common properties.....	28	
[columns][field name][config] / TYPE: "input".....	28	
[columns][field name][config] / TYPE: "text".....	34	
[columns][field name][config] / TYPE: "check".....	35	
[columns][field name][config] / TYPE: "radio".....	37	
[columns][field name][config] / TYPE: "select".....	37	
[columns][field name][config] / TYPE: "group".....	55	
		[columns][field name][config] / TYPE: "none".....64
		[columns][field name][config] / TYPE: "passthrough"..... 64
		[columns][field name][config] / TYPE: "user".....64
		[columns][field name][config] / TYPE: "flex".....66
		[columns][field name][config] / TYPE: "inline".....79
		[types][key] section.....85
		[palettes][key] section..... 89
		<b>Additional \$TCA features.....91</b>
		Special Configuration introduction..... 91
		Special Configuration options.....92
		Soft References..... 95
		Wizards Configuration.....96
		Wizard scripts in the core..... 105
		<b>Extending the \$TCA array..... 113</b>
		Storing the changes.....113
		Customization examples..... 113
		Verifying the \$TCA.....115
		<b>Appendix A – Skinning considerations.....117</b>
		Visual style of TCEforms..... 117
		<b>Appendix B - Performance considerations.....123</b>
		Loading the full \$TCA dynamically.....123
		Benchmarks on dynamic tables:.....123

# Introduction

## About this document

This document aims to describe the global array called `$TCA`. This array describes the database tables that TYPO3 can operate on. It is a very central element of the TYPO3 architecture.

All code examples used in this manual come either from the TYPO3 source code itself or from the extension "examples", which can be downloaded from the TER.

This document used to be a chapter inside "TYPO3 Core APIs".

## What's new

This document has been updated with changes and features introduced in TYPO3 4.4 and 4.5.

The list of new features for TYPO3 4.6 can be found at [http://wiki.typo3.org/Documentation\\_changes\\_in\\_4.6#TCA\\_reference\\_.28doc\\_core\\_tca.29](http://wiki.typo3.org/Documentation_changes_in_4.6#TCA_reference_.28doc_core_tca.29)

## Credits

The original reference to the TCA was written by Kasper Skårhøj. This version has been updated by François Suter.

## Feedback

For general questions about the documentation get in touch by writing to [documentation@typo3.org](mailto:documentation@typo3.org).

If you find a bug in this manual, please file an issue in this manual's bug tracker:

[http://forge.typo3.org/projects/typo3v4-doc\\_core\\_tca/issues](http://forge.typo3.org/projects/typo3v4-doc_core_tca/issues)

Maintaining quality documentation is hard work and the Documentation Team is always looking for volunteers. If you feel like helping please join the documentation mailing list ([typo3.projects.documentation@lists.typo3.org](mailto:typo3.projects.documentation@lists.typo3.org)).

# What is \$TCA?

The Table Configuration Array (or \$TCA) is a global array in TYPO3 which extends the definition of tables beyond what can be done strictly with SQL. First and foremost \$TCA defines which tables are editable in the TYPO3 backend. Database tables with no corresponding entry in \$TCA are "invisible" to the TYPO3 backend. The \$TCA definition of a table also covers the following:

- the relations between that table and other tables
- what fields should be displayed in the backend and with which layout
- how should a field be validated (e.g. required, integer, etc.)

This array is highly extendable using extensions. As a matter of fact – if you consider an absolutely bare bones installation of TYPO3 (that would without even the required extensions) – only a few tables are configured by default in TYPO3. They are to be found in the file `t3lib/stddb/tables.php` and are:

- the "pages" table containing the page tree of TYPO3
- the "be\_users" table containing backend users
- the "be\_groups" table containing backend user groups
- the "sys\_filemounts" table containing file mounts for backend users
- the "sys\_language" table containing the languages in which various elements can be translated
- the "sys\_news" table (since version 4.5) which is used to display information in the backend login screen.

All other tables are configured in extensions.

The file `"t3lib/stddb/tables.php"` contains not only the \$TCA definition. You can also find other global core variables defined there, but they are not discussed in this document. Some of them are explained in the "Core APIs" manual, those related to skinning in the "Core Skinning Guidelines" manual.

## Structure of the \$TCA array

### The table entries (first level)

The "first level" of the \$TCA array is made of the table names (as they appear in the database):

```
$TCA['pages'] = array(
    ...
);
$TCA['tt_content'] = array(
    ...
);
$TCA['tx_examples_haiku'] = array(
    ...
);
```

Here three tables, "pages", "tt\_content" and "tt\_myext" is shown as examples.

### Inside the table entries (second level)

Each table is further defined by an array which configures how the system handles the table, both for display and processing in the backend. The various parts on this second level are called "sections".

The general structure (looking at a single table) is as follows:

```
$TCA['tx_examples_haiku'] = array(
    'ctrl' => array(
        ...
    ),
    'interface' => array(
        ...
    ),
    'feInterface' => array(
```

```

        ....
    ),
    'columns' => array(
        ....
    ),
    'types' => array(
        ....
    ),
    'palettes' => array(
        ....
    ),
);

```

The following table provides a brief description of each the various sections of \$TCA. Each table is covered in more details in its own chapter.

Section	Description
<b>ctrl</b>	<b>The table</b> The "ctrl" section contains properties for the table in general. These are basically divided in two main categories: <ul style="list-style-type: none"> <li>properties which affect how the table is <i>displayed</i> and handled in the backend <i>interface</i> .                          This includes which icon, what name, which columns contains the title value, which column defines the type value etc.</li> <li>properties which determines how it is processed by the system (TCE).                          This includes publishing control, "deleted" flag, if the table can only be edited by admin-users, may only exist in the tree root etc.</li> </ul>
<b>interface</b>	<b>The backend interface handling</b> The "interface" section contains properties related to the tables display in the backend, mostly the Web > List module.
<b>feInterface</b>	<b>Frontend Editing</b> The "feInterface" section contains properties related to Front End editing of the table, mostly related to the feAdmin_lib. Is deprecated in the sense that it will still exist, but will not be (and should not be) extended further.
<b>columns</b>	<b>The individual fields</b> The "columns" section contains configuration for each table <i>field</i> (also called "column") which can be edited by the backend. The configuration includes both properties for the display in the backend as well as the processing of the submitted data. Each field can be configured as a certain "type" (e.g. checkbox, selector, input field, text area, file or db-relation field, user defined etc.) and for each type a separate set of additional properties applies. These properties are clearly explained for each type.
<b>types</b>	<b>The form layout for editing</b> The "types" section defines how the fields in the table (configured in the "columns" section) should be arranged inside the editing form; in which order, with which "palettes" (see below) and with which possible additional features applied.
<b>palettes</b>	<b>The palette fields order</b> A palette is just a list of fields which will be arranged horizontally side-by-side. But the main idea is that these fields can be displayed in the top-frame of the backend interface on request so they don't display inside the main form. In this way they are kind of hidden fields which are brought forth either by clicking an icon in the main form or (more usually) when you place the cursor in a form field of the main form).

## Deeper levels

All properties on the second level either have their own properties or contain a further hierarchy.

In the case of the [columns] section, this will be the fields themselves. For the [types] and [palettes] section this will be the list of all possible types and palettes.

## Glossary

Before you read on, let's just refresh the meaning of a few concepts mentioned on the next pages:

- **TCE**: Short for "TYPO3 Core Engine". Also referred to as "TCEmain". This class (class.t3lib\_tcemain) should ideally handle all updates to records made in the backend of

TYPO3. The class will handle all the rules which may be applied to each table correctly. It will also handle logging, versioning, history/undo features, copying/moving/deleting etc.

- **"list of"**: Typically used like "list of field names". Whenever "list of" is used it means *a list of strings separated by comma and with NO space between the values*.
- **field name**: The name of a field from a database table. Another word for the same is "column" but it is used more rarely, however the meaning is exactly the same.
- **record type**: A record can have different types, expressed by the value of a certain field in the record. This field is defined by the [ctrl][type] value and it affects also which fields ("types"-configuration) is used to display possible form fields.
- **LLL reference**: is a localized string fetched from a locallang file by prefixing the string with "LLL:".

## The [ctrl] section vs. the other sections

In almost the whole system the [ctrl] section of the \$TCA entry for a table plays a crucial role. For *all* tables configured in \$TCA this section *must* exist in \$TCA. The other sections (except [feInterface]) can optionally be stored in another file.

This feature allows scalability since hundreds of tables can be configured with their complete [ctrl] sections while leaving a relatively small memory footprint since they don't define all the other sections by default. Please see the [ctrl]-property dynamicConfigFile and the section "Loading the full \$TCA dynamically" (in Appendix B) for more details.

This has the following implications:

- You can always depend on accessing information in the [ctrl] section, e.g.  
\$TCA['your\_table\_name']['ctrl']
- But *before* you can depend on information in any other section (except [feInterface]) you should:
  1. Call `t3lib_div::loadTCA('your_table_name');` (This will dynamically load the full content of the \$TCA array for the table)
  2. Then access the information, e.g. `$TCA['your_table_name']['columns']['your_field_name']`

# \$TCA array reference

## ['ctrl'] section

The [ctrl] section contains properties for the table in general.

These properties are basically divided into two main categories:

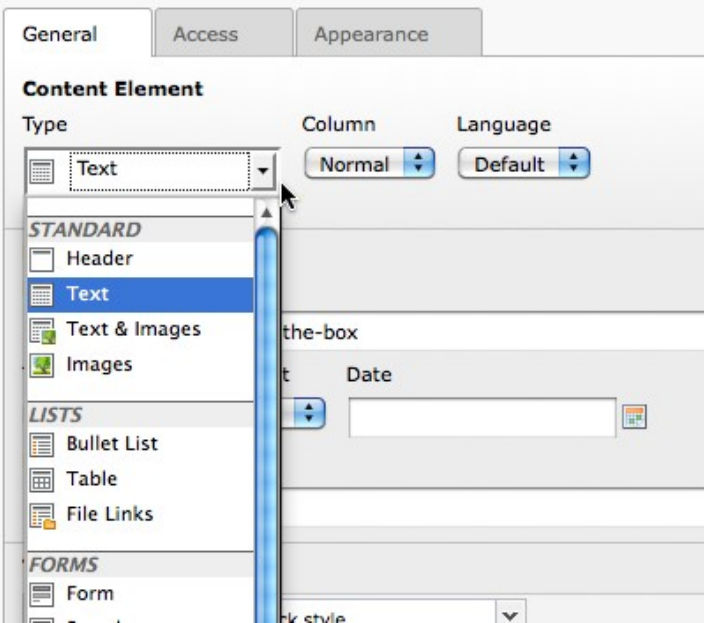
- properties which affect how the table is *displayed* and handled in the backend *interface* .  
This includes which icon, what name, which columns contains the title value, which column defines the type value etc.
- properties which determines how it is processed by the system (TCE).  
This includes publishing control, "deleted" flag, if the table can only be edited by admin-users, may only exist in the tree root etc.

### Reference for the ['ctrl'] section:

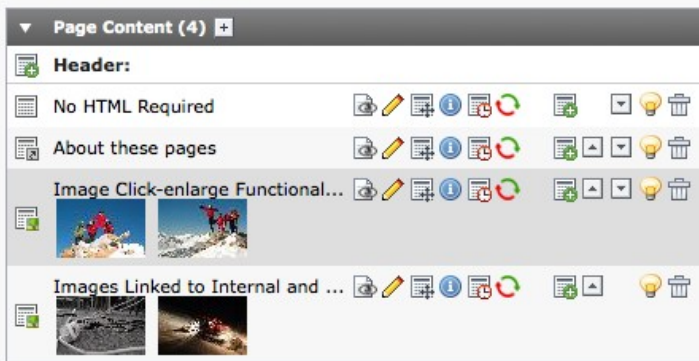
Key	Datatype	Description	Scope
<b>title</b>	string or LLL reference	<p>Contains the <i>system name</i> of the table. Is used for display in the backend.</p> <p>For instance the "tt_content" table is of course named "tt_content" technically. However in the backend display it will be shown as "Pagecontent" when the backend language is English. When another language is chosen, like Danish, then the label "Sideindhold" is shown instead. This value is managed by the "title" value.</p> <p>You can insert plain text values, but the preferred way is to enter a reference to a localized string. See the example below. Refer to the localization section in "Inside TYPO3" for more details.</p> <p><b>Example:</b></p> <pre>\$TCA['sys_template'] = array(     'ctrl' =&gt; array(         'title' =&gt;             'LLL:EXT:cms/locallang_tca.xml:sys_template',</pre> <p>In the above example the "LLL:" prefix tells the system to look up a label from a localized file. The next prefix "EXT:cms" will look for the data in the extension with the key "cms". In that extension the file "locallang_tca.xml" contains a XML structure inside of which one label tag has an index attribute named "sys_template". This tag contains the value to display in the default language. Other languages are provided by the language packs.</p>	Display
<b>label</b>	string (field name)	<p><b>Required!</b></p> <p>Points to the field name of the table which should be used as the "title" when the record is displayed in the system.</p> <p><b>Note:</b> label_userFunc overrides this property (but it is still required).</p>	Display
<b>label_alt</b>	String (comma-separated list of field names)	<p>Comma-separated list of field names, which are holding alternative values to the value from the field pointed to by "label" (see above) if that value is empty. May not be used consistently in the system, but should apply in most cases.</p> <p><b>Example:</b></p> <pre>\$TCA['tt_content'] = array(     'ctrl' =&gt; array(         'label' =&gt; 'header',         'label_alt' =&gt; 'subheader,bodytext',</pre> <p>See t3lib_BEfunc::getRecordTitle() Also see "label_alt_force"</p> <p><b>Note:</b> label_userFunc overrides this property.</p>	Display
<b>label_alt_force</b>	boolean	If set, then the "label_alt" fields are always shown in the title separated by	Display

Key	Datatype	Description	Scope
		<p>comma. See <code>t3lib_BEfunc::getRecordTitle()</code></p> <p><b>Note:</b> <code>label_userFunc</code> overrides this property.</p>	
<b>label_userFunc</b>	string	<p>Function or method reference. This can be used whenever the <code>label</code> or <code>label_alt</code> options don't offer enough flexibility, e.g. when you want to look up another table to create your label. The result of this function overrules the <code>"label"</code>, <code>"label_alt"</code> or <code>"label_alt_force"</code> settings.</p> <p>When calling a method from a class, enter <code>"[classname]-&gt;[methodname]"</code>. The class name must be prefixed <code>"user_"</code> or <code>"tx_"</code>. When using a function, just enter the function name. The function name must be prefixed <code>"user_"</code> or <code>"tx_"</code>. The preferred way is to use a class and a method.</p> <p>Two arguments will be passed to the function/method: The first argument is an array which contains the following information about the record for which to get the title:</p> <pre>\$params['table'] = \$table; \$params['row'] = \$row;</pre> <p>The resulting title must be written to <code>\$params['title']</code> which is passed by reference.</p> <p>The second argument is a reference to the parent object.</p> <p><b>Note:</b> The function file must be included manually (e.g. include it in your <code>ext_tables.php</code> file). When using a class, the preferred way is to declare it with the autoloader.</p> <p><b>Example:</b></p> <p>Let's look at what is done for the <code>"haiku"</code> table of the <code>"examples"</code> extension. First, in the <code>ext_autoload.php</code> file:</p> <pre>\$extensionPath = t3lib_extMgm::extPath('examples'); return array(     'tx_examples_tca' =&gt; \$extensionPath .     'class.tx_examples_tca.php', );</pre> <p>the necessary class is declared. The call to the user function appears in the <code>ext_tables.php</code> file:</p> <pre>\$TCA['tx_examples_haiku'] = array(     'ctrl' =&gt; array(         ...         'label' =&gt; 'title',         'label_userFunc' =&gt; 'tx_examples_tca         -&gt;haikuTitle',         ...     ) );</pre> <p>Finally in <code>class.tx_examples_tca.php</code> is the code itself:</p> <pre>public function haikuTitle(&amp;\$parameters,     \$parentObject) {     \$record =     t3lib_BEfunc::getRecord(\$parameters['table'],     \$parameters['row']['uid']);     \$newTitle = \$record['title'];     \$newTitle .= ' (' .     substr(strip_tags(\$record['poem']), 0, 10) . '...)' ;     \$parameters['title'] = \$newTitle; }</pre>	Display
<b>type</b>	string (field name)	<p>Field name, which defines the <code>"record type"</code>. The value of this field determines which one of the <code>'types'</code> configurations</p>	Display / Proc.



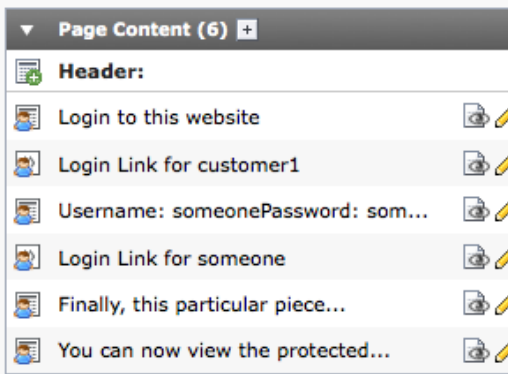
Key	Datatype	Description	Scope
		<p>are used for displaying the fields in the TCEforms. It will probably also affect how the record is used in the context where it belongs.</p> <p>The most widely known usage of this feature is the Content Elements where the "Type:" selector is defined as the "type" field and when you change that selector you will also get another rendering of the form:</p>  <p>It is also used by the "doktype" field in the "pages" table.</p> <p><b>Example:</b> The "dummy" table from the "examples" extension defines different types. The field used for differentiating the types is the "record_type" field. Hence we have the following in the [ctrl] section of the tx_examples_dummy table:</p> <pre>'type' =&gt; 'record_type'</pre> <p>The "record_type" field can takes values ranging from 0 to 2. Accordingly we define types for the same values. Each type defines which fields will be displayed in the BE form. Types are discussed in more details later on.</p> <pre>'types' =&gt; array(     '0' =&gt; array('showitem' =&gt; 'hidden, record_type, title, some_date '),     '1' =&gt; array('showitem' =&gt; 'record_type, title '),     '2' =&gt; array('showitem' =&gt; 'title, some_date, hidden, record_type '), ),</pre>	
<b>hideTable</b>	boolean	Hide this table in record listings.	
<b>requestUpdate</b>	string (list of field names)	This is a list of fields that will trigger an update of the form, on top of the "type" field. This is generally done to hide or show yet more fields depending on the value of the field that triggered the update.	Proc.
<b>iconfile</b>	string	<p>Pointing to the icon file to use for the table. Icons should be dimensioned 16x16 pixels and of the GIF or PNG file type.</p> <p>The value of the option can be any of these:</p> <ul style="list-style-type: none"> <li><b>If there is a slash (/) in the value:</b> It must be a relative file path pointing to the icon file relative to the typo3/ (admin) folder. You may start that path with './' if you like to get your icon from a folder in the PATH_site path.</li> <li>For extensions, see example below.</li> <li><b>If there is just a filename:</b> It must exist in the "typo3/gfx/i/" folder.</li> </ul>	Display

Key	Datatype	Description	Scope
		<ul style="list-style-type: none"> <li><b>If empty/not given:</b> The default icon for a table is defined as "gfx/i/[table_name].gif". (This is an obsolete approach to use since the content of the "gfx/i/" folder should not be changed.)</li> </ul> <p><b>Example: How to assign an icon from an extension</b> For haikus from the "examples" extension, the icon is defined this way:</p> <pre>'iconfile' =&gt; t3lib_extMgm::extRelPath(\$_EXTKEY) . 'icon_tx_examples_haiku.gif',</pre>	
<b>typeicon_column</b>	string (field name)	<p>Field name, whose value decides <i>alternative icons</i> for the table (The default icon is the one defined with the 'iconfile' value.) An icon in the 'typeicons' array may override the default icon if an entry is found for the key having the value of the field pointed to by "typeicon_column" (this feature). <b>Notice:</b> These options ("typeicon_column" and "typeicons") do not work for the pages-table, which is configured by the \$PAGES_TYPES array. Related "typeicons"</p> <p>This feature is used by for instance the "tt_content" table (Content Elements) where each type of content element has its own icon.</p> <p><b>Example:</b> See "typeicons"</p>	Display
<b>typeicons</b>	array	<p>(See "typeicon_column")</p> <p><b>Example of configuration (from the "tt_content" table):</b></p> <pre>'typeicon_column' =&gt; 'CType', 'typeicons' =&gt; array(     'header' =&gt; 'tt_content_header.gif',     'textpic' =&gt; 'tt_content_textpic.gif',     'image' =&gt; 'tt_content_image.gif',     'bullets' =&gt; 'tt_content_bullets.gif',     'table' =&gt; 'tt_content_table.gif',     'splash' =&gt; 'tt_content_news.gif',     'uploads' =&gt; 'tt_content_uploads.gif',     'multimedia' =&gt; 'tt_content_mm.gif',     'menu' =&gt; 'tt_content_menu.gif',     'list' =&gt; 'tt_content_list.gif',     'mailform' =&gt; 'tt_content_form.gif',     'search' =&gt; 'tt_content_search.gif',     'login' =&gt; 'tt_content_login.gif',     'shortcut' =&gt; 'tt_content_shortcut.gif',     'script' =&gt; 'tt_content_script.gif',     'div' =&gt; 'tt_content_div.gif',     'html' =&gt; 'tt_content_html.gif' ),</pre>	Display
<b>thumbnail</b>	string (field name)	<p>Field name, which contains the value for any thumbnails of the records. This could be a field of the "group" type containing a list of file names.</p> <p><b>Example:</b> For the "tt_content" table this option points to the field "image" which contains the list of images that can be attached to the content element:</p> <pre>'thumbnail' =&gt; 'image',</pre>	Display

Key	Datatype	Description	Scope
		<p>The effect of the field can be see in listings in e.g. the "List" module:</p>  <p>(You might have to enable "Show Thumbnails by default" in the "Startup" tab of the User Settings module first in order to see this display).</p>	
<b>selicon_field</b>	string (field name)	<p>Field name, which contains the thumbnail image used to represent the record visually whenever it is shown in TCEforms as a foreign reference selectable from a selector box.</p> <p>Only images in a usual format for the web (i.e. gif, png, jpeg, jpg) are allowed. No scaling is done.</p> <p>You should consider this a feature where you can attach an "icon" to a record which is typically selected as a reference in other records. For example a "category". In such a case this field points out the icon image which will then be shown. This feature can thus enrich the visual experience of selecting the relation in other forms.</p> <p><b>Example:</b> The "backend_layout" table defines the "icon" field as being the one containing reference icons:</p> <pre>\$TCA['backend_layout'] = array (     'ctrl' =&gt; array (         ...         'selicon_field' =&gt; 'icon',         'selicon_field_path' =&gt; 'uploads/media',         ...     ) );</pre> <p>Also see "selicon_field_path" below.</p>	Display
<b>selicon_field_path</b>	string	<p>The path prefix of the value from 'selicon_field'. This must be the same as the "upload_path" of that field.</p> <p>See example above.</p>	Display
<b>sortby</b>	string (field name)	<p>Field name, which is used to manage the <i>order</i> of the records.</p> <p>The field will contain an integer value which positions it at the correct position between other records from the same table on the current page.</p> <p><b>NOTICE:</b> The field should <i>not</i> be editable by the user since the TCE will manage the content automatically in order to manage the order of records.</p> <p>This feature is used by e.g. the "pages" table and "tt_content" table (Content Elements) in order to output the pages or the content elements in the order expected by the editors. Extensions are expected to respect this field.</p> <p>Typically the field name "sorting" is dedicated to this feature.</p> <p>Also see "default_sortby" below.</p>	Display /Proc.
<b>default_sortby</b>	string	<p>If a field name for "sortby" is defined, then this is ignored.</p> <p>Otherwise this is used as the 'ORDER BY' statement to sort the records in</p>	Display

Key	Datatype	Description	Scope
		<p>the table when listed in the TYPO3 backend.</p> <p><b>Example:</b> For the "haikus" table of the "examples" extension, records are listed alphabetically, based on their title:</p> <pre>\$TCA['tx_examples_haiku'] = array(     'ctrl' =&gt; array(         ...         'default_sortby' =&gt; 'ORDER BY title',         ...     ) );</pre>	
<b>mainpalette</b>	comma-separated list of integers (pointing to multiple palette keys)	<p>Points to the palette-number(s) that should always be shown in the bottom of the TCEform.</p> <p><b>Example:</b> The [ctrl] section looks like this:</p> <pre>'mainpalette' =&gt; '1',</pre> <p>The number "1" references a palette. This palette could be something like:</p> <pre>'palettes' =&gt; array(     '1' =&gt; array('showitem' =&gt;         'hidden,starttime,endtime,fe_group'),</pre> <p>Note that "mainpalette" is not much used anymore. It has the drawback of positioning the related fields weirdly when tabs are added to existing tables via extensions (the fields come at the end of the new tabs, which may be disturbing for editors).</p>	Display
<b>canNotCollapse</b>	boolean	<p>By default, fields placed in palettes (see later for more about palettes) are not shown by TCEforms. They appear only once the "Show secondary options" checkbox at the bottom of the screen is checked.</p> <div> <input type="checkbox"/> Show secondary options (palettes)         </div> <p>By setting "canNotCollapse" to true, the palettes of this table will always be displayed, as if the above-mentioned option was always checked. This setting can also be defined per palette (see later).</p>	Display
<b>tstamp</b>	string (field name)	<p>Field name, which is automatically updated to the current timestamp (UNIX-time in seconds) each time the record is updated/saved in the system. Typically the name "tstamp" is used for that field.</p> <p><b>Example:</b> from the [ctrl] section of the "haikus" table:</p> <pre>\$TCA['tx_examples_haiku'] = array(     ...     'tstamp' =&gt; 'tstamp',     'crdate' =&gt; 'crdate',     'cruser_id' =&gt; 'cruser_id',     ... );</pre> <p>The above example shows the same definition for the "crdate" and "cruser_id" fields described below.</p>	Proc.
<b>crdate</b>	string (field name)	<p>Field name, which is automatically set to the current timestamp when the record is created. Is never modified again. Typically the name "crdate" is used for that field. See example above.</p>	Proc.
<b>cruser_id</b>	string (field name)	<p>Field name, which is automatically set to the uid of the backend user (be_users) who originally created the record. Is never modified again. Typically the name "cruser_id" is used for that field. See example above.</p>	Proc.

Key	Datatype	Description	Scope
<b>rootLevel</b>	[0, 1, -1]	<p>Determines where a record may exist in the page tree. There are three options depending on the value:</p> <ul style="list-style-type: none"> <li>• <b>0 (false): Can only exist in the page tree.</b> Records from this table <i>must</i> belong to a page (i.e. have a positive "pid" field value). Thus records cannot be created in the root of the page tree (where "admin" users are the only ones allowed to create records anyways). This is the default behavior.</li> <li>• <b>1 (true): Can only exist in the root.</b> Records must have a "pid"-field value equal to zero. The consequence is that only admin can edit this record.</li> <li>• <b>-1: Can exist in both page tree and root.</b> Records can belong either to a page (positive "pid" field value) or exist in the root of the page tree (where the "pid" field value will be 0 (zero)) <b>Notice:</b> the -1 value will still select foreign_table records for selector boxes only from root (pid=0)</li> </ul> <p><b>Notice:</b> The setting for "rootLevel" is ignored for records in the "pages" table (they are hardcoded to be allowed anywhere, equal to a "-1" setting of rootLevel).</p> <p><b>Warning:</b> this property does not tell the whole story. If set to "0" or "-1", it allows records from the table in the page tree, but <b>not</b> on any kind of page. By default records can be created only in "Folder"-type pages. To enable the creation of records on any kind of page, an additional call must be made:</p> <pre>t3lib_extMgm::allowTableOnStandardPages('tx_examples_haiku');</pre>	Proc. / Display
<b>readOnly</b>	boolean	Records from this table may not be edited in the TYPO3 backend. Such tables are usually called "static".	Proc. / Display
<b>adminOnly</b>	boolean	<p>Records may be changed <i>only</i> by "admin"-users (having the "admin" flag set).</p> <p><b>Example:</b> The "cms" system extension defines the table "sys_template" as being editable only by admin users:</p> <pre>\$TCA['sys_template'] = array (     'ctrl' =&gt; array (         ...         'adminOnly' =&gt; 1,         ...     ) );</pre>	Proc. / Display
<b>editlock</b>	string (field name)	<p>Field name, which – if set – will prevent all editing of the record for non-admin users.</p> <p>The field should be configured as a checkbox type. Non-admins could be allowed to edit the checkbox but if they set it, they will effectively lock the record so they cannot edit it again – and they need an Admin-user to remove the lock.</p> <p>Note that this flag is cleared when a new copy or version of the record is created.</p> <p>This feature is used on the pages table, where it also prevents editing of records on that page (except other pages)! Also, no new records (including pages) can be created on the page.</p>	Proc / Display
<b>origUid</b>	string (field name)	Field name, which will contain the UID of the original record in case a record is created as a copy or new version of another record. Is used when new versions are created from elements and enables the backend to display a visual comparison between a new version and its original.	Proc

Key	Datatype	Description	Scope
<b>delete</b>	string (field name)	<p>Field name, which indicates if a record is considered deleted or not. If this feature is used, then records are not really deleted, but just marked 'deleted' by setting the value of the field name to "1". And in turn the whole system <i>must</i> strictly respect the record as deleted. This means that any SQL query must exclude records where this field is true.</p> <p>This is a very common feature. Most tables use it throughout the TYPO3 Core.</p>	Proc. / Display
<b>enablecolumns</b>	array	<p>Specifies which <i>publishing control features</i> are automatically implemented for the table.</p> <p>This includes that records can be "disabled" or "hidden", have a starting and/or ending time and be access controlled so only a certain front end user group can access them</p> <p>In the frontend libraries the enableFields() function automatically detects which of these fields are configured for a table and returns the proper WHERE clause SQL code for creating select queries.</p> <p>There are the keys in the array you can use. Each of the values must be a field name in the table which should be used for the feature:</p> <p><b>"disabled"</b>: defining hidden-field of record  <b>"starttime"</b>: defining start time-field of record  <b>"endtime"</b>: defining end time-field of record  <b>"fe_group"</b>: defining fe_group-field of record</p> <p><b>Notice:</b> In general these fields do <i>not</i> affect the access or display in the backend! They are primarily related to the frontend. However the icon of records having these features enabled will normally change as these examples show:</p>  <p>See also the "delete" feature which is related, but is active for both frontend and backend.</p> <p><b>Example:</b>  Typically the "enablecolumns" could be configured like this (here for the "tt_content" table):</p> <pre>'enablecolumns' =&gt; array(     'disabled' =&gt; 'hidden',     'starttime' =&gt; 'starttime',     'endtime' =&gt; 'endtime',     'fe_group' =&gt; 'fe_group', ),</pre>	Proc. / Display
<b>searchFields</b>	string	<p>Comma-separated list of fields from the table that will be included when searching for records in the TYPO3 backend. Starting with TYPO3 4.6, no record from a table will ever be found if that table does not have "searchFields" defined.</p> <p>There are finer controls per column, see the "search" property in the list of "Common properties" further in this manual.</p>	Search

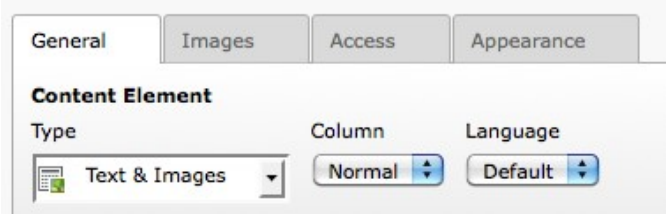
Key	Datatype	Description	Scope
		<p><b>Example:</b> The "tt_content" table has the following definition:</p> <pre> \$TCA['pages'] = array(     'ctrl' =&gt; array(         ...         'searchFields' =&gt;         'title,alias,nav_title,subtitle,url,keywords,description,abstract,author,author_email',         ...     ), ); </pre>	
<b>groupName</b>	string	This option can be used to group records in the new record wizard. If you define a new table and set its "groupName" to the key of another extension, your table will appear in the list of records from that other extension in the new record wizard.	Special
<b>hideAtCopy</b>	boolean	If set, and the "disabled" field from "enablecolumns" (see above) is specified, then records will be disabled/hidden when they are copied.	Proc.
<b>prependAtCopy</b>	string or LLL reference	This string will be prepended the records title field when the record is inserted on the same PID as the original record (thus you can distinguish them). Usually the value is something like " (copy %s)" which tells that it was a copy that was just inserted (The token "%s" will take the copy number).	Proc.
<b>copyAfterDuplFields</b>	string (list of field names)	<p>The fields in this list will automatically have the value of the same field from the 'previous' record transferred when they are <i>copied or moved</i> to the position <i>after</i> another record from same table.</p> <p><b>Example:</b> 'copyAfterDuplFields' =&gt; 'colPos, sys_language_uid',</p>	Proc.
<b>setToDefaultOnCopy</b>	string (list of field names)	<p>These fields are restored to the default value of the record when they are copied.</p> <p><b>Example:</b> \$TCA['sys_action'] = array(     'ctrl' =&gt; array(         'setToDefaultOnCopy' =&gt; 'assign_to_groups',</p>	Proc.
<b>useColumnsForDefaultValues</b>	string (list of field names)	<p>When a new record is created, this defines the fields from the 'previous' record that should be used as default values.</p> <p><b>Example:</b> \$TCA['sys_filemounts'] = array(     'ctrl' =&gt; array(         'useColumnsForDefaultValues' =&gt; 'path,base',</p>	Proc.
<b>shadowColumnsForNewPlaceholders</b>	string (list of field names)	<p>When a new element is created in a draft workspace a placeholder element is created in the Live workspace. Some values must be stored in this placeholder and not just in the overlay record. A typical example would be "sys_language_uid". This property defines the list of fields whose values are "shadowed" to the Live record.</p> <p>All fields listed for this option must be defined in \$TCA[&lt;table&gt;] ['columns'] as well. Furthermore fields which are listed in "transOrigPointerField", "languageField", "label" and "type" are automatically added to this list of fields and do not have to mentioned again here.</p> <p><b>Example:</b> \$TCA['tt_content'] = array(     'ctrl' =&gt; array(         'shadowColumnsForNewPlaceholders' =&gt;         'sys_language_uid,l18n_parent,colPos',</p>	Proc.
<b>is_static</b>	boolean	This marks a table to be "static". A "static table" means that it should not be updated for individual	Used by

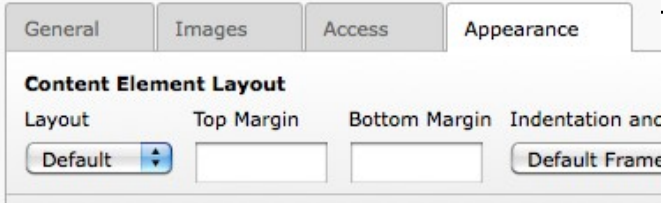
Key	Datatype	Description	Scope
		<p>databases because it is meant to be centrally updated and distributed. For instance static tables could contain country-codes used in many systems.</p> <p>The foremost property of a static table is that the uid's used are the SAME across systems. Import/Export of records expect static records to be common for two systems.</p> <p><b>Example (also including the features "rootLevel", "readOnly" and "adminOnly" above):</b></p> <pre>\$TCA['static_template'] = array(     'ctrl' =&gt; array(         'label' =&gt; 'title',         'tstamp' =&gt; 'tstamp',         'title' =&gt;         'LLL:EXT:statictemplates/locallang_tca.xml:static_template',         'readOnly' =&gt; 1, // Prevents the table from         being altered         'adminOnly' =&gt; 1, // Only admin, if any         'rootLevel' =&gt; 1,         'is_static' =&gt; 1,</pre>	import/ export
<b>fe_cruser_id</b>	string (field name)	Field name which is used to store the uid of a frontend user if the record is created through fe_adminLib	FE
<b>fe_crgroup_id</b>	string (field name)	Field name which is used for storing the uid of a frontend group whose members are allowed to edit through fe_adminLib .	FE
<b>fe_admin_lock</b>	string (field name)	Field name which points to the field name which - as a boolean - will prevent any editing by the fe_adminLib if set. Say if the "fe_cruser_id" field matches the current fe_user normally the field is editable. But with this option, you could make a check-box in the backend that would lock this option.	FE
<b>languageField</b>	string (field name)	<p><b>Localization access control.</b></p> <p>Field name which contains the pointer to the language of the record's content. Language for a record is defined by an integer pointing to a "sys_language" record (found in the page tree root). Backend users can be limited to have edit access for only certain of these languages and if this option is set, edit access for languages will be enforced for this table.</p> <p>The values in this field may be the following:</p> <p><b>-1</b> : (ALL) The record does not represent any specific language. Localization access control is never carried out for such a record. Typically this is used if the record has content which itself handles localization (such as plugins or flexforms).</p> <p><b>0</b> : The default language of the system. Localization access control applies.</p> <p><b>Values &gt; 0</b> : Points to a uid of a sys_language record representing a possible language for translation. Localization access control applies.</p> <p>The field name pointed to should be a single value selector box (maxitems &lt;=1) saving its value into an integer field in the database.</p>	Proc / Display
<b>transOrigPointerField</b>	string (field name)	<p>Name of the field used by translations to point back to the original record (i.e. the record in the default language of which they are a translation). If this value is found being set together with "languageField" then TCEforms will show the default translation value under the fields in the main form. This is very neat if translators are to see what they are translating of course...</p> <p>Must be configured in \$TCA[&lt;table&gt;]['columns'], at least as a passthrough type.</p>	Proc / Display
<b>transForeignTable</b>	string (table name)	<p>Translations may be stored in a separate table, instead of the same one. In such a case, the name of the translation table is stored in this property. The translation table in turn will use the "transOrigPointerTable" property to point back to this table.</p> <p>This is used in the TYPO3 Core for the "pages" table, which uses the "pages_language_overlay" table to hold the translations.</p>	



Key	Datatype	Description	Scope
		<p><b>Example:</b></p> <pre>\$TCA['pages'] = array(     'ctrl' =&gt; array(         'transForeignTable' =&gt;         'pages_language_overlay',      \$TCA['pages_language_overlay'] = array (         'ctrl' =&gt; array (             'transOrigPointerField' =&gt; 'pid',             'transOrigPointerTable' =&gt; 'pages',</pre> <p>Note that the "transOrigPointerField" is still used, but within the table holding the translations.</p> <p><i>WARNING: This is still not fully for all other tables than the "pages" table. You should expect some issues and inconsistencies when using this translation method.</i></p>	
<b>transOrigPointerTable</b>	string (table name)	Symmetrical property to "transForeignTable". See above for explanations.	Proc / Display
<b>transOrigDiffSourceField</b>	string (field name)	<p>Field name which will be updated with the value of the original language record whenever the translation record is updated. This information is later used to compare the current values of the default record with those stored in this field and if they differ there will be a display in the form of the difference visually. This is a big help for translators so they can quickly grasp the changes that happened to the default language text.</p> <p>The field type in the database should be a large text field (clob/blob). You don't have to configure this field in \$TCA[&lt;table&gt;]['columns'], but if you do, select the "passthrough" type. That will enable that the undo function to also work on this field.</p>	Proc / Display
<b>versioningWS</b>	boolean / version number	<p>If set, versioning is enabled for this table. If integer it indicates a version number of versioning features.</p> <ul style="list-style-type: none"> <li>Version 2: Support for moving elements was added. ("V2" is used to mark features)</li> </ul> <p>Versioning in TYPO3 is based on this scheme:</p> <pre>[Online version, pid&gt;=0] 1- * [Offline versions, pid=-1]</pre> <p>Offline versions are identified by having a pid value = -1 and they refer to their online version by the field "t3ver_oid". Offline versions of the "Page" and "Branch" types (contrary to "Element" type) can have child records which points to the uid of their offline "root" version with their pid fields (as usual). These children records are typically copies of child elements of the online version of the offline root version, but are not considered "versions" of them in a technical sense, hence they don't point to them with their t3ver_oid field (and shouldn't).</p> <p>In the backend "Offline" is labeled "Draft" while "Online" is labeled "Live".</p> <p>In order for versioning to work on a table there are certain requirements; Tables supporting versioning must have these fields:</p> <ul style="list-style-type: none"> <li><b>"t3ver_oid"</b> - For offline versions; pointing back to online version uid. For online: 0 (zero)</li> <li><b>"t3ver_id"</b> - Incremental integer (version number)</li> <li><b>"t3ver_label"</b> - Version label, e.g. "1.1.1" or "Christmas edition"</li> <li><b>"t3ver_wsuid"</b> - For offline versions: Workspace ID of version. For all workspace Ids apart from 0 (zero) there can be only one version of an element per ID. For online: 0 (zero) unless t3ver_state is set in which case it plays a role for previews in the backend (to no de-select placeholders for workspaces, see t3lib_BEfunc::versioningPlaceholderClause()) and for publishing of</li> </ul>	Proc.

Key	Datatype	Description	Scope
		<p>move-to-actions (see t3lib_BEfunc::getMovePlaceholder())</p> <ul style="list-style-type: none"> <li>• <b>“t3ver_state”</b> - Contains special states of a version used when new, deleted, moved content requires versioning. <ul style="list-style-type: none"> <li>• For an <b>online</b> version: <ul style="list-style-type: none"> <li>• “1” or “2” means that it is a temporary placeholder for a new element (which is the offline version of this record)</li> <li>• “3” means it is a “move-to-location” placeholder and t3ver_move_id holds uid of online record (with an offline version) to move . Unlike for “1” and “2” there is <i>no offline version</i> of this record type! (V2 feature)</li> <li>• If “t3ver_state” has a value &gt;0 it should never be shown in Live workspace.</li> </ul> </li> <li>• For an <b>offline</b> version: <ul style="list-style-type: none"> <li>• “1” or “2” means that when published, the element must be deleted (placeholder for delete-action).</li> <li>• “-1” means it is just an indication that the online version has the flag set to “1” (is a placeholder for new records.). This only affects display, not processing anywhere.</li> <li>• “4” means this version is a “move-pointer” for the online record and an online “move-to-location” (t3ver_state=3) record exists. (V2 feature)</li> </ul> </li> </ul> </li> <li>• <b>“t3ver_stage”</b> - Contains the ID of the stage at which the record is. Special values are “0” which still refers to “edit”, “-10” refers to “ready to publish”.</li> <li>• <b>“t3ver_count”</b> - 0/offline=draft/never published, 0/online=current, 1/offline=archive, 1+=multiple online/offline occurrences (incrementation happens when versions are swapped offline.)</li> <li>• <b>“t3ver_tstamp”</b> - Timestamp of last swap/publish action.</li> <li>• <b>“t3ver_move_id”</b> - For online records with t3ver_state=3 this indicates the online record to move to this location upon publishing of the offline version of the online record “t3ver_move_id” points to.</li> <li>• The fields <b>pid</b> and <b>uid</b> should have “signed” attributes in MySQL (so their content can be negative!)</li> </ul> <p><b>Corresponding SQL definitions:</b></p> <pre> t3ver_oid int(11) DEFAULT '0' NOT NULL, t3ver_id int(11) DEFAULT '0' NOT NULL, t3ver_wsid int(11) DEFAULT '0' NOT NULL, t3ver_label varchar(30) DEFAULT '' NOT NULL, t3ver_state tinyint(4) DEFAULT '0' NOT NULL, t3ver_stage int(11) DEFAULT '0' NOT NULL, t3ver_count int(11) DEFAULT '0' NOT NULL, t3ver_tstamp int(11) DEFAULT '0' NOT NULL, t3ver_move_id int(11) DEFAULT '0' NOT NULL, </pre> <p><b>Special “t3ver_swapmode” field for pages</b></p> <p>When pages are versioned it is an option whether content and even the branch of the page is versioned. This is determined by the parameter “treeLevels” set when the page is versioned. “-1” means swap only record, 0 means record and content and &gt;0 means full branch. When the version is later published the swapping will happen accordingly.</p>	
<b>versioningWS_alwaysAllowLiveEdit</b>	boolean	If set, this table can always be edited live even in a workspace and even if “live editing” is not enabled in a custom workspace. For instance this is set by default for Backend user and group records since it is assumed that administrators like the flexibility of editing backend users without having to go to the Live workspace.	
<b>versioning_followPages</b>	boolean	<p>(Only for other tables than “pages”)</p> <p>If set, content from this table will get copied along when a new version of a page is created.</p> <p><b>Tracking Originals</b></p> <p>It is highly recommended to use the “origUid” feature for tables whose records are copied with pages that are versioned with content or subtree since this will enable the possibility of content comparison between</p>	Proc.

Key	Datatype	Description	Scope
		current and future versions.	
<b>dividers2tabs</b>	integer	<p>This key defines the activation of tabs, according to the following values:</p> <p>0: disabled (default)  1: activated, empty tabs are removed  2: activated, empty tabs are disabled</p> <p>When tabs are activated, the special field name "--div--" used in the types configuration will be interpreted as starting a new tab in a tab-menu for the record. The second part after "--div--" is the title of the tab.</p> <p>If you place a "--div--" field as the very first element in the types configuration it will just be used to set the title of the first tab (which is by default "General").</p> <p><b>Example:</b>  The [ctrl] section of table "tt_content" contains the following:</p> <pre>\$TCA['tt_content'] = array (     'ctrl' =&gt; array (         'dividers2tabs' =&gt; 1</pre> <p>Then the types make use of "--div--" fields. Example for the "text"-type (usage of "--div--" highlighted in bold):</p> <pre>'types' =&gt; array(     '1' =&gt; array(         'showitem' =&gt; 'CType',     ),     ...     'text' =&gt; array(         'showitem' =&gt;             '--palette--;LLL:EXT:cms/ locallang_ttc.xml:palette.general;general,             --palette--;LLL:EXT:cms/ locallang_ttc.xml:palette.header;header,             bodytext;LLL:EXT:cms/ locallang_ttc.xml:bodytext_formlabel;;richtext:rte_tra nsform[flag=rte_enabled]mode=ts_css],             rte_enabled;LLL:EXT:cms/ locallang_ttc.xml:rte_enabled_formlabel,             <b>--div--;LLL:EXT:cms/ locallang_ttc.xml:tabs.access,</b>             --palette--;LLL:EXT:cms/ locallang_ttc.xml:palette.visibility;visibility,             --palette--;LLL:EXT:cms/ locallang_ttc.xml:palette.access;access,             <b>--div--;LLL:EXT:cms/ locallang_ttc.xml:tabs.appearance,</b>             --palette--;LLL:EXT:cms/ locallang_ttc.xml:palette.frames;frames,             --palette--;LLL:EXT:cms/ locallang_ttc.xml:palette.textlayout;textlayout,             <b>--div--;LLL:EXT:cms/ locallang_ttc.xml:tabs.extended',</b>     ),</pre> <p>This will render a tab menu for the record where the fields are distributed on the various tabs:</p>  <p>Here another tab is activated and another part of the form is shown:</p>	

Key	Datatype	Description	Scope
		 <p>Since TYPO3 4.3, it is customary for most tables to make use of tabs for improved usability.</p>	
<b>dynamicConfigFile</b>	string	<p>Reference to the complete \$TCA entry content.</p> <p>Filename of the PHP file which contains the <i>full configuration</i> of the table in \$TCA. The [ctrl] part (and [feInterface] if used) are always mandatory, but the rest may be placed in such a file in order to limit the amount of memory consumed by the \$TCA array (when e.g. the columns definitions are not needed).</p> <p>The format of the value may be:</p> <ul style="list-style-type: none"> <li>an absolute path (this is used for extensions, see example below).</li> <li><b>prefixed with "T3LIB:"</b> This indicates that it's located in t3lib/install/</li> <li>any other path is considered to be relative to "typo3conf/"</li> </ul> <p><b>Example:</b> Looking at the definition of the "haikus" table, we find the following in the "ext_tables.php" file:</p> <pre>\$TCA['tx_examples_haiku'] = array(     'ctrl' =&gt; array(         ...         'dynamicConfigFile' =&gt;             t3lib_extMgm::extPath(\$_EXTKEY) . 'tca.php',         ...     ) );</pre> <p>Then in the file "tca.php" is PHP code which completes the \$TCA entry for the table:</p> <pre>&lt;?php \$TCA['tx_examples_haiku'] = array(     'ctrl' =&gt; \$TCA['tx_examples_haiku']['ctrl'],     'columns' =&gt; array(         'hidden' =&gt; array(             'exclude' =&gt; 1,             'label' =&gt;                 'LLL:EXT:lang/locallang_general.xml:LGL.hidden',             'config' =&gt; array(                 'type' =&gt; 'check',                 'default' =&gt; '0'             )         ),         ...     ),     ... );</pre> <p>Note how the [ctrl] section is referenced so as not to be lost.</p> <p>See Appendix B for a detailed discussion of dynamically loading \$TCA.</p>	API
<b>EXT[extension_key]</b>	array	<p>User-defined content for extensions. You can use this as you like. Let's say that you have an extension with the key "myext", then you have the right to define properties for:</p> <pre>...['ctrl']['EXT']['myext'] = ... (whatever you define)</pre> <p>Note that this is just a convention. You can use some other syntax but with the risk that it conflicts with some other extension or future changes in the TYPO3 Core.</p>	Ext.

## Examples

Here are a couple examples of complete configurations of [ctrl] sections.

```
$TCA['pages'] = array(
    'ctrl' => array(
        'label' => 'title',
        'tstamp' => 'tstamp',
        'sortby' => 'sorting',
        'title' => 'LLL:EXT:lang/locallang_tca.xml:pages',
        'type' => 'doktype',
        'versioningWS' => 2,
        'origUid' => 't3_origuid',
        'delete' => 'deleted',
        'crdate' => 'crdate',
        'hideAtCopy' => 1,
        'prependAtCopy' => 'LLL:EXT:lang/locallang_general.xml:LGL.prependAtCopy',
        'cruser_id' => 'cruser_id',
        'editlock' => 'editlock',
        'useColumnsForDefaultValues' => 'doktype,fe_group,hidden',
        'dividers2tabs' => 1,
        'enablecolumns' => array(
            'disabled' => 'hidden',
            'starttime' => 'starttime',
            'endtime' => 'endtime',
            'fe_group' => 'fe_group',
        ),
        'transForeignTable' => 'pages_language_overlay',
        'typeicon_column' => 'doktype',
        'typeicon_classes' => array(
            '1' => 'apps-pagetree-page-default',
            '1-hideinmenu' => 'apps-pagetree-page-not-in-menu',
            ...
            'contains-news' => 'apps-pagetree-folder-contains-news',
            'default' => 'apps-pagetree-page-default',
        ),
        'typeicons' => array(
            '1' => 'pages.gif',
            '254' => 'sysf.gif',
            '255' => 'recycler.gif',
        ),
        'dynamicConfigFile' => 'T3LIB:tbl_pages.php',
    ),
);
```

This is found in file "t3lib/stdtdb/tables.php". Here are a few notes:

- When pages are displayed in the backend, the "label" property indicates that you will see the content from the field named "title" shown as the title of the page record.
- The field called "sorting" will be used to determine the order in which pages are displayed within each branch of the page tree.
- The title for the pages table as shown in the backend (e.g. "Pages" in english, "Sider" in danish etc...) is defined as coming from a "locallang" file.
- The "type" field will be the one named "doktype". This determines the set of fields shown in the edit forms in the backend.
- Note on the last line how the dynamic configuration file is referenced.

Configuration for the tt\_content table (Content Elements) is no less complete. It can be found in file "typo3/sysextd/cms/ext\_tables.php":

```
// *****
// This is the standard TypoScript content table, tt_content
// *****
$TCA['tt_content'] = array (
    'ctrl' => array (
        'label' => 'header',
        'label_alt' => 'subheader,bodytext',
        'sortby' => 'sorting',
        'tstamp' => 'tstamp',
        'crdate' => 'crdate',
```

```

'cruser_id' => 'cruser_id',
'title' => 'LLL:EXT:cms/locallang_tca.xml:tt_content',
'delete' => 'deleted',
'versioningWS' => 2,
'versioning_followPages' => true,
'origUid' => 't3_origuid',
'type' => 'CType',
'hideAtCopy' => true,
'prependAtCopy' => 'LLL:EXT:lang/locallang_general.xml:LGL.prependAtCopy',
'copyAfterDuplFields' => 'colPos,sys_language_uid',
'useColumnsForDefaultValues' => 'colPos,sys_language_uid',
'shadowColumnsForNewPlaceholders' => 'colPos',
'transOrigPointerField' => 'l18n_parent',
'transOrigDiffSourceField' => 'l18n_diffsource',
'languageField' => 'sys_language_uid',
'enablecolumns' => array (
    'disabled' => 'hidden',
    'starttime' => 'starttime',
    'endtime' => 'endtime',
    'fe_group' => 'fe_group',
),
'typeicon_column' => 'CType',
'typeicon_classes' => array(
    'header' => 'mimetypes-x-content-header',
    ...
    'default' => 'mimetypes-x-content-text',
),
'typeicons' => array (
    'header' => 'tt_content_header.gif',
    ...
    'html' => 'tt_content_html.gif'
),
'thumbnail' => 'image',
'requestUpdate' => 'list_type,rte_enabled',
'dynamicConfigFile' => t3lib_extMgm::extPath($EXTKEY). 'tbl_tt_content.php',
'dividers2tabs' => 1
);

```

- of particular note is the "enablecolumns" property. It is quite extensive for this table since it is a frontend-related table. Thus proper access rights, publications dates, etc. must be enforced.
- every type of content element has its own icon and its own class, used in conjunction with the skinning API to visually represent that type in the TYPO3 backend.
- the column "image" is defined as the one to use to fetch any thumbnails related to the record.

## ['interface'] section

Contains configuration for display and listing in various parts of the core backend:

Key	Datatype	Description
<b>showRecordFieldList</b>	string (list of field names)	Defines which fields are shown in the show-item dialog. E.g. 'doktype,title,alias,hidden,...'
<b>always_description</b>	boolean	<p>If set, the description/helpicons are always shown regardless of the configuration of the user. Works only in TCEforms and for tables loaded via <code>t3lib_BEfunc::loadSingleTableDescription()</code></p> <p><b>Start:</b>  <input type="checkbox"/> <input type="text"/></p> <p><b>Stop:</b>  <input type="checkbox"/> <input type="text"/></p>
<b>maxDBListItems</b>	integer	Max number of items shown in the List module

Key	Datatype	Description
<b>maxSingleDBListItems</b>	integer	Max number of items shown in the List module, if this table is listed in Extended mode (listing only a single table)

**Example**

This is how the "pages" table is configured for these settings (in t3lib/stddb/tables.php):

```
'interface' => array(  
    'showRecordFieldList' => 'doktype,title',  
    'maxDBListItems' => 30,  
    'maxSingleDBListItems' => 50  
),
```

## ['feInterface'] section

The "feInterface" section contains properties related to Front End Editing of the table, mostly related to the feAdmin\_lib.

Is deprecated in the sense that it will still exist, but will not be (and should not be) extended further.

Key	Datatype	Description
<b>editableRecordFields</b>	string (list of field names)	List of fields, example: '*name, *type, biography, filmography'. Used for front-end edit module created by Rene Fritz <r.fritz@colorcube.de>
<b>fe_admin_fieldList</b>	string (list of field names)	List of fields allowed for editing/creation with the fe_adminLib module, see media/scripts/fe_adminLib, example: 'pid,name,title,address'

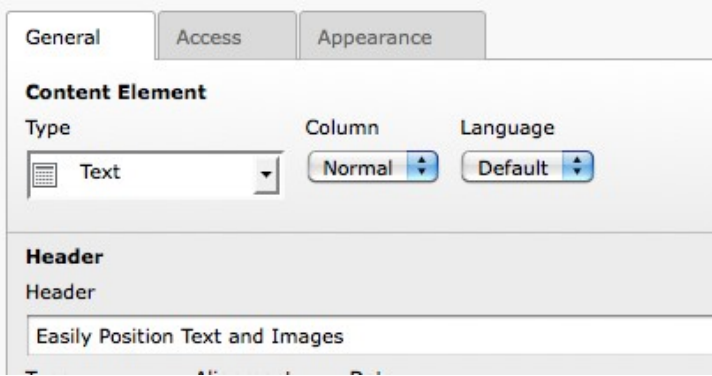
## ['columns'][\$field name] section

The "columns" section contains configuration for each table *field* (also called "column") which can be edited by the backend.

The configuration includes both properties for the display in the backend as well as the processing of the submitted data.

Each field can be configured as a certain "type" (e.g. checkbox, selector, input field, text area, file or db-relation field, user defined etc.) and for each type a separate set of additional properties applies. These properties are clearly explained below for each type.

This table shows the keys of the ['columns'][\$field name] array:

Key	Datatype	Description	Scope
<b>label</b>	string or LLL reference	<b>Required!</b> The name of the field as it is shown in the interface: 	Display
<b>exclude</b>	boolean	If set, all backend users are prevented from editing the field unless they are members of a backend user group with this field added as an "Allowed Excludefield" (or "admin" user). See "Inside TYPO3" document about permissions.	Proc.
<b>l10n_mode</b>	string (keyword)	Localization mode. Only active if the ctrl-directive "languageField" is set.  The main relevance is when a record is localized by an API call in TCEmain that makes a copy of the default language record. You can think of this process as copying all fields from the source record, except if a special mode applies as defined below:  Keywords are: <ul style="list-style-type: none"> <li><b>exclude</b> – Field will not be shown in TCEforms if this record is a localization of the default language. (Works basically like a</li> </ul>	Display / Proc.



Key	Datatype	Description	Scope
		<p>display condition.) Excluded fields will not be copied when a language-copy is made. May have frontend implications similar to “mergeIfNotBlank”.</p> <ul style="list-style-type: none"> <li>• <b>mergeIfNotBlank</b> – Field will be editable but if the field value is blank the value from the default translation is used (this can be very useful for images shared from the default record). Requires frontend support. In the backend the effect is that the field content is not copied when a new “localization copy” is made.</li> <li>• <b>noCopy</b> – Like mergeIfNotBlank but without the implications for the frontend; The field is just not copied.</li> <li>• <b>prefixLangTitle</b> – The field will get copied, but the content is prefixed with the title of the language. Works only for field types like “text” and “input”</li> </ul> <p>As mentioned above if “l10n_mode” is not set for a given field, that field is just copied as is to the translated record.</p> <p>(Doesn't apply to flexform fields.)</p>	
<b>l10n_display</b>	list of keywords	<p>Localization display. see: <i>l10n_mode</i></p> <p>This option can be used to define the language related field rendering. This has nothing to do with the processing of language overlays and data storage but the display of form fields.</p> <p>Keywords are:</p> <ul style="list-style-type: none"> <li>• <b>hideDiff</b> – The differences to the default language field will not be displayed.</li> <li>• <b>defaultAsReadonly</b> – This renders the field as read only field with the content of the default language record. The field will be rendered even if 'l10n_mode' is set to 'exclude'. While 'exclude' define the field not to be translated this option activate display of the default data.</li> </ul>	Display
<b>l10n_cat</b>	string (keyword)	<p>Localization category.</p> <p><b>Keywords:</b> text,media</p> <p>When localization mode is set for a TCEforms, it must be either of these values. Only the fields that have l10n_cat set to the localization mode is shown. Used to limit display so only most relevant fields are shown to translators. It doesn't prevent editing of other fields if records are edited outside localization mode, it merely works as a display condition.</p> <p>It is also used in localization export (pending at this moment).</p>	Display
<b>config</b>	array	<p>Contains the actual configuration properties of the fields display and processing behavior. The possibilities for this array depend on the value of the array key "type" within the array. Each valid value for "type" is shown below in a separate table. Furthermore there are some properties common to all field types, described in the next chapter "[columns][field name][config] / Common properties".</p>	-
<b>displayCond</b>	string	<p>Contains a condition rule for whether to display the field or not.</p> <p>A rule is a string divided into several parts by ":" (colons). The first part is the rule-type and the subsequent parts will depend on the rule type. Currently these rule values can be used:</p> <ul style="list-style-type: none"> <li>• <b>FIELD</b> : This evaluates based on another fields value in the record. <ul style="list-style-type: none"> <li>• Part 1 is the field name</li> <li>• Part 2 is the evaluation type. These are the possible options: <ul style="list-style-type: none"> <li>• <b>REQ</b>: Requires the field to have a "true" value. False values</li> </ul> </li> </ul> </li> </ul>	Display

Key	Datatype	Description	Scope
		<p>are "" (blank string) and 0 (zero) or if the field does not exist at all. All else is true.</p> <p>For the REQ evaluation type Part3 of the rules string must be the string "true" or "false". If "true" then the rules returns "true" if the evaluation is true. If "false" then the rules returns "true" if the evaluation is false.</p> <ul style="list-style-type: none"> <li>• <b>&gt; / &lt; / &gt;= / &lt;=</b> : Evaluates if the field value is greater than, less than the value in "Part 3"</li> <li>• <b>= / !=</b> : Evaluates if the field value is equal to value in "Part 3" (or not, if the negation flag, "!" is prefixed)</li> <li>• <b>IN / !IN</b> : Evaluates if the field value is in the comma list equal to value in "Part 3" (or not, if the negation flag, "!" is prefixed)</li> <li>• <b>- / !-</b> : Evaluates if the field value is in the range specified by value in "Part 3" ([min] - [max]) (or not, if the negation flag, "!" is prefixed)</li> <li>• <b>EXT</b> : This evaluates based on current status of extensions. <ul style="list-style-type: none"> <li>• Part 1 is the extension key</li> <li>• Part 2 is the evaluation type: <ul style="list-style-type: none"> <li>• <b>LOADED</b> : Requires the extension to be loaded if Part3 is "true" and reversed if Part3 is "false".</li> </ul> </li> </ul> </li> <li>• <b>REC</b> : This evaluates based on the current record (doesn't make sense for flexform fields) <ul style="list-style-type: none"> <li>• Part 1 is the type. <ul style="list-style-type: none"> <li>• <b>NEW</b> : Requires the record to be new if Part2 is "true" and reversed if Part2 is "false".</li> </ul> </li> </ul> </li> <li>• <b>HIDE_L10N_SIBLINGS</b> : (FlexForms only) This evaluates based on whether the field is a value for the default language or an alternative language. Works only for &lt;langChildren&gt;=1, otherwise it has no effect. <ul style="list-style-type: none"> <li>• Part 1: Keywords: "except_admin" = will still show field to admin users</li> </ul> </li> <li>• <b>HIDE_FOR_NON_ADMINS</b>: This will hide the field for all non-admin users while admins can see it. Useful for FlexForm container fields which are not supposed to be edited directly via the FlexForm but rather through some other interface (TemplaVoilà's Page module for instance).</li> <li>• <b>VERSION</b>: <ul style="list-style-type: none"> <li>• Part 1 is the type: <ul style="list-style-type: none"> <li>• <b>IS</b> : Part 2 is "true" or "false": If true, the field is shown only if the record is a version (pid == -1)</li> </ul> </li> </ul> </li> </ul> <p>For FlexForm elements the fields are tags on same level. If &lt;langChildren&gt; is enabled, then the value of other fields on same level is taken from the same language.</p> <p>The field-values of the FlexForm-parent record are prefixed with "parentRec.". These fields can be used like every other field (since TYPO3 4.3).</p> <p><b>Example:</b> This example will require the field named "tx_templavoila_ds" to be true, otherwise the field for which this rule is set will not be displayed:</p> <pre>'displayCond' =&gt; 'FIELD:tx_templavoila_ds:REQ:true',</pre> <p>This example requires the extension "static_info_tables" to be loaded, otherwise the field is not displayed (this is useful if the field makes a look-up on a table coming from another extension!):</p> <pre>'displayCond' =&gt; 'EXT:static_info_tables:LOADED:true',</pre> <p>This example would require the header-field of the FlexForm-parent record to be true, otherwise the FlexForm field is not displayed (works only within FlexForm datastructure definitions):</p> <pre>&lt;displayCond&gt;FIELD:parentRec.header:REQ:true&lt;/displayCond&gt;</pre>	

Key	Datatype	Description	Scope
<b>defaultExtras</b>	string	<p>In the “types” configuration of a field you can specify on position 4 a string of "extra configuration". This string will be the default string of extra options for a field regardless of types configuration. For instance this can be used to create an RTE field without having to worry about special configuration in “types” config.</p> <p>This is also the way by which you can enable the RTE for FlexForm fields.</p> <p><b>Example value:</b></p> <pre>richtext[cut copy paste formatblock textcolor bold italic underline left center right orderedlist unorderedlist outdent indent link table image line chMode]:rte_transform[mode=ts_css imgpath=uploads/tx_mininews/rte/]</pre>	





## ['columns']['*field name*']['config'] / Common properties

There are a number of properties which are common to all field types. They are described below.




Key	Datatype	Description	Scope
<b>type</b>	string	This defines the type of field. It must one of the values described in the following chapters.	Display / Proc.
<b>form_type</b>	string	This will override the field type when displaying it as a form. It can take any of the values available for "type" above.	Display
<b>default</b>	-	This property can be used to set a default value for the field. Its data type is whatever is appropriate for the given field.	Display / Proc.
<b>softref</b>	string	Used to attach "soft reference parsers". See under "Additional TCA features" for information about softref keys. The syntax for this value is <code>key1,key2[parameter1;parameter2;...],...</code>	Proc.
<b>readOnly</b>	boolean	Renders the form in a way that the user can see the values but cannot edit them. The rendering is as similar as possible to the normal rendering but may differ in layout and size. <b>Notice:</b> Read-only rendering might not be implemented by user defined form items! It is up to each developer to implement read-only rendering for its own user-types.	Display
<b>search</b>	array	<p>Defines additional search-related options for a given field.</p> <ul style="list-style-type: none"> <li>• <b>pidonly (boolean):</b> searches in the column only if search happens on the single page (does not search the field if searching in the whole table)</li> <li>• <b>case (boolean):</b> makes the search case-sensitive. This requires a proper database collation for the field (see your database documentation)</li> <li>• <b>andWhere (string):</b> additional SQL WHERE statement without 'AND'. With this it is possible to place an additional condition on the field when it is searched (see example below).</li> </ul> <p><b>Example:</b> The "tt_content" table has the following definition:</p> <pre>\$TCA['tt_content'] = array(     ...     'columns' =&gt; array(         ...         'bodytext' =&gt; array(             ...             'config' =&gt; array(                 ...             ),             'search' =&gt; array(                 'andWhere' =&gt; 'CType=\'text\' OR CType=\'textpic\'',             )         ),         ...     ),     ... );</pre> <p>This means that the "bodytext" field of the "tt_content" table will be searched in only for elements of type Text and Text with image. This helps make any search more relevant.</p>	Search

## ['columns']['*field name*']['config'] / TYPE: "input"

The type "input" generates an <input> field, possibly with additional features applied.

Key	 <b>Page title:</b>	Scope
type	<input type="text" value="Above Center"/> 	Display / Proc.
size		to Display
max	 <b>Subtitle:</b>	Display
default	<input type="text"/>	Display / Proc.
eval	<input type="checkbox"/>  <b>Navigation title:</b>	Display / Proc.
<p>The evaluation functions will be executed in the list-order.</p> <p>Keywords:</p> <ul style="list-style-type: none"> <li><b>required</b> : A non-empty value is required in the field (otherwise the form cannot be saved).</li> <li><b>trim</b> : The value in the field will have white spaces around it trimmed away.</li> <li><b>date</b> : The field will evaluate the input as a date, automatically converting the input to a UNIX-time in seconds. The display will be like "12-8-2003" while the database value stored will be "1060639200".</li> <li><b>datetime</b> : The field will evaluate the input as a date with time (detailed to hours and minutes), automatically converting the input to a UNIX-time in seconds. The display will be like "16:32 12-8-2003" while the database value will be "1060698720".</li> <li><b>time</b> : The field will evaluate the input as a timestamp in seconds for the current day (with a precision of minutes). The display will be like "23:45" while the database will be "85500".</li> <li><b>timesec</b> : The field will evaluate the input as a timestamp in seconds for the current day (with a precision of seconds). The display will be like "23:45:13" while the database will be "85513".</li> <li><b>year</b> : Evaluates the input to a year between 1970 and 2038. If you need any year, then use "int" evaluation instead.</li> <li><b>int</b> : Evaluates the input to an integer.</li> <li><b>upper</b> : Converts to uppercase (only A-Z plus a selected set of Western European special chars).</li> <li><b>lower</b> : Converts the string to lowercase (only A-Z plus a selected set of Western European special chars).</li> <li><b>alpha</b> : Allows only a-zA-Z characters.</li> <li><b>num</b> : Allows only 0-9 characters in the field.</li> <li><b>alphanum</b> : Same as "alpha" but allows also "0-9"</li> <li><b>alphanum_x</b> : Same as "alphanum" but allows also "_" and "-" chars.</li> <li><b>nospace</b> : Removes all occurrences of space characters (chr(32))</li> <li><b>md5</b> : Will convert the inputted value to the md5-hash of it (The JavaScript MD5() function is found in typo3/md5.js)</li> </ul>		

Key	Datatype	Description	Scope
		<ul style="list-style-type: none"> <li><b>is_in</b> : Will filter out any character in the input string which is <i>not</i> found in the string entered in the key "is_in" (see below).</li> <li><b>password</b> : Will show "*****" in the field after entering the value and moving to another field. Thus passwords can be protected from display in the field. <b>Notice</b> that the value during <i>entering it</i> is visible!</li> <li><b>double2</b> : Converts the input to a floating point with 2 decimal positions, using the "." (period) as the decimal delimited (accepts also "," for the same).</li> <li><b>unique</b> : Requires the field to be unique for the <i>whole</i> table. (Evaluated on the server only). NOTICE: When selecting on unique-fields, make sure to select using "AND pid&gt;=0" since the field CAN contain duplicate values in other versions of records (always having PID = -1). This also means that if you are using versioning on a table where the unique-feature is used you cannot set the field to be truly unique in the database either!</li> <li><b>uniqueInPid</b> : Requires the field to be unique for the current PID (among other records on the same page). (Evaluated on the server only)</li> <li><b>tx_*</b> : User defined form evaluations. See below.</li> </ul> <p>All the above evaluations (unless noted) are done by JavaScript with the functions found in the script t3lib/jsfunc.evalfield.js  "(TCE)" means the evaluation is done in the TCE on the server. The class used for this is t3lib_TCEmain.</p> <p><b>Example:</b>  Setting the field to evaluate the input to a date returned to the database in UNIX-time (seconds)</p> <pre>'eval' =&gt; 'date',</pre> <p>Trimming the value for white space before storing in the database (important for varchar fields!)</p> <pre>'eval' =&gt; 'trim',</pre> <p>By this configuration the field will be stripped for any space characters, converted to lowercase, only accepted if filled in and on the server the value is required to be unique for all records from this table:</p> <pre>'eval' =&gt; 'nospace,lower,unique,required'</pre> <p><b>User-defined form evaluations:</b>  You can supply your own form evaluations in an extension by creating a class with two functions, one which returns the JavaScript code for client side validation called returnFieldJS() and one which does the server side validation called evaluateFieldValue().</p> <p><b>The function evaluateFieldValue() has 3 arguments:</b></p> <ul style="list-style-type: none"> <li><b>\$value</b> :The field value to be evaluated.</li> <li><b>\$is_in</b> : The "is_in" value of the field configuration from TCA (see below).</li> <li><b>&amp;\$set</b> : Boolean defining if the value is written to the database or not. Must be passed by reference and changed if needed.</li> </ul> <p><b>Example:</b></p> <pre>class.tx_exampleextraevaluations_extraeval1.php:  &lt;?php class tx_exampleextraevaluations_extraeval1 {     function returnFieldJS() {         return '             return value + " [added by JS]";         ';     }     function evaluateFieldValue(\$value, \$is_in, &amp;\$set)</pre>	

Key	Datatype	Description	Scope
		<pre> {     return \$value . ' [added by PHP]'; } ?&gt;  ext_localconf.php  &lt;?php // here we register "tx_exampleextraevaluations_extraeval1" \$TYPO3_CONF_VARS['SC_OPTIONS']['tce']['formevals'] ['tx_exampleextraevaluations_extraeval1'] = 'EXT:example_extraevaluations/ class.tx_exampleextraevaluations_extraeval1.php'; ?&gt; </pre>	
<b>is_in</b>	string	<p>If the evaluation type "is_in" (see above, under key "eval") is used for evaluation, then the characters in the input string should be found in this string as well. This value is also passed as argument to the evaluation function if a user-defined evaluation function is registered.</p>	Display / Proc.
<b>checkbox</b>	string	<p><b>This setting is not used anymore since TYPO3 4.5.</b> To set a default value, use the "default" property.</p> <p>If defined (even empty), a checkbox is placed <i>before</i> the input field. If a value other than the value of 'checkbox' (this value) appears in the input-field the checkbox is checked.</p> <p><b>Example:</b></p> <pre>'checkbox' =&gt; '123',</pre> <p>If you set this value then entering "12345" in the field will render this:</p>  <p>But if you either uncheck the checkbox or just enter the value "123" you will an empty input field and no checkbox set - however the value of the field <i>will be</i> "123":</p>  <p>This feature is useful for date-fields for instance. In such cases the checkbox will allow people to quickly remove the date setting (equal to setting the date to zero which actually means 1-1 1970 or something like that).</p> <p><b>Example listing:</b></p> <pre> 'config' =&gt; array(     'type' =&gt; 'input',     'size' =&gt; '8',     'max' =&gt; '20',     'eval' =&gt; 'date',     'checkbox' =&gt; '0',     'default' =&gt; '0' ) </pre> <p>Will create a field like this below. Checking the checkbox will insert the date of the current day. Unchecking the checkbox will just remove the value and silently sent a zero to the server (since the value of the key "checkbox" is set to "0").</p> 	Display / Proc.
<b>range</b>	array	<p>An array which defines an integer range within which the value must be.</p> <p><b>Keys:</b>  "lower": Defines the lower integer value.</p>	Proc.

Key	Datatype	Description	Scope
		<p>"upper": Defines the upper integer value.</p> <p>You can specify both or only one of them.</p> <p><b>Notice:</b> This feature is evaluated <i>on the server only</i> so any regulation of the value will have happened after saving the form.</p> <p><b>Example:</b> Limits an integer to be within the range 10 to 1000:</p> <pre>'eval' =&gt; 'int', 'range' =&gt; array(     'lower' =&gt; 10,     'upper' =&gt; 1000 ),</pre> <p>In this example the upper limit is set to the last day in year 2020 while the lowest possible value is set to the date of yesterday.</p> <pre>'range' =&gt; array(     'upper' =&gt; mktime(0, 0, 0, 12, 31, 2020),     'lower' =&gt; mktime(0,0,0,date('m'), date('d'), date('Y')) )</pre>	
wizards	array	[See section later for options]	Display

Now follows some code listings as examples:

#### Example: A "date" field

This is the typical configuration for a date field, like "starttime":

```
'starttime' => array(
    'exclude' => 1,
    'label' => 'LLL:EXT:lang/locallang_general.php:LGL.starttime',
    'config' => array(
        'type' => 'input',
        'size' => '8',
        'max' => '20',
        'eval' => 'date',
        'default' => '0'
    )
),
```

#### Example: A "username" field

In this example the field is for entering a username (from "fe\_users"). A number of requirements are imposed onto the field, namely that it must be unique within the page where the record is stored, must be in lowercase and without spaces in it:

```
'username' => array(
    'label' => 'LLL:EXT:cms/locallang_tca.php:fe_users.username',
    'config' => array(
        'type' => 'input',
        'size' => '20',
        'max' => '50',
        'eval' => 'nospace,lower,uniqueInPid,required'
    )
),
```

#### Example: A typical input field

This is just a very typical configuration which sets the size and a character limit to the field. In addition the input value is trimmed for surrounding whitespace which is a very good idea when you enter values into varchar fields.

```
'name' => array(
    'exclude' => 1,
    'label' => 'LLL:EXT:lang/locallang_general.php:LGL.name',
    'config' => array(
        'type' => 'input',
```



```
        'size' => '40',  
        'eval' => 'trim',  
        'max' => '80'  
    ),  
),
```

***Example: Required values***

Here the field is required to be filled in:

```
    'title' => array(  
        'label' => 'LLL:EXT:cms/locallang_tca.php:fe_groups.title',  
        'config' => array(  
            'type' => 'input',  
            'size' => '20',  
            'max' => '20',  
            'eval' => 'trim,required'  
        )  
    ),
```

## ['columns']['field name']['config'] / TYPE: "text"

This field type generates a <textarea> field or inserts a RTE (Rich Text Editor).

Such a field looks like this:



The screenshot shows the 'TypoScript Configuration' window with the sub-header 'Page TSConfig'. It features a large, empty text area for configuration input. A small 'TS' icon is visible in the bottom right corner of the text area.

Key	Datatype	Description	Scope
<b>type</b>	string	<i>[Must be set to "text"]</i>	Display / Proc.
<b>cols</b>	integer	Abstract value for the width of the <textarea> field. To set the textarea to the full width of the form area, use the value 48. Default is 30.	Display
<b>rows</b>	integer	The number of rows in the textarea. May be corrected for harmonization between browsers. Will also automatically be increased if the content in the field is found to be of a certain length, thus the field will automatically fit the content.  Default is 5. Max value is 20.	Display
<b>wrap</b>	["off", "virtual"]	Determines the wrapping of the textarea field. There are two options: <ul style="list-style-type: none"> <li><b>"virtual"</b>: (Default) The textarea will automatically wrap the lines like it would be expected for editing a text.</li> <li><b>"off"</b>: The textarea will <i>not</i> wrap the lines as you would expect when editing some kind of code.</li> </ul> <p><b>Notice:</b> If the string "nowrap" is found among options in the fields extra configuration from the "types" listing this will override the setting here to "off".</p> <p><b>Example:</b> This configuration will create a textarea useful for entry of code lines since it will not wrap the lines:</p> <pre>'config' =&gt; array(     'type' =&gt; 'text',     'cols' =&gt; '40',     'rows' =&gt; '15',     'wrap' =&gt; 'off', )</pre>	Display
<b>default</b>	string	Default value	Display / Proc.
<b>eval</b>	list of keywords	Configuration of field evaluation. Some of these evaluation keywords will trigger a JavaScript pre-evaluation in the form. Other evaluations will be performed in the backend. The evaluation functions will be executed in the list-order.  Keywords: <ul style="list-style-type: none"> <li><b>required</b> : A non-empty value is required in the field (otherwise the form cannot be saved).</li> <li><b>trim</b> : The value in the field will have white spaces around it trimmed away.</li> <li><b>tx_*</b> : User-defined form evaluations. See the "eval" key description for input-type field above.</li> </ul>	Display / Proc.
<b>is_in</b>	string	If a user-defined evaluation is used for the field (see above, under key "eval"), then this values will be passed as argument to the user-defined evaluation function.	Display / Proc.

Key	Datatype	Description	Scope
<b>wizards</b>	array	[See section later for options]	Display

Now follows some code listings as examples:

**Example: A quite normal field**

This is the typical configuration for a textarea field:

```
'message' => array(
    'label' => 'LLL:EXT:sys_note/locallang_tca.php:sys_note.message',
    'config' => array(
        'type' => 'text',
        'cols' => '40',
        'rows' => '15',
    )
),
```

## ['columns']['field name']['config'] / TYPE: "check"

This type creates checkbox(es). Such elements might look like this:

You can also configure checkboxes to appear in an array:

You can have between 1 and 10 checkboxes and the field type in the database must be an integer. No matter how many checkboxes you have each check box will correspond to a single bit in the integer value. Even if there is only one checkbox (which in turn means that you should theoretically check the bit-0 of values from single-checkbox fields and not just whether it is true or false!).

Key	Datatype	Description	Scope
<b>type</b>	string	[Must be set to "check"]	Display / Proc.
<b>items</b>	array	<p>If set, this array will create an array of checkboxes instead of just a single "on/off" checkbox.</p> <p><b>Notice:</b> You can have a maximum of 10 checkboxes in such an array and each element is represented by a single bit in the integer value which ultimately goes into the database.</p> <p>In this array each entry is itself an array where the first entry is the label (string or LLL reference) and the second entry is a blank value. The value sent to the database will be an integer where each bit represents the state of a checkbox in this array.</p> <p><b>Example:</b></p> <pre>'items' =&gt; array(     array('Green tomatoes', ''),     array('Red peppers', '') ),</pre>	Display
<b>cols</b>	integer	<p>How many columns the checkbox array are shown in. Range is 1-10, 1 being default.</p> <p>(Makes sense only if the 'array' key is defining a checkbox array)</p>	Display
<b>showIfRTE</b>	boolean	If set, this field will show <i>only</i> if the RTE editor is enabled (which includes	Display

Key	Datatype	Description	Scope
		correct browser version and user-rights altogether.)	
<b>default</b>	integer	Setting the default value of the checkbox(es).  <b>Notice:</b> Each bit corresponds to a check box (even if only one checkbox which maps to bit-0).	Display / Proc.
<b>itemsProcFunc</b>	string (function reference)	PHP function which is called to fill / manipulate the array with elements.  The function/method will have an array of parameters passed to it (where the item-array is passed by reference in the key 'items'). By modifying the array of items, you alter the list of items. For more information, see how user-functions are specified in the section about 'wizards' some pages below here.	Display

Now follows some code listings as examples:

#### Example: A single checkbox

A plain vanilla checkbox:

```
'enforce_date' => array(
    'exclude' => 0,
    'label' => 'LLL:EXT:examples/locallang_db.xml:tx_examples_dummy.enforce_date',
    'config' => array(
        'type' => 'check',
    )
),
```

#### Example: A checkbox array

This is an example of a checkbox array with two checkboxes in it. The first checkbox will have bit-0 and the second bit-1:

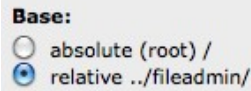
```
'l18n_cfg' => array(
    'exclude' => 1,
    'label' => 'LLL:EXT:cms/locallang_tca.xml:pages.l18n_cfg',
    'config' => array(
        'type' => 'check',
        'items' => array(
            array(
                'LLL:EXT:cms/locallang_tca.xml:pages.l18n_cfg.I.1',
                ''
            ),
            array(
                $GLOBALS['TYP03_CONF_VARS']['FE']
                ['hidePagesIfNotTranslatedByDefault'] ?
                'LLL:EXT:cms/locallang_tca.xml:pages.l18n_cfg
                .I.2a' :
                'LLL:EXT:cms/locallang_tca.xml:pages.l18n_cfg
                .I.2',
                ''
            ),
        )
    ),
),
```

If we wanted both checkboxes to checked by default, we would set the "default" property to '3' (since this contains both bit-0 and bit-1).

## ['columns']['field name']['config'] / TYPE: "radio"

Radio buttons are seldom used, but sometimes they can be more attractive than their more popular sisters (selector boxes).

Here you see radio buttons in action for the "Filemounts" records:



Key	Datatype	Description	Scope
<b>type</b>	string	<i>[Must be set to "radio"]</i>	<i>Display / Proc.</i>
<b>items</b>	array	<b>Required.</b>  An array of the values which can be selected. Each entry is in itself an array where the <i>first entry</i> is the <i>title</i> (string or LLL reference) and the <i>second entry</i> is the <i>value</i> of the radio button.  See example below.	Display
<b>default</b>	mixed	Default value.	Display / Proc.
<b>itemsProcFunc</b>	string (function reference)	PHP function which is called to fill / manipulate the array with elements.  The function/method will have an array of parameters passed to it (where the item-array is passed by reference in the key 'items'). By modifying the array of items, you alter the list of items. For more information, see how user-functions are specified in the section about 'wizards' some pages below here.	Display

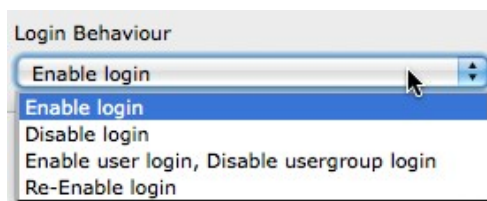
### Example:

An example of radio buttons configuration from "sys\_filemounts" (see above):

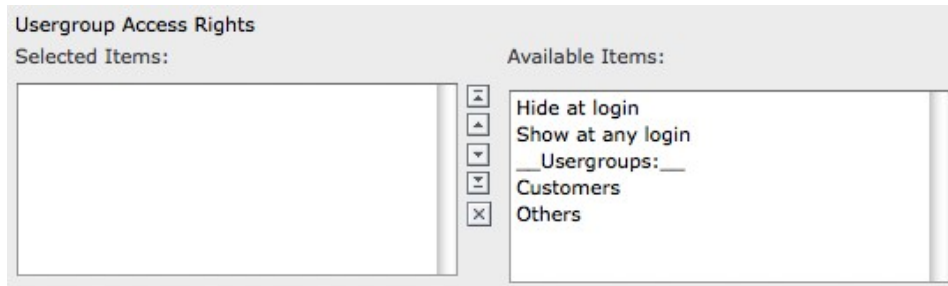
```
'base' => array(
    'label' => 'LLL:EXT:lang/locallang_tca.xml:sys_filemounts.base',
    'config' => array(
        'type' => 'radio',
        'items' => array(
            array('LLL:EXT:lang/locallang_tca.xml:sys_filemounts.base_absolute', 0),
            array('LLL:EXT:lang/locallang_tca.xml:sys_filemounts.base_relative', 1)
        ),
        'default' => 0
    )
)
```

## ['columns']['field name']['config'] / TYPE: "select"

Selectors boxes are very common elements in forms. By the "select" type you can create selector boxes. In the most simple form this is a list of values among which you can chose only one. In that way it is similar to the "radio" type above.



It is also possible to configure more complex types where the values from from a look up in another database table and you can even have a type where more than one value can be selected in any given order you like.



Key	Datatype	Description	Scope
<b>type</b>	string	[Must be set to "select"]	Display / Proc.
<b>items</b>	array	<p>Contains the elements for the selector box unless the property "foreign_table" or "special" has been set in which case automated values are set in addition to any values listed in this array.</p> <p>Each element in this array is in itself an array where:</p> <ul style="list-style-type: none"> <li>• First value is the <b>item label</b> (string or LLL reference)</li> <li>• Second value is the <b>value of the item</b>.</li> <li>• The special value "--div--" is used to insert a non-selectable value that appears as a divider label in the selector box (only for maxitems &lt;=1)</li> <li>• Values must not contain "," (comma) and " " (vertical bar). If you want to use "authMode" you should also refrain from using ":" (colon).</li> <li>• Third value is an optional icon. Default from "t3lib/gfx/" but if prepended with "../" it will be taken from any PATH_site directory. You can also prepend the files "ext/" and "sysext/" if they are in global extension directories. And finally - taking precedence over any other value - files prepended with "EXT:" will be found in the respective extension.</li> <li>• Fourth value is an optional description text. This is only shown when the list is shown by renderMode "checkbox".</li> <li>• Fifth value is reserved as keyword "EXPL_ALLOW" or "EXPL_DENY". See option "authMode" / "individual" for more details.</li> </ul> <p><b>Example:</b> A configuration could look like this:</p> <pre>'type' =&gt; 'select', 'items' =&gt; array(     array('English', ''),     array('Danish', 'dk'),     array('German', 'de'), )</pre> <p>A more complex example could be this (includes icons):</p> <pre>'type' =&gt; 'select', 'items' =&gt; array(     array('LLL:EXT:cms/locallang_ttc.php:k1', 0, 'selicons/k1.gif'),     array('LLL:EXT:cms/locallang_ttc.php:k2', 1, 'selicons/k2.gif'),     array('LLL:EXT:cms/locallang_ttc.php:k3', 2, 'selicons/k3.gif'), )</pre>	Display
<b>itemsProcFunc</b>	string	PHP function which is called to fill / manipulate the array with	Display

Key	Datatype	Description	Scope
	(function reference)	elements.  The function/method will have an array of parameters passed to it (where the item-array is passed by reference in the key 'items'). By modifying the array of items, you alter the list of items. For more information, see how user-functions are specified in the section about 'wizards' some pages below here.	
<b>selicon_cols</b>	integer (>0)	The number of rows in which to position the icons for the selector box. Default is to render as many columns as icons.	Display
<b>suppress_icons</b>	string	Lets you disable display of icons. Can be nice to do if icons are coming from foreign database records and you don't want them. Set it to "IF_VALUE_FALSE" if you <i>only</i> want to see icons when a value (non-blank, non-zero) is selected. Otherwise no icons are shown. Set it to "ONLY_SELECTED" if you <i>only</i> want to see an icon for the selected item. Set to "1" (true) if you never want any icons.	Display
<b>iconsInOptionTags</b>	boolean	If set, icons will appear in the <option> tags of the selector box. This feature seems only to work in Mozilla.	Display
<b>noIconsBelowSelect</b>	boolean	Disables the rendering of the icons after the select even when icons for the <select>s <option> tags were supplied and iconsInOptionTags was set.	Display
<b>foreign_table</b>	string (table name)	The item-array will be filled with records from the table defined here. The table must be configured in \$TCA. See the other related options below.	Proc. / Display
<b>foreign_table_where</b>	string (SQL WHERE clause)	The items from "foreign_table" are selected with this WHERE-clause. The table is joined with the "pages"-table and items are selected only from pages where the user has read access! (Not checking DB mount limitations!)  <b>Example:</b>  <pre>AND [foreign_table].pid=0 ORDER BY [foreign_table].sorting</pre> <b>Markers:</b> You can use markers in the WHERE clause: <ul style="list-style-type: none"> <li>• <code>###REC_FIELD_[field name]###</code></li> <li>• <code>###THIS_UID###</code> - is current element uid (zero if new).</li> <li>• <code>###CURRENT_PID###</code> - is the current page id (pid of the record).</li> <li>• <code>###STORAGE_PID###</code></li> <li>• <code>###SITEROOT###</code></li> <li>• <code>###PAGE_TSCONFIG_ID###</code> - a value you can set from Page TSconfig dynamically.</li> <li>• <code>###PAGE_TSCONFIG_IDLIST###</code> - a value you can set from Page TSconfig dynamically.</li> <li>• <code>###PAGE_TSCONFIG_STR###</code> - a value you can set from Page TSconfig dynamically.</li> </ul> <p>The markers are preprocessed so that the value of CURRENT_PID and PAGE_TSCONFIG_ID are always integers (default is zero), PAGE_TSCONFIG_IDLIST will always be a comma-separated list of integers (default is zero) and PAGE_TSCONFIG_STR will be addslashes'ed before substitution (default is blank string).</p> <p>See example below "Simple selector box with TSconfig markers".</p>	Proc. / Display
<b>foreign_table_prefix</b>	string or LLL reference	Label prefix to the title of the records from the foreign-table.	Display
<b>foreign_table_loadIcons</b>	boolean	If set, then the icons for the records of the foreign table are loaded and presented in the form.	Display

Key	Datatype	Description	Scope
		This depends on the 'selicon_field' of the foreign tables [ctrl] section being configured.	
<b>neg_foreign_table</b> <b>neg_foreign_table_where</b> <b>neg_foreign_table_prefix</b> <b>neg_foreign_table_loadIcons</b> <b>neg_foreign_table_importValueField</b>	[mixed]	<p>Four options corresponding to the 'foreign_table'-keys but records from this table will be referenced by <i>negative</i> uid-numbers (unless if MM is configured in which case it works like the group-type).</p> <p>'neg_foreign_table' is active only if 'foreign_table' is defined also.</p>	Display / Proc.
<b>fileFolder</b>	string	<p>Specifying a folder from where files are added to the item array. Specify the folder relative to the PATH_site, possibly using the prefix "EXT:" to point to an extension folder. Files from the folder is selected recursively to the level specified by "fileFolder_recurisions" (see below) and only files of the extension defined by "fileFolder_extList" is selected (see below). Only the file reference relative to the "fileFolder" is stored. If the files are images (gif,png,jpg) they will be configured as icons (third parameter in items array).</p> <p><b>Example:</b></p> <pre> 'config' =&gt; array (   'type' =&gt; 'select',   'items' =&gt; array (     array('', 0),   ),   'fileFolder' =&gt;   'EXT:cms/tslib/media/flags/' ,   'fileFolder_extList' =&gt; 'png,jpg,jpeg,gif',   'fileFolder_recurisions' =&gt; 0,   'selicon_cols' =&gt; 8,   'size' =&gt; 1,   'minitems' =&gt; 0,   'maxitems' =&gt; 1, ) </pre>	Display / Proc
<b>fileFolder_extList</b>	string	<p>List of extensions to select. If blank, all files are selected. Specify list in lowercase.</p> <p>See "t3lib_div::getAllFilesAndFoldersInPath()"</p>	Display / Proc
<b>fileFolder_recurisions</b>	integer	<p>Depth of directory recursions. Default is 99. Specify in range from 0-99.</p> <p>0 (zero) means no recursion into subdirectories.</p> <p>See "t3lib_div::getAllFilesAndFoldersInPath()"</p>	Display / Proc
<b>allowNonIdValues</b>	boolean	<p><b>If "foreign_table" is enabled:</b></p> <p>If set, then values which are not integer ids will be allowed. May be needed if you use itemsProcFunc or just enter additional items in the items array to produce some string-value elements for the list. Notice: If you mix non-database relations with database relations like this, DO NOT use integers for values and DO NOT use "_" (underscore) in values either!</p> <p>Notice: Will not work if you also use "MM" relations!</p>	Proc.
<b>default</b>	string	<p>Default value.</p> <p>If empty, the first element in the items array is selected.</p>	Display / Proc.
<b>dontRemapTablesOnCopy</b>		<p>(See same feature for type="group", internal_type="db")</p> <p>Set it to the exact same value as "foreign_table" if you don't want values to be remapped on copy.</p>	Proc.
<b>rootLevel</b>	boolean	<p>If set, the "foreign_table_where" will be ignored and a "pid=0" will be added to the query to select only records from root level of the page tree.</p>	Display
<b>MM</b>	string (table name)	<p>Means that the relation to the records of "foreign_table" / "neg_foreign_table" is done with a M-M relation with a third "join" table.</p>	Proc.



Key	Datatype	Description	Scope
		<p>That table has three columns as a minimum:</p> <ul style="list-style-type: none"> <li><i>uid_local</i>, <i>uid_foreign</i> for uids respectively.</li> <li><i>sorting</i> is a required field used for ordering the items</li> <li><i>sorting_foreign</i> is required if the relation is bidirectional (see description and example below table)</li> <li><i>tablenames</i> is used if multiple tables are allowed in the relation.</li> <li><i>uid</i> (auto-incremented and PRIMARY KEY) may be used if you need the “multiple” feature (which allows the same record to be references multiple times in the box. See “MM_hasUidField”</li> <li>Other fields may exist, in particular if MM_match_fields is involved in the set up.</li> </ul> <p><b>Example SQL #1</b> (most simple MM table):</p> <pre>CREATE TABLE user_testmmrelations_one_rel_mm (   uid_local int(11) DEFAULT '0' NOT NULL,   uid_foreign int(11) DEFAULT '0' NOT NULL,   sorting int(11) DEFAULT '0' NOT NULL,    KEY uid_local (uid_local),   KEY uid_foreign (uid_foreign) );</pre> <p><b>Example SQL #2</b> (Advanced with UID field, “ident” used with MM_match_fields and sorting_foreign for bidirectional MM relations):</p> <pre># # Table structure for table # 'user_testmmrelations_two_rel_mm' # # CREATE TABLE user_testmmrelations_two_rel_mm (   uid int(11) NOT NULL auto_increment,   uid_local int(11) DEFAULT '0' NOT NULL,   uid_foreign int(11) DEFAULT '0' NOT NULL,   tablenames varchar(30) DEFAULT '' NOT NULL,   sorting int(11) DEFAULT '0' NOT NULL,   sorting_foreign int(11) DEFAULT '0' NOT NULL,   ident varchar(30) DEFAULT '' NOT NULL,    KEY uid_local (uid_local),   KEY uid_foreign (uid_foreign),   PRIMARY KEY (uid), );</pre> <p>The field name of the config is not used for data-storage anymore but rather it's set to the number of records in the relation on each update, so the field should be an integer. Notice: Using MM relations you can ONLY store real relations for foreign tables in the list - no additional string values or non-record values.</p> <p><b>MM relations and flexforms</b> MM relations has been tested to work with flexforms if not in a repeated element in a section. See example below.</p>	
<b>MM_opposite_field</b>	string (field name)	<p>If you want to make an mm relation editable from the foreign side (bidirectional) of the relation as well, you need to set MM_opposite_field on the foreign side to the field name on the local side.</p> <p>E.g. if the field "companies.employees" is your local side and you want to make the same relation editable from the foreign side of the relation in a field called persons.employers, you would need to set the MM_opposite_field value of the TCA configuration of the persons.employers field to the string "employees".</p>	Proc.

Key	Datatype	Description	Scope
		<i>Notice: Bidirectional references only gets registered once on the native side in sys_refindex</i>	
<b>MM_match_fields</b>	array	Array of field=>value pairs to both insert and match against when writing/reading MM relations	
<b>MM_insert_fields</b>	array	Array of field=>value pairs to insert when writing new MM relations	
<b>MM_table_where</b>	string (SQL WHERE)	Additional where clause used when reading MM relations.	
<b>MM_hasUidField</b>	boolean	If the "multiple" feature is used with MM relations you MUST set this value to true and include a UID field! Otherwise sorting and removing relations will be buggy.	
<b>special</b>	string (any of keywords)	<p>This configures the selector box to fetch content from some predefined internal source. These are the possibilities:</p> <ul style="list-style-type: none"> <li>• <b>tables</b> - the list of TCA tables is added to the selector (excluding "adminOnly" tables).</li> <li>• <b>pagetypes</b> - all "doktype"-values for the "pages" table are added.</li> <li>• <b>exclude</b> - the list of "excludeFields" as found in \$TCA is added.</li> <li>• <b>modListGroup</b> - module-lists added for groups.</li> <li>• <b>modListUser</b> - module-lists added for users.</li> <li>• <b>explicitValues</b> - List values that require explicit permissions to be allowed or denied. (See "authMode" directive for the "select" type).</li> <li>• <b>languages</b> - List system languages (sys_language records from page tree root + Default language)</li> <li>• <b>custom</b> - Custom values set by backend modules (see TYPO3_CONF_VARS[BE][customPermOptions])</li> </ul> <p>As you might have guessed these options are used for backend user management and pretty worthless for most other purposes.</p>	Display / Proc.
<b>size</b>	integer	Height of the selector box in TCEforms.	Display
<b>autoSizeMax</b>	integer	If set, then the height of multiple-item selector boxes (maxitems > 1) will automatically be adjusted to the number of selected elements, however never less than "size" and never larger than the integer value of "autoSizeMax" itself (takes precedence over "size"). So "autoSizeMax" is the maximum height the selector can ever reach.	Display
<b>selectedListStyle</b>	string	If set, this will override the default style of the selector box with selected items (which is "width:200px"). Applies for when maxitems is > 1	Display
<b>itemListStyle</b>	string	If set, this will override the default style of the selector box with available items to select (which is "width:200px"). Applies for when maxitems is > 1	Display
<b>renderMode</b>	string (any of keywords)	<p>(Only for maxitems &gt; 1)</p> <p>Renders the list of multiple options as either a list of checkboxes or as a selector box with multiple choices. The data type is fully compatible with an ordinary multiple element list except that duplicate values cannot be represented for obvious reasons (option "multiple" does not work) and the order of values is fixed.</p> <p>Keywords are:</p> <ul style="list-style-type: none"> <li>• <b>checkbox</b> - Renders a list of checkboxes</li> <li>• <b>singlebox</b> - Renders a single multiple selector box</li> <li>• <b>tree</b> - Renders the selector as tree. This will work properly only when referring to a foreign table, so make sure that the "foreign_table" property is set. See "treeConfig" property configuration options.</li> </ul>	

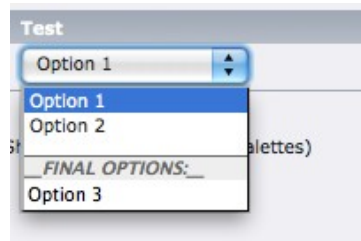
Key	Datatype	Description	Scope
		<p>When renderMode is “checkbox” or “singlebox” all values selected by “foreign_table” settings will automatically have their icon part in the items array set to the record icon (unless overruled by “selicon_field” of that table).</p> <p><b>Notice:</b> “maxitems” and “minitems” are not enforced in the browser for any of the render modes here! However they will be on the server. It is recommended to set “minitems” to zero and “maxitems” to a very large number exceeding the possible number of values you can select (for instance set it to 1000 or so).</p>	
<b>treeConfig</b>	(configuration options)	<p>Configuration if the renderMode is set to "tree". Either childrenField or parentField has to be set - childrenField takes precedence.</p> <p><b>Sub-properties:</b></p> <ul style="list-style-type: none"> <li>• <b>childrenField (string):</b> Field name of the foreign_table that references the uid of the child records (either child</li> <li>• <b>parentField (string):</b> Field name of the foreign_table that references the uid of the parent record</li> <li>• <b>rootUid (integer, optional):</b> uid of the record that shall be considered as the root node of the tree. In general this might be set by Page TSconfig</li> <li>• <b>appearance (array, optional):</b> <ul style="list-style-type: none"> <li>• <b>showHeader (boolean):</b> Whether to show the header of the tree that contains a field to filter the records and allows to expand or collapse all nodes</li> <li>• <b>expandAll (boolean):</b> Whether to show the tree with all nodes expanded</li> <li>• <b>maxLevels (integer):</b> The maximal amount of levels to be rendered (can be used to stop possible recursions)</li> <li>• <b>nonSelectableLevels (list, default "0"):</b> Comma-separated list of levels that will not be selectable, by default the root node (which is "0") cannot be selected</li> </ul> </li> </ul>	
<b>multiple</b>	boolean	<p>Allows the <i>same item</i> more than once in a list.</p> <p>If used with bidirectional MM relations it must be set for both the native and foreign field configuration. Also, with MM relations in general you must use a UID field in the join table, see description for “MM”</p>	Display / Proc.
<b>maxitems</b>	integer > 0	Maximum number of items in the selector box. (Default = 1)	Display / Proc
<b>minitems</b>	integer > 0	Minimum number of items in the selector box. (Default = 0)	Display
<b>wizards</b>	array	[See section later for options]	Display
<b>disableNoMatchingValueElement</b>	boolean	If set, then no element is inserted if the current value does not match any of the existing elements. A corresponding options is also found in Page TSconfig.	Display
<b>authMode</b>	string keyword	<p>Authorization mode for the selector box. Keywords are:</p> <ul style="list-style-type: none"> <li>• <b>explicitAllow</b> – All static values from the “items” array of the selector box will be added to a matrix in the backend user configuration where a value must be explicitly selected if a user (other than admin) is allowed to use it!</li> <li>• <b>explicitDeny</b> – All static values from the “items” array of the selector box will be added to a matrix in the backend user configuration where a value must be explicitly selected if a user should be denied access.</li> <li>• <b>individual</b> – State is individually set for each item in the selector box. This is done by the keywords “EXPL_ALLOW” and “EXPL_DENY” entered at the 5. position in the item array (see “items” configuration above). Items without any of</li> </ul>	Display / Proc

Key	Datatype	Description	Scope
		<p>these keywords can be selected as usual without any access restrictions applied.</p> <p><b>Notice:</b> The authentication modes will work only with values that are statically present in the “items” configuration. Any values added from foreign tables, file folder or by user processing will <i>not</i> be configurable and the evaluation of such values is not guaranteed for!</p> <p><b>maxitems &gt; 1</b>  “authMode” works also for selector boxes with maxitems &gt; 1. In this case the list of values is traversed and each value is evaluated. Any disallowed values will be removed.  If all submitted values turns out to be removed the result will be that the field is not written – basically leaving the old value. For maxitems &lt;=1 (single value) this means that a non-allowed value is just not written. For multiple values (maxitems &gt;1) it depends on whether any elements are left in the list after evaluation of each value.</p>	
<b>authMode_enforce</b>	string keyword	<p>Various additional enforcing options for authMode.</p> <p>Keywords are:</p> <ul style="list-style-type: none"> <li><b>strict</b> - If set, then permission to edit the record will be granted only if the “authMode” evaluates OK. The default is that a record having an authMode configured field with a “non-allowed” value can be edited – just the value of the authMode field cannot be set to a value that is not allowed.</li> </ul> <p><b>Notice:</b> This works only when maxitems &lt;=1 (and no MM relations) since the “raw” value in the record is all that is evaluated!</p>	Display / Proc
<b>exclusiveKeys</b>	string (list of)	<p>List of keys that exclude any other keys in a select box where multiple items could be selected.</p> <p>“Show at any login” of “fe_groups” (tables “pages” and “tt_content”) is an example where such a configuration is used.</p>	
<b>localizeReferencesAtParentLocalization</b>	boolean	<p>Defines whether referenced records should be localized when the current record gets localized (mostly used in Inline Relational Record Editing)</p>	Proc.

Here follow some code listings as examples:

**Example - A simple selector box:**

This is the most simple selector box you can get. It contains a static set of options you can select from:



```
'tx_examples_options' => array (
    'exclude' => 0,
    'label' => 'LLL:EXT:examples/locallang_db.xml:fe_users.tx_examples_options',
    'config' => array (
        'type' => 'select',
        'items' => array (
            array('LLL:EXT:examples/locallang_db.xml:fe_users.tx_examples_options.I.0', '1'),
            array('LLL:EXT:examples/locallang_db.xml:fe_users.tx_examples_options.I.1', '2'),
```

```
array('LLL:EXT:examples/locallang_db.xml:fe_users.tx_examples_options.I.2', '--div--'),
array('LLL:EXT:examples/locallang_db.xml:fe_users.tx_examples_options.I.3', '3'),
),
'size' => 1,
'maxitems' => 1,
),
),
```

In the configuration the elements are configured by the "items" array. Each entry in the array contains pairs of label/value. Notice the third entry of the "items" array. It defines a *divider*. This value cannot be selected. It only helps to divide the list of options with a label indicating a new section.

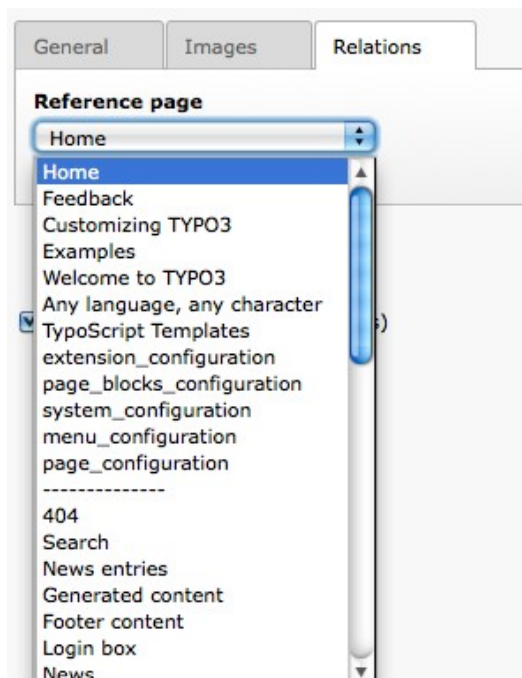
#### Example - Simple selector box with TSconfig markers

This example shows the use of markers inside the "foreign\_table\_where" clause and how the corresponding TSconfig must be set up.

In the TCA definition of the "haiku" table ("examples" extension) there is a simple select field to create a reference to a page in the "pages" table:

```
'reference_page' => array(
    'label' => 'LLL:EXT:examples/locallang_db.xml:tx_examples_haiku.reference_page',
    'config' => array(
        'type' => 'select',
        'foreign_table' => 'pages',
        'foreign_table_where' => "AND pages.title LIKE '%###PAGE_TSCONFIG_STR###'",
        'size' => 1,
        'minitems' => 0,
        'maxitems' => 1
    ),
),
```

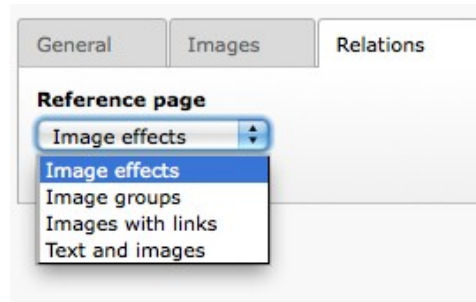
Without any TSconfig, the selector will display a full list of pages:



Let's add the following bit of Tsconfig to the page containing our "haiku" record:

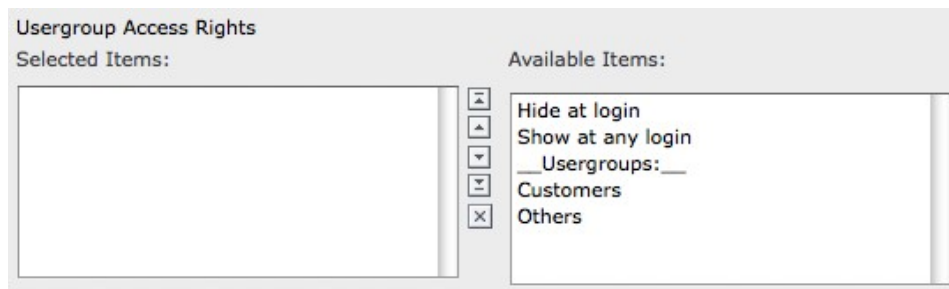
```
TCEFORM.tx_examples_haiku.reference_page.PAGE_TSCONFIG_STR = image
```

The list of pages that we can select from is now reduced to:



**Example - A multiple value selector with contents from a database table**

The user group selector is based on the fe\_groups table. It appears as a multiple selector:



The corresponding TCA configuration:

```
'fe_group' => array(
    'exclude' => 1,
    'label' => 'LLL:EXT:lang/locallang_general.xml:LGL.fe_group',
    'config' => array(
        'type' => 'select',
        'size' => 7,
        'maxitems' => 20,
        'items' => array(
            array(
                'LLL:EXT:lang/locallang_general.xml:LGL.hide_at_login',
                -1,
            ),
            array(
                'LLL:EXT:lang/locallang_general.xml:LGL.any_login',
                -2,
            ),
            array(
                'LLL:EXT:lang/locallang_general.xml:LGL.usergroups',
                '--div--',
            ),
        ),
        'exclusiveKeys' => '-1,-2',
        'foreign_table' => 'fe_groups',
        'foreign_table_where' => 'ORDER BY fe_groups.title',
    ),
),
```

The value stored in the database will be a *comma list of uid numbers* of the records selected.

The interesting point of this example is that it shows that static values can be mixed with values fetched from a database table.

### Example - Using a look up table for single value

In this case the selector box looks up languages in a static table from an extension "static\_info\_tables":



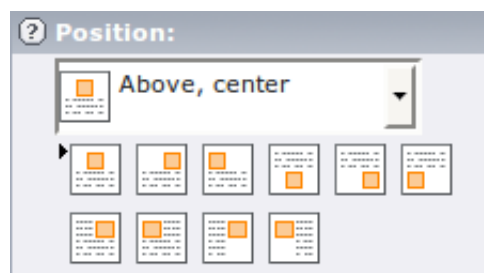
The configuration looks like this (taken from the sys\_lang table):

```
'static_lang_isocode' => array(
    'exclude' => 1,
    'label' => 'LLL:EXT:lang/locallang_tca.php:sys_language.isocode',
    'displayCond' => 'EXT:static_info_tables:LOADED:true',
    'config' => array(
        'type' => 'select',
        'items' => array(
            array('', 0),
        ),
        'foreign_table' => 'static_languages',
        'foreign_table_where' => 'AND static_languages.pid=0 ORDER BY
static_languages.lg_name_en',
        'size' => 1,
        'minitems' => 0,
        'maxitems' => 1,
    ),
),
```

Notice how a condition is set that this box should only be displayed *if* the extension it relies on exists! This is very important since otherwise the table will not be in the database and we will get SQL errors.

### Example - Adding icons for selection

This example shows how you can add icons to the selection choice very easily. Each icon is associated with an option in the selector box and clicking the icon will automatically select the option in the selector box and move the black arrow:



The configuration looks like this.

```
'imageorient' => array(
    'label' => 'LLL:EXT:cms/locallang_ttc.xml:imageorient',
    'config' => array(
        'type' => 'select',
        'items' => array(
            array(
                'LLL:EXT:cms/locallang_ttc.xml:imageorient.I.0',
                0,
                'selicons/above_center.gif',
            ),
            array(
                'LLL:EXT:cms/locallang_ttc.xml:imageorient.I.1',
                1,
                'selicons/above_right.gif',
            ),
            array(
                'LLL:EXT:cms/locallang_ttc.xml:imageorient.I.2',
                2,
                'selicons/above_left.gif',
            ),
            ...
            array(
                'LLL:EXT:cms/locallang_ttc.xml:imageorient.I.10',
                26,
                'selicons/intext_left_nowrap.gif',
            ),
        ),
        'selicon_cols' => 6,
        'default' => '0',
        'iconsInOptionTags' => 1,
    ),
),
```

Notice how each label/value pair contains an icon reference on the third position. Towards the bottom the layout of the icons is defined as being arranged in 6 columns.

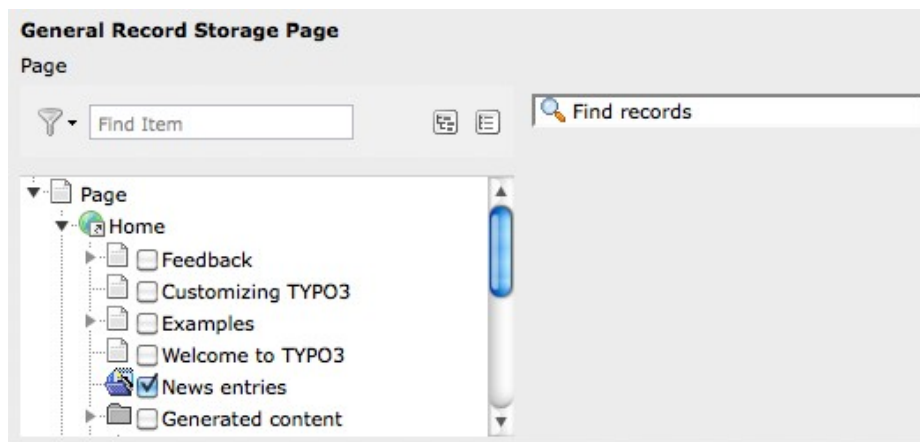
**Example - Render the General Record Storage Page selector as a tree of page**

The following configuration change:

```
t3lib_div::loadTCA('pages');
$tempConfiguration = array(
    'type' => 'select',
    'foreign_table' => 'pages',
    'size' => 10,
    'renderMode' => 'tree',
    'treeConfig' => array(
        'expandAll' => true,
        'parentField' => 'pid',
        'appearance' => array(
            'showHeader' => TRUE,
        ),
    ),
);
$TCA['pages']['columns']['storage_pid']['config'] = array_merge(
    $TCA['pages']['columns']['storage_pid']['config'],
    $tempConfiguration
);
```



Will transform the General Record Storage Page selector into:



### Example - Adding wizards

This example shows how wizards can be added to a selector box. The three typical wizards for a selector box is edit, add and list items. This enables the user to create new items in the look up table while being right at the selector box where he want to select them:

The configuration is rather long and looks like this (notice, that wizards are not exclusively available for selector boxes!):

```
'file_mountpoints' => array(
    'label' => 'LLL:EXT:lang/locallang_tca.xml:be_users.options_file_mounts',
    'config' => array(
        'type' => 'select',
        'foreign_table' => 'sys_filemounts',
        'foreign_table_where' => ' AND sys_filemounts.pid=0 ORDER BY
sys_filemounts.title',
        'size' => '3',
        'maxitems' => '10',
        'autoSizeMax' => 10,
        'iconsInOptionTags' => 1,
        'wizards' => array(
            '_PADDING' => 1,
            '_VERTICAL' => 1,
            'edit' => array(
                'type' => 'popup',
                'title' =>
                'LLL:EXT:lang/locallang_tca.xml:file_mountpoints_edit_title',
                'script' => 'wizard_edit.php',
                'icon' => 'edit2.gif',
                'popup_onlyOpenIfSelected' => 1,
                'JSopenParams' =>
                'height=350,width=580,status=0,menubar=0,scrollbars=1',
            ),
            'add' => array(
                'type' => 'script',
                'title' =>
                'LLL:EXT:lang/locallang_tca.xml:file_mountpoints_add_title',
                'icon' => 'add.gif',
                'params' => array(
                    'table' => 'sys_filemounts',
                    'pid' => '0',
                    'setValue' => 'prepend'
                ),
                'script' => 'wizard_add.php',
            ),
            'list' => array(
                'type' => 'script',
                'title' =>
                'LLL:EXT:lang/locallang_tca.xml:file_mountpoints_list_title',
                'icon' => 'list.gif',
                'params' => array(
                    'table' => 'sys_filemounts',
                    'pid' => '0',
                ),
                'script' => 'wizard_list.php',
            )
        )
    )
)
```

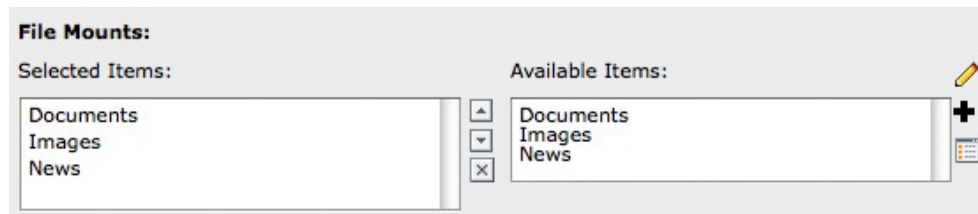
```

    ),
    ),
),

```

The part with the wizards is highlighted in bold. See the wizard section in this document for more information.

Notice the configuration of "autoSizeMax". This value will make the height of the selector boxes adjust



themselves automatically depending on the content in them. The result is as follows:

#### Example – Bidirectional MM relations

For a table, "user\_testmmrelations\_two", we have a field "rel" with configured with MM relations:

```

"rel" => Array (
    "exclude" => 1,
    "label" => "Relations:",
    "config" => Array (
        "type" => "select",
        "foreign_table" => "user_testmmrelations_one",
        "foreign_table_where" => "ORDER BY user_testmmrelations_one.uid",

        "size" => 10,
        "minitems" => 0,
        "maxitems" => 10,
        "MM" => "user_testmmrelations_two_rel_mm",
        'MM_match_fields' => array('ident'=>'table_two')
    )
),

```

The MM table is called "user\_testmmrelations\_two\_rel\_mm", and the field "ident" is used to match on with the word "table\_two". Doing this enables us to use the *same MM* table for other fields using other keywords for the "ident" field.

In another table "user\_testmmrelations\_one" a field called "rel2" constitutes the foreign side of the bidirectional relation:

```

"rel2" => Array (
    "label" => "Foreign relation:",
    "config" => Array (
        "type" => "select",
        "foreign_table" => "user_testmmrelations_two",
        "foreign_table_where" => "ORDER BY user_testmmrelations_two.uid",

        "size" => 10,
        "minitems" => 0,
        "maxitems" => 10,
        "MM" => "user_testmmrelations_two_rel_mm",
        'MM_match_fields' => array('ident'=>'table_two'),
        "MM_opposite_field" => "rel"
    )
),

```

Notice how in both cases "foreign\_table" points to the table name of the other. Also they use the exact same set up except in the foreign side case above the field "MM\_opposite\_field" is set to "rel" - the name of the field in table "user\_testmmrelations\_two"!

The SQL looks like:

```
#
# Table structure for table 'user_testmmrelations_two_rel_mm'
#
#
CREATE TABLE user_testmmrelations_two_rel_mm (
  uid int(11) NOT NULL auto_increment,
  uid_local int(11) DEFAULT '0' NOT NULL,
  uid_foreign int(11) DEFAULT '0' NOT NULL,
  tablenames varchar(30) DEFAULT '' NOT NULL,
  sorting int(11) DEFAULT '0' NOT NULL,
  sorting_foreign int(11) DEFAULT '0' NOT NULL,
  ident varchar(30) DEFAULT '' NOT NULL,

  KEY uid_local (uid_local),
  KEY uid_foreign (uid_foreign),
  PRIMARY KEY (uid),
);
```

In the backend the form could look like:

So, from a record in table two (native) there are two relations made to records in table one.

If we look at one of the records from table one we see the relation made from “TWO (1)”:

In the database it looks like this:

	uid	uid_local	uid_foreign	tablenames	sorting	sorting_foreign	ident
<input type="checkbox"/>	1	1	1		1	0	table_two
<input type="checkbox"/>	2	1	2		2	0	table_two

### Example – FlexForms and MM relations

MM relations can be used with flexforms. Here is an example:

The flexform field configuration looks like this:

```
<rell>
  <TCEforms>
    <label>Relation:</label>
    <config>
      <type>group</type>
      <internal_type>db</internal_type>
      <allowed>user_testmmrelations_three</allowed>
      <size>10</size>
      <minitems>0</minitems>
      <maxitems>10</maxitems>
      <MM>user_testmmrelations_two_rel_mm</MM>
      <MM_match_fields>
        <ident>table_one_flex</ident>
      </MM_match_fields>
      <multiple>1</multiple>
      <MM_hasUidField>1</MM_hasUidField>
    </config>
  </TCEforms>
</rell>
```

As you can see the same element (titled “3-3 (UID-3)”) is selected twice (the “<multiple>” flag has been set) – and as a consequence <MM\_hasUidField>1</MM\_hasUidField> is set as well. In fact this configuration is *sharing the MM table* with another field (see the previous example) so the configuration

```
<MM_match_fields>
  <ident>table_one_flex</ident>
</MM_match_fields>
```

makes sure all MM relations for this flexform field is marked with the string “table\_one\_flex”.

In the database the entry looks like:

		<u>uid</u>	<u>uid</u>	<u>local</u>	<u>uid</u>	<u>foreign</u>	<u>tablenames</u>	<u>sorting</u>	<u>sorting</u>	<u>foreign</u>	<u>ident</u>
<input type="checkbox"/>			1	1		1		1		1	table_two
<input type="checkbox"/>			2	1		2		2		0	table_two
<input type="checkbox"/>			3	63		3		1		0	table_one_flex
<input type="checkbox"/>			4	63		2		2		0	table_one_flex
<input type="checkbox"/>			5	63		3		3		0	table_one_flex

(The first two entries belong to that other field, see previous example).

Of course you can specify a dedicated join table to the flexform instead of sharing it.

**What will not work in flexforms** is if you put MM relation fields in elements that can get repeated, like in sections:

The screenshot shows a TCA flexform configuration with three sections. Each section has a 'Title of FlexForm' field and a 'Relation' field. The first two sections have a single relation entry (3-5 (UID-5) and 3-1 (UID-1) respectively), while the third section has two relation entries (3-3 (UID-3) and 3-2 (UID-2)). Each section also has a 'TEST three' button.

Here I have added three sections and tried to add entries to each. However, when saved the two last entries are loaded for all of them. The result of the save was:

The reason is that the fields all use the same uid (that of the record) to find the MM records. This could work when MM fields were used outside sections of flexform fields which could only occur one time per record, but here it's not possible.

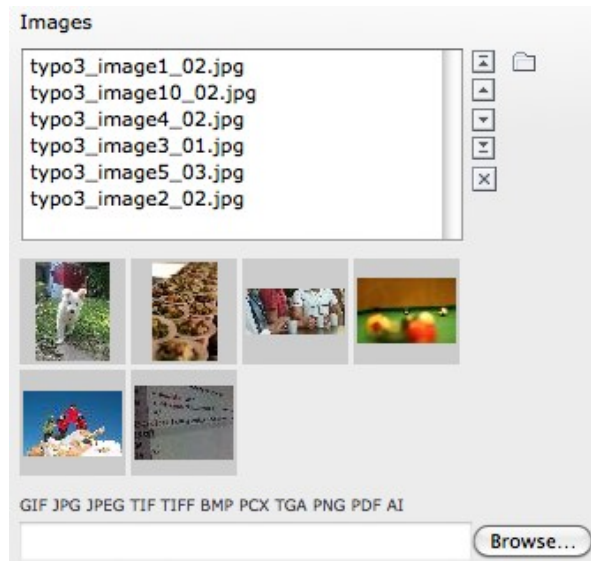
## Data format of "select" elements

Since the "select" element allows to store references to multiple elements we might want to look at how these references are stored internally. The principle is the same as with the "group" type (see below).

## ['columns']['field name']['config'] / TYPE: "group"

The group element in TYPO3 makes it possible to create references to records from multiple tables in the system. This is especially useful (compared to the "select" type) when records are scattered over the page tree and requires the Element Browser to be selected. In this example, Content Element records are attached (taken from the "Insert records" content element):

The "group" element is also the element you can use to bind files to records in TYPO3. In this case image files:



One thing to notice about such a field is that the files that are referenced actually get moved into an internal file folder for TYPO3! It does not create references to the files' original positions!

Key	Datatype	Description	Scope
<b>type</b>	string	[Must be set to "group"]	Display / Proc.
<b>internal_type</b>	string	<p><b>Required!</b> Configures the internal type of the "group" type of element. There are four possible options to choose from:</p> <ul style="list-style-type: none"> <li>"file" - this will create a field where files can be attached to the record</li> <li>"file_reference" - this will create a field where files can be referenced. In contrast to "file", referenced files (usually from fileadmin/) will not be copied to the upload folder. <b>Warning:</b> use this carefully. <i>Your references will be broken if you delete referenced files in the filesystem!</i></li> <li>"folder" - this will create a field where folders can be attached to the record</li> <li>"db" - this will create a field where database records can be attached as references.</li> </ul> <p>The default value is none of them - you must specify one of the values correctly!</p>	Display / Proc.
<b>allowed</b>	string (list of)	<p><b>For the "file" internal type (Optional):</b> A lowercase comma list of file extensions that are permitted. E.g. 'jpg,gif,txt'. Also see 'disallowed'.</p> <p><b>For the "db" internal type (Required!):</b> A comma list of tables from \$TCA. For example the value could be "pages,be_users". Value from these tables are always the 'uid' field. First table in list is understood as the <i>default table</i>, if a table-name is not prepended to the value. If the value is '*' then all tables are allowed (in this case <i>you should set "prepend_tname"</i> so all tables are prepended with their table name for sure).</p> <p><i>Notice</i>, if the field is the foreign side of a bidirectional MM relation, only the first table is used and that must be the table of the records on the native side of the relation.</p>	Proc. / Display
<b>disallowed</b>	string (list of)	[internal_type = <i>file</i> ONLY]	Proc. / Display

Key	Datatype	Description	Scope
		<p>Default value is '*' which means that anything file-extension which is not allowed is denied.</p> <p>If you set this value (to for example "php,php3") AND the "allowed" key is an empty string all extensions are permitted <i>except</i> ".php" and ".php3" files (works like the [BE][fileExtensions] config option).</p> <p>In other words:</p> <ul style="list-style-type: none"> <li>If you want to permit <i>only certain</i> file-extensions, use 'allowed' and not disallowed.</li> <li>If you want to permit <i>all file-extensions</i> except a few, set 'allowed' to blank ("" ) and enter the list of denied extensions in 'disallowed'.</li> <li>If you wish to <i>allow all extensions</i> with no exceptions, set 'allowed' to '*' and disallowed to "</li> </ul>	
<b>MM</b>	string (table name)	<p>Defines MM relation table to use.</p> <p>Means that the relation to the files/db is done with a M-M relation through a third "join" table.</p> <p>A MM-table must have these four columns:</p> <ul style="list-style-type: none"> <li><b>uid_local</b> - for the local uid.</li> <li><b>uid_foreign</b> - for the foreign uid. If the "internal_type" is "file" then the "uid_foreign" should be a varchar or 60 or so (for the filename) instead of an unsigned integer as you would use for the uid.</li> <li><b>tablenames</b> - is required if you use multi-table relations and this field must be a varchar of approx. 30 In case of files, the tablenames field is never used.</li> <li><b>sorting</b> - is a required field used for ordering the items.</li> </ul> <p><i>see [columns][fieldname][config] / TYPE: "select" =&gt; MM for additional features.</i></p>	Proc.
<b>MM_opposite_field</b>	string (field name)	<i>see [columns][fieldname][config] / TYPE: "select" =&gt; MM_opposite_field</i>	Proc.
<b>MM_match_fields</b>	array	<i>see [columns][fieldname][config] / TYPE: "select" =&gt; MM_match_fields</i>	
<b>MM_insert_fields</b>	array	<i>see [columns][fieldname][config] / TYPE: "select" =&gt; MM_insert_fields</i>	
<b>MM_table_where</b>	string (SQL WHERE)	<i>see [columns][fieldname][config] / TYPE: "select" =&gt; MM_table_where</i>	
<b>MM_hasUidField</b>	boolean	<i>see [columns][fieldname][config] / TYPE: "select" =&gt; MM_hasUidField</i>	
<b>max_size</b>	integer	<p><i>[internal_type = file ONLY]</i></p> <p>Files: Maximum file size allowed in KB</p>	Proc.
<b>uploadfolder</b>	string	<p><i>[internal_type = file ONLY]</i></p> <p>Path to folder under PATH_site in which the files are stored. Example: 'uploads' or 'uploads/pictures'</p> <p><b>Notice:</b> TYPO3 does NOT create a reference to the file in its original position! It makes a <i>copy</i> of the file into this folder and from that moment that file is not supposed to be manipulated from outside. Being in the upload folder means that files are understood as a part of the database content and should be managed by TYPO3 only.</p> <p><b>Warning:</b> do NOT add a trailing slash (/) to the upload folder otherwise the full path stored in the references will contain a double slash (e.g. "uploads/pictures//stuff.png").</p>	Proc.
<b>prepend_tname</b>	boolean	<i>[internal_type = db ONLY]</i>	Proc.

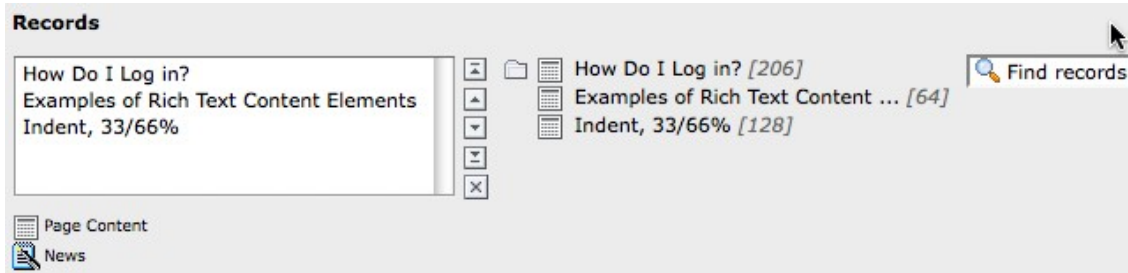


Key	Datatype	Description	Scope
		Will prepend the table name to the stored relations (so instead of storing "23" you will store e.g. "tt_content_23").	
<b>dontRemapTablesOnCopy</b>	string (list of tables)	<i>[internal_type = db ONLY]</i> A list of tables which should <i>not</i> be remapped to the new element uids if the field holds elements that are copied in the session.	Proc.
<b>show_thumbs</b>	boolean	Show thumbnails for the field in the TCEform	Display
<b>size</b>	integer	Height of the selector box in TCEforms.	Display
<b>autoSizeMax</b>	integer	If set, then the height of element listing selector box will automatically be adjusted to the number of selected elements, however never less than "size" and never larger than the integer value of "autoSizeMax" itself (takes precedence over "size"). So "autoSizeMax" is the maximum height the selector can ever reach.	Display
<b>selectedListStyle</b>	string	If set, this will override the default style of element selector box (which is "width:200px").	Display
<b>multiple</b>	boolean	Allows the <i>same item</i> more than once in a list.  If used with bidirectional MM relations it must be set for both the native and foreign field configuration. Also, with MM relations in general you must use a UID field in the join table, see description for "MM"	Display / Proc.
<b>maxitems</b>	integer > 0	Maximum number of items in the selector box. (Default = 1)	Display / Proc?
<b>minitems</b>	integer > 0	Minimum number of items in the selector box. (Default = 0)	Display / Proc?
<b>disable_controls</b>	string	Disables sub-controls inside "group" control. Comma-separated list of values. Possible values are: browser (disables browse button for list control), list (disables list and browse button, but not delete button), upload (disables upload control) and delete (disables the delete button). See example image below.  <b>NOTE:</b> if you use the delete button when the list is disabled, <b>all</b> entries in the list will be deleted.	Display / Proc.

Key	Datatype	Description	Scope
		<div> <div> <b>Images (all controls available)</b> <div> <div></div> <div> <div></div> <div></div> <div></div> </div> </div> <div> GIF JPG JPEG TIF TIFF BMP PCX TGA PNG PDF AI </div> <div> <div></div> <div>Browse...</div> </div> </div> <div> <b>Images (no file browser)</b> <div> <div></div> <div> <div></div> <div></div> <div></div> </div> </div> <div> GIF JPG JPEG TIF TIFF BMP PCX TGA PNG PDF AI </div> <div> <div></div> <div>Browse...</div> </div> </div> <div> <b>Images (no upload button)</b> <div> <div></div> <div> <div></div> <div></div> <div></div> </div> </div> <div> GIF JPG JPEG TIF TIFF BMP PCX TGA PNG PDF AI </div> <div> <div></div> <div></div> </div> </div> <div> <b>Images (no list display)</b> <div> <div></div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div>Browse...</div> </div> </div> <div> <b>Images (no delete button)</b> <div> <div></div> <div> <div></div> <div></div> </div> </div> <div> GIF JPG JPEG TIF TIFF BMP PCX TGA PNG PDF AI </div> <div> <div></div> <div>Browse...</div> </div> </div> <div> <b>Images (no controls at all)</b> </div> </div>	
wizards	array	[See section later for options]	Display

### Example - References to database records

The "Insert records" content element can be used to reference records from the "tt\_content" table (and possibly others, like "tt\_news" in the screenshot below):



The corresponding TCA code:

```
'records' => array(
    'label' => 'LLL:EXT:cms/locallang_ttc.xml:records',
    'config' => array(
        'type' => 'group',
        'internal_type' => 'db',
        'allowed' => 'tt_content',
        'size' => '5',
        'maxitems' => '200',
        'minitems' => '0',
        'show_thumbs' => '1',
        'wizards' => array(
            'suggest' => array(
                'type' => 'suggest',
            ),
        ),
    ),
),
```

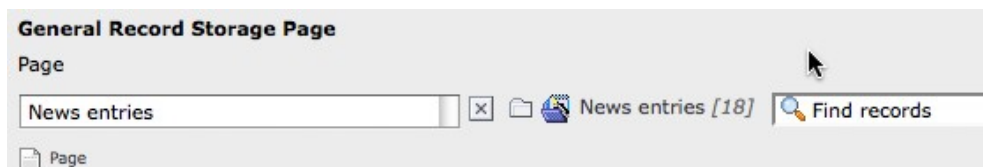
Note in particular how the "internal\_type" of the group field is set to "db". Then the allowed tables is defined as "tt\_content" (Content Elements table). This could very well be a list of tables which means you can mix references as you like!

The limit is set to a maximum of 200 references and thumbnails should be displayed, if possible. Finally a suggest wizard is added.

In this case it wouldn't have made sense to use a "select" type field since the situation implies that records might be found all over the system in a table which could potentially carry thousands of entries. In such a case the right thing to do is to use the "group" field so you have the Element Browser available for selector of the records.

### Example - Reference to another page

You will often see "group" type fields used when a reference to another page is required. This makes sense since pages can hardly be presented effectively in a big selector box and thus the Element Browser that follows the "group" type fields is useful. An example is the "General Record Storage page" reference:



The configuration looks like:

```
'storage_pid' => array(
    'exclude' => 1,
    'label' => 'LLL:EXT:lang/locallang_tca.php:storage_pid',
    'config' => array(
        'type' => 'group',
        'internal_type' => 'db',
        'allowed' => 'pages',
        'size' => '1',
        'maxitems' => '1',
    ),
),
```

```

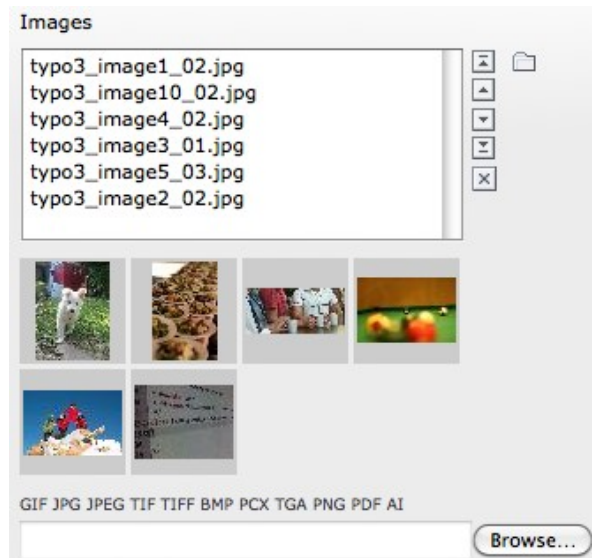
        'minitems' => '0',
        'show_thumbs' => '1',
        'wizards' => array(
            'suggest' => array(
                'type' => 'suggest',
            ),
        ),
    ),
),
),

```

Notice how "maxitems" is used to ensure that only one relation is created despite the ability of the "group" type field to create multiple references.

### Example - Attaching images

Attaching files to a database record is also achieved with group-type fields:



Notice how all the image names end with "\_0" and some number. This happens because all files attached to records through a group-type field are copied to a location defined by the "uploadfolder" setting in the configuration (see below). When a file is referenced several times, it is also copied several times. TYPO3 automatically appends a number so that each reference is unique.

```

'image' => array(
    'label' => 'LLL:EXT:lang/locallang_general.xml:LGL.images',
    'config' => array(
        'type' => 'group',
        'internal_type' => 'file',
        'allowed' => $GLOBALS['TYPO3_CONF_VARS']['GFX']['imagefile_ext'],
        'max_size' => $GLOBALS['TYPO3_CONF_VARS']['BE']['maxFileSize'],
        'uploadfolder' => 'uploads/pics',
        'show_thumbs' => '1',
        'size' => '3',
        'maxitems' => '200',
        'minitems' => '0',
        'autoSizeMax' => 40,
    ),
),

```

Notice how the "group" type is defined to contain files. Next the list of allowed file extensions are defined (here, taking the default list of image types for TYPO3). A maximum size (in kilobytes) for files is also defined. The "uploadfolder" property indicates that all files will be copied to the "uploads/pics" folder. Notice that this path is relative to the PATH\_site of TYPO3, one directory below PATH\_typo3.

### Data format of "group" elements

Since the "group" element allows to store references to multiple elements we might want to look at how these references are stored internally.

### Storage methods

There are two main methods for this:

- Stored in a comma list
- Stored with a join table (MM relation)

The default and most wide spread method is the comma list.

### Reserved tokens

In the comma list the token "," is used to separate the values. In addition the pipe sign "|" is used to separate value from label value when delivered to the interface. Therefore these tokens are not allowed in reference values, not even if the MM method is used.

### The "Comma list" method (default)

When storing references as a comma list the values are simply stored one after another, separated by a comma in between (with no space around!). The database field type is normally a varchar, text or blob field in order to handle this.

From the examples above the four Content Elements will be stored as "26,45,49,1" which is the UID values of the records. The images will be stored as their filenames in a list like "DSC\_7102\_background.jpg,DSC\_7181.jpg,DSC\_7102\_background\_01.jpg".

Since "db" references can be stored for multiple tables the rule is that uid numbers *without* a table name prefixed are implicitly from the first table in the allowed table list! Thus the list "26,45,49,1" is implicitly understood as "tt\_content\_26,tt\_content\_45,tt\_content\_49,tt\_content\_1". That would be equally good for storage, but by default the "default" table name is not prefixed in the stored string. As an example, lets say you wanted a relation to a Content Element and a Page in the same list. That would look like "tt\_content\_26,pages\_123" or alternatively "26,pages\_123" where "26" implicitly points to a "tt\_content" record given that the list of allowed tables were "tt\_content,pages".

### The "MM" method

Using the MM method you have to create a new database table which you configure with the key "MM". The table must contain a field, "uid\_local" which contains the reference to the uid of the record that contains the list of elements (the one you are editing). The "uid\_foreign" field contains the uid of the reference record you are referring to. In addition a "tablename" and "sorting" field exists if there are references to more than one table.

Lets take the examples from before and see how they would be stored in an MM table:

uid_local	uid_foreign	tablename	sorting
[uid of the record you are editing]	26	tt_content	1
[uid of the record you are editing]	45	tt_content	2
[uid of the record you are editing]	49	tt_content	3
[uid of the record you are editing]	1	tt_content	4

Or for "tt\_content\_26,pages\_123":

uid_local	uid_foreign	tablename	sorting
[uid of the record you are editing]	26	tt_content	1
[uid of the record you are editing]	123	pages	2

Or for "DSC\_7102\_background.jpg,DSC\_7181.jpg,DSC\_7102\_background\_01.jpg":

uid_local	uid_foreign	tablename	sorting
[uid of the record you are editing]	DSC_7102_background.jpg	N/A	1
[uid of the record you are editing]	DSC_7181.jpg	N/A	2
[uid of the record you are editing]	DSC_7102_background_01.jpg	N/A	3

### API for getting the reference list

In t3lib/ the class "t3lib\_loaddbgroup" is designed to transform the stored reference list values into an array where all uids are paired with the right table name. Also, this class will automatically retrieve the list of MM relations. In other words, it provides an API for getting the references from "group" elements into a PHP array regardless of storage method.

### Passing the list of references to TCEforms

Regardless of storage method, the reference list has to be "enriched" with proper title values when given to TCEforms for rendering. In particular this is important for database records. Passing the list "26,45,49,1" will not give TCEforms a chance to render the titles of the records.

The t3lib/ class "t3lib\_transferdata" is doing such transformations (among other things) and this is how the transformation happens:

Int. type:	In Database:	When given to TCEforms:
"db"	26,45,49,1	tt_content_26 %20ads%20asdf%20asdf%20,tt_content_45 This%20is%20a%20test%20%28copy%203%29,tt_content_49 %5B...%5D,tt_content_1 %5B...%5D
"file"	DSC_7102_background.jpg,DSC_7181.jpg,DSC_7102_background_01.jpg	DSC_7102_background.jpg DSC_7102_background.jpg,DSC_7181.jpg DSC_7181.jpg,DSC_7102_background_01.jpg DSC_7102_background_01.jpg

The syntax is:

```
[ref. value][ref. label rawurlencoded],[ref. value][ref. label rawurlencoded],....
```

Values are transferred back to the database as a comma separated list of values without the labels but if labels are in the value they are automatically removed.

Alternatively you can also submit each value as an item in an array; TCEmain will detect an array of values and implode it internally to a comma list. (This is used for the "select" type, in renderMode "singlebox" and "checkbox").

### Managing file references

When a new file is attached to a record the TCE will detect the new file based on whether it has a path prefixed or not. New files are copied into the upload folder that has been configured and the final value list going into the database will contain the new filename of the copy.

If images are removed from the list that is detected by simply comparing the original file list with the one submitted. Any files not listed anymore are deleted.

Examples:

Current DB value	Submitted data from TCEforms	New DB value	Processing done
first.jpg,second.jpg	first.jpg,/www/typo3/fileadmin/newfile.jpg,second.jpg	first.jpg,newfile_01.jpg,second.jpg	/www/typo3/fileadmin/newfile.jpg was copied to "uploads/[some-dir]/newfile_01.jpg". The filename was appended with "_01" because another file with the name "newfile.jpg" already existed in the location.
first.jpg,second.jpg	first.jpg	first.jpg	"uploads/[some-dir]/second.jpg" was deleted from the location.

## ['columns']['*field name*']['config'] / TYPE: "none"

This type will just show the value of the field in the backend. The field is not editable.

Key	Datatype	Description
<b>type</b>	string	<i>[Must be set to "none"]</i>
<b>pass_content</b>	boolean	If set, then content from the field is directly outputted in the <div> section. Otherwise the content will be passed through htmlspecialchars() and possibly nl2br() if there is configuration for rows. Be careful to set this flag since it allows HTML from the field to be outputted on the page, thereby creating the possibility of XSS security holes.
<b>rows</b>	integer	If this value is greater than 1 the display of the non-editable content will be shown in a <div> area trying to simulate the rows/columns known from a "text" type element.
<b>cols</b>	integer	See "rows" and "size"
<b>fixedRows</b>	boolean	If this is set the <div> element will not automatically try to fit the content length but rather respect the size selected by the value of the "rows" key.
<b>size</b>	integer	If rows is less than one, the "cols" value is used to set the width of the field and if "cols" is not found, then "size" is used to set the width. The measurements corresponds to those of "input" and "text" type fields.

## ['columns']['*field name*']['config'] / TYPE: "passthrough"

Can be saved/updated through TCE but the value is not evaluated in any way and the field has no rendering in the TCEforms.

You can use this to send values directly to the database fields without any automatic evaluation. But still the update gets logged and the history/undo function will work with such values.

Since there is no rendering mode for this field type it is specifically fitted for direct API usage with the TCEmain class.

Key	Datatype	Description
<b>type</b>	string	<i>[Must be set to "passthrough"]</i>

### Example:

This field is found in a number of table, e.g. the "pages" table. It is used by the system extension "impexp" to store some information.

```
'tx_impexp_origuid' => array('config' => array('type' => 'passthrough'))
```

## ['columns']['*field name*']['config'] / TYPE: "user"

Allows you to render a whole form field by a user function or class method.

Key	Datatype	Description
<b>type</b>	string	<i>[Must be set to "user"]</i>

Key	Datatype	Description
<b>userFunc</b>	string	<p>Function or method reference.</p> <p>If you want to call a function, just enter the function name. The function name must be prefixed "user_" or "tx_".</p> <p>If you want to call a method in a class, enter "[classname]-&gt;[methodname]". The class name must be prefixed "user_" or "tx_".</p> <p>Two arguments will be passed to the function/method: The first argument is an array (passed by reference) which contains the current information about the current field being rendered. The second argument is a reference to the parent object (an instance of the t3lib_TCEforms class).</p> <p>The array with the current information will contain any parameters declared with the "parameters" property described below.</p> <p><b>Notice:</b> The class must be registered with the TYPO3 autoloader. It's also possible to include it manually, but using the autoloader is the preferred way.</p>
<b>parameters</b>	array	<p>Array that will be passed as is to the userFunc as the "parameters" key of the first argument received by the user function.</p> <p>See example below.</p>
<b>noTableWrapping</b>	boolean	<p>If set, then the output from the user function will <i>not</i> be wrapped in the usual table - you will have to do that yourself.</p>

**Example:**

This field is rendered by custom PHP code:



The configuration in TCA is as simple as this:

```

'tx_examples_special' => array (
    'exclude' => 0,
    'label' => 'LLL:EXT:examples/locallang_db.xml:fe_users.tx_examples_special',
    'config' => array (
        'type' => 'user',
        'size' => '30',
        'userFunc' => 'EXT:examples/class.tx_examples_tca.php:tx_examples_tca->specialField',
        'parameters' => array(
            'color' => 'blue'
        )
    )
),

```

This is how the corresponding PHP class looks like:

```

class tx_examples_tca {
    function specialField($PA, $fObj) {
        $color = (isset($PA['parameters']['color'])) ? $PA['parameters']['color'] : 'red';
        $formField = '<div style="padding: 5px; background-color: ' . $color . ';">';
        $formField .= '<input type="text" name="' . $PA['itemFormElName'] . '"';
        $formField .= ' value="' . htmlspecialchars($PA['itemFormElValue']) . '"';
        $formField .= ' onchange="' . htmlspecialchars(implode(' ',
        $PA['fieldChangeFunc'])) . '"';
        $formField .= $PA['onFocus'];
        $formField .= '</div>';
        return $formField;
    }
}

```

This is not the place to dig into more details about user-defined forms. By this example you can start yourself up but you will have to figure out by yourself what options are available in the \$PA array and how to use them.

Note in particular how the "parameters" array declared in the TCA configuration can be retrieved as part of the first argument (\$PA) received by the method invoked.



## ['columns']['field name']['config'] / TYPE: "flex"

Rendering a FlexForm element - essentially this consists of a hierarchically organized set of fields which will have their values saved into a single field in the database, stored as XML.

Key	Datatype	Description
<b>type</b>	string	[Must be set to "flex"]
<b>ds_pointerField</b>	string	Field name(s) in the record which point to the field where the key for "ds" is found. Up to two field names can be specified comma separated.
<b>ds</b>	array	<p>Data Structure(s) defined in an array.</p> <p>Each key is a value that can be pointed to by "ds_pointerField". Default key is "default" which is what you should use if you do not have a "ds_pointerField" value of course.</p> <p>If you specified more than one ds_pointerField, the keys in this "ds" array should contain comma-separated value pairs where the asterisk * matches all values (see the example below). If you don't need to switch for the second ds_pointerField, it's also possible to use only the first ds_pointerField's value as a key in the "ds" array without necessarily suffixing it with ",*" for a catch-all on the second ds_pointerField.</p> <p>For each value in the array there are two options:</p> <ul style="list-style-type: none"> <li>• Either enter XML directly</li> <li>• Make a reference to an external XML file</li> </ul> <p><b>Example with XML directly entered:</b></p> <pre>'config' =&gt; array(     'type' =&gt; 'flex',     'ds_pointerField' =&gt; 'list_type',     'ds' =&gt; array(         'default' =&gt; '             &lt;T3DataStructure&gt;                 &lt;ROOT&gt;                     &lt;type&gt;array&lt;/type&gt;                     &lt;el&gt;                         &lt;xmlTitle&gt;                             &lt;TCEforms&gt;                                 &lt;label&gt;The Title:&lt;/label&gt;                                 &lt;config&gt;                                     &lt;type&gt;input&lt;/type&gt;                                     &lt;size&gt;48&lt;/size&gt;                                 &lt;/config&gt;                             &lt;/TCEforms&gt;                         &lt;/xmlTitle&gt;                     &lt;/el&gt;                 &lt;/ROOT&gt;             &lt;/T3DataStructure&gt;         '     ) )</pre> <p><b>Example with XML in external file:</b> (File reference is relative)</p> <pre>'config' =&gt; array(     'type' =&gt; 'flex',     'ds_pointerField' =&gt; 'list_type',     'ds' =&gt; array(         'default' =&gt; 'FILE:EXT:mininews/flexform_ds.xml'     ) )</pre> <p><b>Example using two ds_pointerFields</b> (as used for tt_content.pi_flexform since TYPO3 4.2.0):</p> <pre>'config' =&gt; array(     'type' =&gt; 'flex',     'ds_pointerField' =&gt; 'list_type,CType',     'ds' =&gt; array( </pre>

Key	Datatype	Description
		<pre>         'default' =&gt; 'FILE:...',         'tt_address_pil,list' =&gt; 'FILE:EXT:tt_address/pil/flexform.xml', // DS for list_type=tt_address_pil and CType=list         '*,table' =&gt; 'FILE:EXT:css_styled_content/flexform_ds.xml', // DS for CType=table, no matter which list_type value         'tx_myext_pil' =&gt; 'FILE:EXT:myext/flexform.xml', // DS for list_type=tx_myext_pil without specifying a CType at all     )     ) </pre>
<b>ds_tableField</b>	string	<p>Contains the value “[table]:[field name]” from which to fetch Data Structure XML.</p> <p>“ds_pointerField” is in this case the pointer which should contain the uid of a record from that table.</p> <p>This is used by TemplaVoila extension for instance where a field in the tt_content table points to a TemplaVoila Data Structure record:</p> <pre> 'tx_templavoila_flex' =&gt; array(     'exclude' =&gt; 1,     'label' =&gt; '...',     'displayCond' =&gt; 'FIELD:tx_templavoila_ds:REQ:true' ,     'config' =&gt; array(         'type' =&gt; 'flex',         'ds_pointerField' =&gt; 'tx_templavoila_ds',         'ds_tableField' =&gt; 'tx_templavoila_datastructure:dataprot',     ) ), </pre>
<b>ds_pointerField_search Parent</b>	string	<p>Used to search for Data Structure recursively back in the table assuming that the table is a tree table. This value points to the “pid” field. See “templavoila” for example - uses this for the Page Template.</p>
<b>ds_pointerField_search Parent_subField</b>	string	<p>Points to a field in the “rootline” which may contain a pointer to the “next-level” template. See “templavoila” for example - uses this for the Page Template.</p>

## Pointing to a Data Structure

Basically the configuration for a FlexForm field is all about pointing to the Data Structure which will contain form rendering information in the application specific tag “<TCEforms>”.

For general information about the backbone of a Data Structure, please see the <T3DataStructure> chapter in the Data Formats section.

## FlexForm facts

FlexForms create a form-in-a-form. The content coming from this form is still stored in the associated database field - but as an XML structure (stored by t3lib\_div::array2xml())!

The “TCA” information needed to generate the FlexForm fields are found inside a <T3DataStructure> XML document. When you configure a FlexForm field in a Data Structure (DS) you can use basically all column types documented here for TCA. The limitations are:

- “unique” and “uniqueInPid” evaluation is not available
- You cannot nest FlexForm configurations inside of FlexForms.
- Charset follows that of the current backend (that is “forceCharset” or the backend users language selection)

## <T3DataStructure> extensions for “<TCEforms>”

For FlexForms the DS is extended with a tag, “<TCEforms>” which define all settings specific to the

FlexForms usage.

Also a few meta tag features are used.

Sometimes it may be necessary to reload flexform if content of the field in the flexform is changed. This is accomplished by adding “<onChange>reload</onChange>” inside <TCEforms>. A typical example for that is a field that defines operational modes for an extension. When the mode changes, a flexform may need to show a new set of fields. By combining the <onChange> tag for mode fields with <displayCond> tag for other fields, it is possible to create truly dynamic flexforms.

Notice that changing the mode does not delete hidden field values of the flexform. Always use the “mode” field to determine which parameters to use.

The tables below document the extension elements:

“Array” Elements:

Element	Description	Child elements
<meta>	Can contain application specific meta settings. For FlexForms this means a definition of how languages are handled in the form.	<langChildren> <langDisable>
<[application tag]>	In this case the application tag is “<TCEforms>”	<i>A direct reflection of a ['columns'] ['field name']['config'] PHP array configuring a field in TCA. As XML this is expressed by array2xml()'s output. See example below.</i>
<ROOT><TCEforms>	For <ROOT> elements in the DS you can add application specific information about the sheet that the <ROOT> element represents.	<sheetTitle> <sheetDescription> <sheetShortDescr>

“Value” Elements:

Element	Format	Description
<langDisable>	boolean, 0/1	If set, then handling of localizations is disabled. Otherwise FlexForms will allow editing of additional languages than the default according to “sys_languages” table contents. The language you can select from is the language configured in “sys_languages” but they <i>must</i> have ISO country codes set - see example below.
<langChildren>	boolean, 0/1	If set, then localizations are bound to the default values 1-1 (“value” level). Otherwise localizations are handled on “structure level”
<sheetTitle>	string or LLL reference	Specifies the title of the sheet.
<sheetDescription>	string or LLL reference	Specifies a description for the sheet shown in the flexform.
<sheetShortDescr>	string or LLL reference	Specifies a short description of the sheet used in the tab-menu.

## Sheets and FlexForms

FlexForms always resolve sheet definitions in a Data Structure. If only one sheet is defined that must be the “sDEF” sheet (default). In that case no tab-menu for sheets will appear (see examples below).

## FlexForm data format, <T3FlexForms>

When saving FlexForm elements the content is stored as XML using `t3lib_div::array2xml()` to convert the internal PHP array to XML format. The structure is as follows:

“Array” Elements:

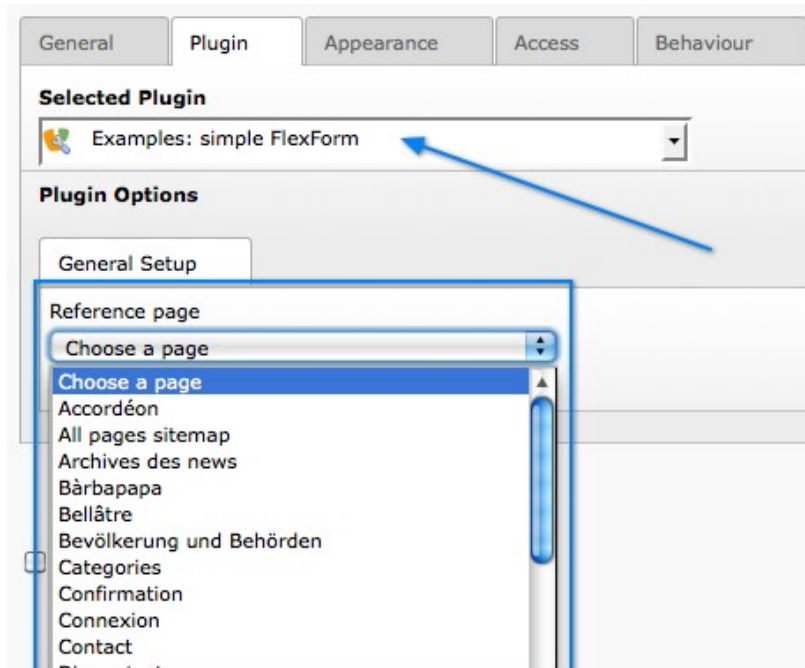
Element	Description	Child elements
<b>&lt;T3FlexForms&gt;</b>	Document tag	<b>&lt;meta&gt;</b> <b>&lt;data&gt;</b>
<b>&lt;meta&gt;</b>	Meta data for the content. For instance information about which sheet is active etc.	<b>&lt;currentSheetId&gt;</b> <b>&lt;currentLangId&gt;</b>
<b>&lt;data&gt;</b>	Contains the data; sheets, language sections, field and values	<b>&lt;sheet&gt;</b>
<b>&lt;sheet&gt;</b>	Contains the data for each sheet in the form. If there are no sheets, the default sheet “<sDEF>” is always used.	<b>&lt;sDEF&gt;</b> <b>&lt;s_[sheet keys]&gt;</b>
<b>&lt;sDEF&gt;</b> <b>&lt;[sheet keys]&gt;</b>	For each sheet it contains elements for each language. If <b>&lt;meta&gt;&lt;langChildren&gt;</b> is false then all languages are stored on this level, otherwise only the <b>&lt;IDEF&gt;</b> tag is used.	<b>&lt;IDEF&gt;</b> <b>&lt;[ISO language code]&gt;</b>
<b>&lt;IDEF&gt;</b> <b>&lt;[language keys]&gt;</b>	For each language the fields in the form will be available on this level.	<b>&lt;[field name]&gt;</b>
<b>&lt;[field name]&gt;</b>	For each field name there is at least one element with the value, <b>&lt;vDEF&gt;</b> . If <b>&lt;meta&gt;&lt;langChildren&gt;</b> is true then there will be a <b>&lt;v*&gt;</b> tag for each language holding localized values.	<b>&lt;vDEF&gt;</b> <b>&lt;v[ISO language code]&gt;</b>
<b>&lt;currentLangId&gt;</b>	Numerical array of language ISO codes + “DEF” for default which are currently displayed for editing.	<b>&lt;n[0-x]&gt;</b>

#### “Value” Elements:

Element	Format	Description
<b>&lt;vDEF&gt;</b> <b>&lt;v[ISO language code]&gt;</b>	string	Content of the field in default or localized versions
<b>&lt;currentSheetId&gt;</b>	string	Points to the currently shown sheet in the DS.

#### Example: Simple FlexForm

The extension “examples” provides some sample FlexForms. The “simple FlexForm” plugin provides a very basic configuration with just a select-type field to choose a page from the “pages” table.



The DS used to render this field is found in the file “flexform\_dsl.xml” inside the “examples” extension. Notice the **<TCEforms>** tags:

```
<T3DataStructure>
  <meta>
    <langDisable>1</langDisable>
```

```

</meta>
<sheets>
  <sDEF>
    <ROOT>
      <TCEforms>
        <sheetTitle>LLL:EXT:examples/locallang_db.xml:
examples.pi_flexform.sheetGeneral</sheetTitle>
      </TCEforms>
      <type>array</type>
    </el>
    <pageSelector>
      <TCEforms>
        <label>LLL:EXT:examples/locallang_db.xml:
examples.pi_flexform.pageSelector</label>
      <config>
        <type>select</type>
        <items type="array">
          <numIndex index="0" type="array">
            <numIndex
index="0">LLL:EXT:examples/locallang_db.xml:examples.pi_flexform.choosePage</numIndex>
            <numIndex index="1">0</numIndex>
          </numIndex>
        </items>
        <foreign_table>pages</foreign_table>
        <foreign_table_where>ORDER BY
title</foreign_table_where>
        <minitems>0</minitems>
        <maxitems>1</maxitems>
      </config>
    </TCEforms>
  </pageSelector>
</el>
</ROOT>
</sDEF>
</sheets>
</T3DataStructure>

```

It's clear that the contents of <TCEforms> is a direct reflection of the field configurations we normally set up in the \$TCA array.

The Data Structure for this FlexForm is loaded in the “pi\_flexform” field of the “tt\_content” table by adding the following to the ext\_tables.php file of the “examples” extension:

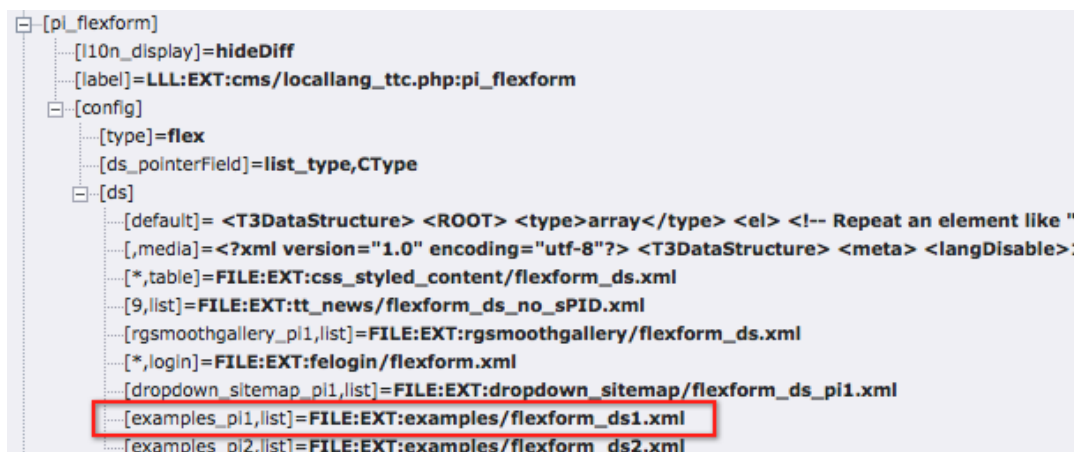
```

$TCA['tt_content']['types']['list']['subtypes_addlist'][$_EXTKEY . '_pi1'] = 'pi_flexform';
t3lib_extMgm::addPiFlexFormValue($_EXTKEY . '_pi1', 'FILE:EXT:examples/flexform_ds1.xml');

```

In the first line the tt\_content field “pi\_flexform” is added to the display of fields when the Plugin type is selected and set to “examples\_pi1”. In the second line the DS xml file is configured to be the source of the FlexForm DS used.

If we browse the definition for the “pi\_flexform” field in “tt\_content” using the Admin > Configuration module, we can see the following:



As you can see there are quite a few extensions that have added pointers to their Data Structures. Towards the bottom we can find the one we have just been looking at.

**Example: FlexForm with two sheets**

In this example we create a FlexForm field with two “sheets”. Each sheet can contain a separate FlexForm structure. We build it up on top of the previous example, so the first sheet still has a select-type field related to the “pages” table. In the second sheet, we add a simple input field and a text field.

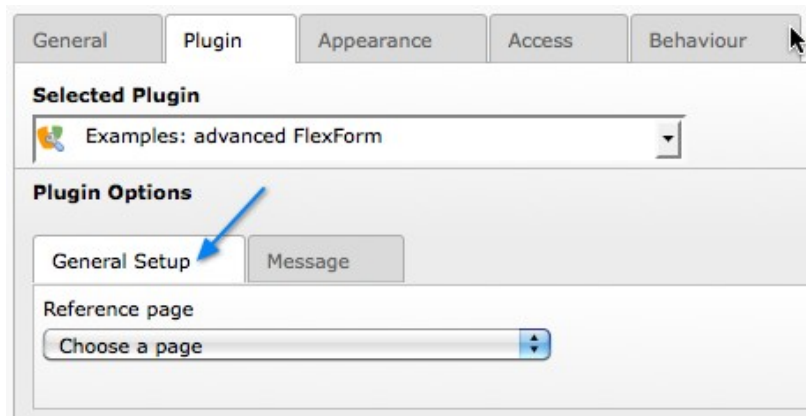
```

<T3DataStructure>
  <meta>
    <langDisable>1</langDisable>
  </meta>
  <sheets>
    <sDEF>
      <ROOT>
        <TCEforms>
          <sheetTitle>LLL:EXT:examples/locallang_db.xml:
examples.pi_flexform.sheetGeneral</sheetTitle>
        </TCEforms>
        <type>array</type>
        <el>
          <pageSelector>
            <TCEforms>
              <label>LLL:EXT:examples/locallang_db.xml:
examples.pi_flexform.pageSelector</label>
              <config>
                <type>select</type>
                <items type="array">
                  <numIndex index="0" type="array">
                    <numIndex
index="0">LLL:EXT:examples/locallang_db.xml:examples.pi_flexform.choosePage</numIndex>
                    <numIndex index="1">0</numIndex>
                  </numIndex>
                </items>
                <foreign_table>pages</foreign_table>
                <foreign_table_where>ORDER BY
title</foreign_table_where>
                <minitems>0</minitems>
                <maxitems>1</maxitems>
              </config>
            </TCEforms>
          </pageSelector>
        </el>
      </ROOT>
    </sDEF>
    <s_Message>
      <ROOT>
        <TCEforms>
          <sheetTitle>LLL:EXT:
examples/locallang_db.xml:examples.pi_flexform.s_Message</sheetTitle>
        </TCEforms>
        <type>array</type>
        <el>
          <header>
            <TCEforms>
              <label>LLL:EXT:
examples/locallang_db.xml:examples.pi_flexform.header</label>
              <config>
                <type>input</type>
                <size>30</size>
              </config>
            </TCEforms>
          </header>
          <message>
            <TCEforms>
              <label>LLL:EXT:
examples/locallang_db.xml:examples.pi_flexform.message</label>
              <config>
                <type>text</type>
                <cols>40</cols>
                <rows>5</rows>
              </config>
            </TCEforms>
          </message>
        </el>
      </ROOT>
    </s_Message>
  </sheets>

```

</T3DataStructure>

The part that is different from the first Data Structure is highlighted in bold. The result from this configuration is a form which looks like this:



The screenshot shows a configuration window with five tabs: General, Plugin, Appearance, Access, and Behaviour. The 'Plugin' tab is selected. Below the tabs, there is a section titled 'Selected Plugin' with a dropdown menu showing 'Examples: advanced FlexForm'. Below this is a section titled 'Plugin Options' containing two sub-tabs: 'General Setup' and 'Message'. The 'General Setup' sub-tab is selected and highlighted with a blue arrow. Below the sub-tabs, there is a section titled 'Reference page' with a dropdown menu showing 'Choose a page'.

This looks very much like the first example, but notice the second tab. Clicking on “Message”, we can access the second sheet which shows some other fields:

The screenshot shows a configuration window with tabs: General, Plugin, Appearance, Access, and Behaviour. The 'Plugin' tab is active. Under 'Selected Plugin', a dropdown menu shows 'Examples: advanced FlexForm'. Below this, 'Plugin Options' are shown with two sub-tabs: 'General Setup' and 'Message'. A blue arrow points to the 'Message' sub-tab. The 'Message' sub-tab contains two text input fields: 'Header' and 'Message'.

This screenshot is similar to the one above but with a different styling. It shows the same configuration window with the 'Plugin' tab active. The 'Message' sub-tab under 'Plugin Options' is selected, indicated by a red arrow. The 'Header' and 'Message' text input fields are visible.

If you look at the XML stored in the database field “pi\_flexform” this is how it looks:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<T3FlexForms>
  <data>
    <sheet index="sDEF">
      <language index="lDEF">
        <field index="pageSelector">
          <value index="vDEF">9</value>
        </field>
      </language>
    </sheet>
    <sheet index="s_Message">
      <language index="lDEF">
        <field index="header">
          <value index="vDEF">My Header</value>
        </field>
        <field index="message">
          <value index="vDEF">And my message.
On several lines.</value>
        </field>
      </language>
    </sheet>
  </data>
```



```
</T3FlexForms>
```

Notice how the data of the two sheets are separated (sheet names highlighted in bold above).

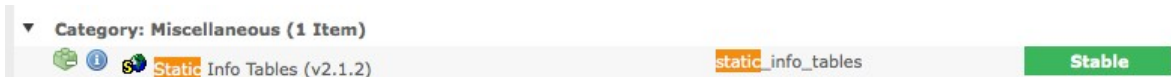
### Example: Rich Text Editor in FlexForms

Creating a RTE in FlexForms is done by adding “defaultExtras” content to the <TCEforms> tag:

```
<TCEforms>
  <config>
    <type>text</type>
    <cols>48</cols>
    <rows>5</rows>
  </config>
  <label>Subtitle</label>
  <defaultExtras>richtext[*]:rte_transform[mode=ts_css]</defaultExtras>
</TCEforms>
```

## Handling languages in FlexForms

FlexForms allows you to handle translations of content in two ways. But before you can enable those features you have to install the extension “static\_info\_tables” which contains country names and ISO-language codes which are the ones by which FlexForms stores localized content:



Then you must configure languages in the database:



And finally, you have to make sure that each of these languages points to the right ISO code:



By default, you will not see any changes. Indeed if you look at the example XML displayed above, you will notice the following line, at the top, in the “meta” section:

```
<langDisable>1</langDisable>
```

This means that translation of the FlexForm is disabled. In the example above, the FlexForm is part of a content element. That content element can still be translated as usual. What we're going to look at below is how a FlexForm field may end up containing its own translations. There are two methods for this.

### Localization method #1

The first localization method just requires to change the “langDisable” flag mentioned above to 0:

```
<langDisable>0</langDisable>
```

This means that translations are now allowed for that FlexForm. This is how it looks like:

The screenshot displays the 'Selected Plugin' section with a dropdown menu showing 'Examples: advanced FlexForm with localization type 1'. Below this, the 'Plugin Options' section is shown for three languages: DEF (French), DE (German), and EN (English). Each language section contains two tabs: 'General Setup' and 'Message'. The 'Message' tab is selected for each language. Each 'Message' tab contains a 'Header' field and a 'Message' field, both of which are empty text boxes.

The data XML in the data base looks like this:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<T3FlexForms>
  <data>
    <sheet index="sDEF">
      <language index="lDEF">
        <field index="pageSelector">
          <value index="vDEF">9</value>
        </field>
      </language>
      <language index="lDE">
        <field index="pageSelector">
          <value index="vDEF"></value>
        </field>
      </language>
    </sheet>
  </data>
</T3FlexForms>
```

```

        <language index="LEN">
            <field index="pageSelector">
                <value index="vDEF"></value>
            </field>
        </language>
    </sheet>
    <sheet index="s_Message">
        <language index="LDEF">
            <field index="header">
                <value index="vDEF">My Header</value>
            </field>
            <field index="message">
                <value index="vDEF">And my message.
On several lines.</value>
            </field>
        </language>
        <language index="LDE">
            <field index="header">
                <value index="vDEF">Hallo!</value>
            </field>
            <field index="message">
                <value index="vDEF">Das is auf Deutsch!</value>
            </field>
        </language>
        <language index="LEN">
            <field index="header">
                <value index="vDEF"></value>
            </field>
            <field index="message">
                <value index="vDEF"></value>
            </field>
        </language>
    </sheet>
</data>
</T3FlexForms>

```

Note how each language is stored separately at a level above the “field” level. Each language tag carries an attribute identifying the language like “LDE” or “LEN”.

### Localization method #2

In the first method of localization each language can potentially contain a differently structured data set. This is possible because as soon as a DS defines sections with array objects inside the number of objects can be individual!

The second method of localization handles each language on the *value* level instead, thus requiring a translation for each and every field in the default language! You enable this by setting the “langChildren” tag to “1” in the “meta” section:

```

<meta>
    <langDisable>0</langDisable>
    <langChildren>1</langChildren>
</meta>

```

The editing form will now look like this:

**Selected Plugin**

Examples: advanced FlexForm with localization type 2

**Plugin Options**

General Setup | Message

**Header**

[Text Field]

Header

[Text Field]

Header

[Text Field]

**Message**

[Text Area]

Message

[Text Area]

Message

[Text Area]

You can see how all translation fields for the “Header” are grouped together with the default header. Likewise for the “Message” field.

The difference is also seen in the <T3FlexForms> content:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<T3FlexForms>
  <data>
    <sheet index="sDEF">
      <language index="LDEF">
        <field index="pageSelector">
          <value index="vDEF"></value>
          <value index="vDE"></value>
          <value index="vEN"></value>
          <value index="vDE.vDEFbase"></value>
          <value index="vEN.vDEFbase"></value>
        </field>
      </language>
    </sheet>
    <sheet index="s_Message">
      <language index="LDEF">
        <field index="header">
```

```

        <value index="vDEF">My header</value>
        <value index="vDE">Hallo!</value>
        <value index="vEN"></value>
        <value index="vDE.vDEFbase">My header</value>
        <value index="vEN.vDEFbase">My header</value>
    </field>
    <field index="message">
        <value index="vDEF">And my message.

On several lines.</value>
        <value index="vDE">Das is auf Deutsch!</value>
        <value index="vEN"></value>
        <value index="vDE.vDEFbase">And my message.

On several lines.</value>
        <value index="vEN.vDEFbase">And my message.

On several lines.</value>
    </field>
</language>
</sheet>
</data>
</T3FlexForms>

```

In this case, there's only on "language" tag per sheet and all values are repeated with a language index attribute to tell them apart.

The additional "value" tags with an index attribute like "vDE.vDEFbase" are used to store the previous value that the field contained, so that a translation diff view can be displayed:

**NOTICE:** The two localization methods are NOT compatible! You cannot suddenly change from the one method to the other without having to do some conversion of the data format. That is obvious when you look at how the two methods also require different data structures underneath!

## ['columns']['field name']['config'] / TYPE: "inline"

Inline-Relational-Record-Editing (IRRE) offers a way of directly editing parent-child-relations in one backend view. New child records are created using AJAX calls to prevent a reload of the complete backend view. This type was first integrated in TYPO3 4.1.

**Edit 1:n CSV: Hotels "Hotel California" on page "Generated content"**

**Language:**  
Default

**Hide:**  
☐

**Title:**  
Hotel California

**Offers:**  
Create new

Single room

**Language:**  
Default

**Hide:**  
☐

**Title:**  
Single room

**Prices:**  
Create new

Per night

Please note that IRRE does not fully work in conjunction with versioning. Only 1:n relationships are supported in workspaces (since TYPO3 4.5).



### Note

TCAdefaults.<table>.pid = <page id> can be used to define the pid of new child records. Thus, it's possible to have special storage folders on a per-table-basis.

Key	Datatype	Description	Scope
<b>type</b>	string	[Must be set to "inline"]	Display / Proc.
<b>foreign_table</b>	string (table name)	[Must be set, there is no type "inline" without a foreign table] The table name of the child records is defined here. The table must be configured in \$TCA. See the other related options below.	Display / Proc.
<b>appearance</b>	array	Has information about the appearance of child-records, namely: <ul style="list-style-type: none"> <li>collapseAll (boolean) Show all child-records collapsed (if false, all are expanded)</li> </ul>	Display

Key	Datatype	Description	Scope
		<ul style="list-style-type: none"> <li>• <i>expandSingle</i> (boolean) Show only one child-record expanded each time. If a collapsed record is clicked, the currently open one collapses and the clicked one expands.</li> <li>• <i>newRecordLinkAddTitle</i> (boolean) Adds the title of the foreign_table to the “New record” link. false: “Create new” true: “Create new &lt;title of foreign_table&gt;”, e.g. “Create new address”</li> <li>• <i>newRecordLinkPosition</i> (string) <b>Deprecated:</b> use <i>levelLinksPosition</i> instead</li> <li>• <i>levelLinksPosition</i> (string) Values: 'top', 'bottom', 'both', 'none' – default: 'top' Defines where to show the “New record” link in relation to the child records.</li> <li>• <i>useCombination</i> (boolean) This is only useful on bidirectional relations using an intermediate table with attributes. In a “combination” it is possible to edit the attributes AND the related child record itself. If using a foreign_selector in such a case, the foreign_unique property <b>must</b> be set to the same field as the foreign_selector.</li> <li>• <i>useSortable</i> (boolean) Active Drag&amp;Drop Sorting by the script.aculo.us Sortable object.</li> <li>• <i>showPossibleLocalizationRecords</i> (boolean) Show unlocalized records which are in the original language, but not yet localized.</li> <li>• <i>showRemovedLocalizationRecords</i> (boolean) Show records which were once localized but do not exist in the original language anymore.</li> <li>• <i>showAllLocalizationLink</i> (boolean) Defines whether to show the "localize all records" link to fetch untranslated records from the original language.</li> <li>• <i>showSynchronizationLink</i> (boolean) Defines whether to show a "synchronize" link to update to a 1:1 translation with the original language.</li> <li>• <i>enabledControls</i> (array) Associative array with the keys 'info', 'new', 'dragdrop', 'sort', 'hide', 'delete', 'localize'. If the accordant values are set to a boolean value (true or false), the control is shown or hidden in the header of each record.</li> </ul>	
<b>behaviour</b>	array	<p>Has information about the behavior of child-records, namely:</p> <ul style="list-style-type: none"> <li>• <i>localizationMode</i> ('keep', 'select') Defines in general whether children are really localizable (set to 'select') or just taken from the default language (set to 'keep'). If this property is not set, but the affected parent and child tables were localizable, the mode 'select' is used by default. <ul style="list-style-type: none"> <li>• Mode 'keep': This is not a real localization, since the children are taken from the parent of the original language. But the children can be moved, deleted, modified etc. on the localized parent which - of course - also affects the original language.</li> <li>• Mode 'select': This mode provides the possibility to have a selective localization and to compare localized data to the pendants of the original language. Furthermore this mode is extended by a 'localize all' feature, which works similar to the localization of content on pages, and a 'synchronize' feature which offers the possibility to synchronize a</li> </ul> </li> </ul>	Display / Proc.

Key	Datatype	Description	Scope
		<p>localization with its original language.</p> <ul style="list-style-type: none"> <li><i>localizeChildrenAtParentLocalization</i> (boolean) Defines whether children should be localized when the localization of the parent gets created.</li> <li><i>disableMovingChildrenWithParent</i> (boolean) Disables that child records get moved along with their parent records.</li> </ul>	
<b>foreign_field</b>	string	The <i>foreign_field</i> is the field of the child record pointing to the parent record. This defines where to store the uid of the parent record.	Display / Proc.
<b>foreign_label</b>	string	If set, it overrides the label set in <code>\$TCA[&lt;foreign_table&gt;]['ctrl']['label']</code> for the inline-view.	Display / Proc.
<b>foreign_selector</b>	string	<p>A selector is used to show all possible child records that could be used to create a relation with the parent record. It will be rendered as a multi-select-box. On clicking on an item inside the selector a new relation is created.</p> <p>The <i>foreign_selector</i> points to a field of the <i>foreign_table</i> that is responsible for providing a selector-box – this field on the <i>foreign_table</i> usually has the type “select” and also has a “<i>foreign_table</i>” defined.</p>	Display / Proc.
<b>foreign_sortby</b>	string	Define a field on the child record (or on the intermediate table) that stores the manual sorting information. It is possible to have a different sorting, depending from which side of the relation we look at parent or child.	Display / Proc.
<b>foreign_default_sortby</b>	string	If a field name for <i>foreign_sortby</i> is defined, then this is ignored. Otherwise this is used as the “ORDER BY” statement to sort the records in the table when listed.	Display
<b>foreign_table_field</b>	string	<p>The <i>foreign_table_field</i> is the field of the child record pointing to the parent record. This defines where to store the table name of the parent record. On setting this configuration key together with <i>foreign_field</i>, the child record knows what its parent record is – so the child record could also be used on other parent tables.</p> <p>This issue is also known as “weak entity”.</p> <p>Do not confuse with <i>foreign_table</i> or <i>foreign_field</i>. It has its own behavior.</p>	Display / Proc.
<b>foreign_unique</b>	string	Field which must be unique for all children of a parent record. Example: Say you have two tables, products, your parent table, and prices, your child table (products) can have multiple prices. The prices table has a field called customer_group, which is a selector box. Now you want to be able to specify prices for each customer group when you edit a product, but of course you don't want to specify contradicting prices for one product (i.e. two different prices for the same customer_group). That's why you would set <i>foreign_unique</i> to the field name “customer_group”, to prevent that two prices for the same customer group can be created for one product.	Display / Proc.
<b>MM</b>	string (table name)	<p>Means that the relation to the records of “foreign_table” is done with a M-M relation with a third “join” table.</p> <p>That table typically has three columns:</p> <ul style="list-style-type: none"> <li><i>uid_local</i>, <i>uid_foreign</i> for uids respectively.</li> <li><i>sorting</i> is a required field used for ordering the items.</li> </ul> <p>The field which is configured as “inline” is not used for data-storage any more but rather it's set to the number of records in the relation on each update, so the field should be an integer.</p> <p>Notice: Using MM relations you can ONLY store real relations for foreign tables in the list - no additional string values or non-record values (so no attributes).</p>	Proc.
<b>size</b>	integer	Height of the selector box in TCEforms.	Display
<b>autoSizeMax</b>	integer	If set, then the height of multiple-item selector boxes (maxitem > 1) will automatically be adjusted to the number of selected elements,	Display



Key	Datatype	Description	Scope
		however never less than "size" and never larger than the integer value of "autoSizeMax" itself (takes precedence over "size"). So "autoSizeMax" is the maximum height the selector can ever reach.	
<b>maxitems</b>	integer > 0	Maximum number of items in the selector box. Defaults to 100000. Note that this is different from types "select" and "group" which default to 1.	Display / Proc
<b>minitems</b>	integer > 0	Minimum number of items in the selector box. (Default = 0)	Display
<b>symmetric_field</b>	string	This works like foreign_field, but in case of using bidirectional symmetric relations. symmetric_field defines in which field on the foreign_table the uid of the "other" parent is stored.	Display / Proc.
<b>symmetric_label</b>	string	If set, it overrides the label set in \$TCA[<foreign_table>]['ctrl'] ['label'] for the inline-view and only if looking to a symmetric relation from the "other" side.	Display / Proc.
<b>symmetric_sortby</b>	string	This works like foreign_sortby, but in case of using bidirectional symmetric relations. Each side of a symmetric relation could have its own sorting, so symmetric_sortby defines a field on the foreign_table where the sorting of the "other" side is stored.	Display / Proc.

**Example "comma-separated list":**

This combines companies with persons (employees) using a comma separated list, so no "foreign\_field" is used here.

```
$TCA['company'] = array(
    'ctrl' => ...,
    'interface' => ...,
    'feInterface' => ...,
    'columns' => array(
        'hidden' => ...,
        'employees' => array(
            'exclude' => 1,
            'label' => 'LLL:EXT:myextension/locallang_db.xml:company.employees',
            'config' => array(
                'type' => 'inline',
                'foreign_table' => 'person',
                'maxitems' => 10,
                'appearance' => array(
                    'collapseAll' => 1,
                    'expandSingle' => 1,
                ),
            ),
        ),
    ),
    'types' => ...
    'palettes' => ...
);
```

**Example "attributes on anti-symmetric intermediate table":**

This example combines companies with persons (employees) using an intermediate table. It is also possible to add attributes to every relation – in this example, an attribute "jobtype" on the "person\_company" table is defined. It is also possible to look at the relation from both sides (parent and child).

```
$TCA['person'] = array(
    'columns' => array(
        'employers' => array(
            'label' => 'LLL:EXT:myextension/locallang_db.xml:person.employers',
            'config' => array(
                'type' => 'inline',
                'foreign_table' => 'person_company',
                'foreign_field' => 'person',
                'foreign_label' => 'company',
            ),
        ),
    ),
);
```

```

$TCA['company'] = array(
    'columns' => array(
        'employees' => array(
            'label' => 'LLL:EXT:myextension/locallang_db.xml:company.employees',
            'config' => array(
                'type' => 'inline',
                'foreign_table' => 'person_company',
                'foreign_field' => 'company',
                'foreign_label' => 'person',
            ),
        ),
    ),
);

$TCA['person_company'] = array(
    'columns' => array(
        'person' => array(
            'label' => 'LLL:EXT:myextension/locallang_db.xml:person_company.person',
            'config' => array(
                'type' => 'select',
                'foreign_table' => 'person',
                'size' => 1,
                'minitems' => 0,
                'maxitems' => 1,
            ),
        ),
        'company' => array(
            'label' => 'LLL:EXT:myextension/locallang_db.xml:person_company.company',
            'config' => array(
                'type' => 'select',
                'foreign_table' => 'company',
                'size' => 1,
                'minitems' => 0,
                'maxitems' => 1,
            ),
        ),
        'jobtype' => array(
            'label' => 'LLL:EXT:myextension/locallang_db.xml:person_company.jobtype',
            'config' => array(
                'type' => 'select',
                'items' => array(
                    array('Project Manager (PM)', '0'),
                    array('Chief Executive Officer (CEO)', '1'),
                    array('Chief Technology Officer (CTO)', '2'),
                ),
                'size' => 1,
                'maxitems' => 1,
            ),
        ),
    ),
);

```

#### **Example “attributes on symmetric intermediate table”:**

This example combines two persons with each other – imagine they are married. One person on the first side is the husband, and one person on the other side is the wife (or generally “spouse” in the example below). Symmetric relations combine object of the same with each other and it does not depend, from which side someone is looking to the relation – so the husband knows it's wife and the wife also know it's husband.

Sorting could be individually defined for each of the both sides (perhaps this should not be applied to a wife-husband-relationship in real life).

```

$TCA['person'] = array(
    'columns' => array(
        'employers' => array(
            'label' => 'LLL:EXT:myextension/locallang_db.xml:person.employers',
            'config' => array(
                'type' => 'inline',
                'foreign_table' => 'person_symmetric',
                'foreign_field' => 'person',
                'foreign_sortby' => 'sorting_person',
                'foreign_label' => 'spouse',
                'symmetric_field' => 'spouse',
            ),
        ),
    ),
);

```

```
        'symmetric_sortby' => 'sorting_spouse',
        'symmetric_label' => 'person',
    ),
),
);

$TCA['person_symmetric'] = array(
    'columns' => array(
        'person' => array(
            'label' => 'LLL:EXT:myextension/locallang_db.xml:person_symmetric.person',
            'config' => array(
                'type' => 'select',
                'foreign_table' => 'person',
                'size' => 1,
                'minitems' => 0,
                'maxitems' => 1,
            ),
        ),
        'spouse' => array(
            'label' => 'LLL:EXT:myextension/locallang_db.xml:person_symmetric.spouse',
            'config' => array(
                'type' => 'select',
                'foreign_table' => 'person',
                'size' => 1,
                'minitems' => 0,
                'maxitems' => 1,
            ),
        ),
        'someattribute' => array(
            'label' => 'LLL:EXT:myextension/locallang_db.xml:person_symmetric.someattribute',
            'config' => array(
                'type' => 'input',
            ),
        ),
        'sorting_person' => array(
            'config' => array(
                'type' => 'passthrough',
            ),
        ),
        'sorting_spouse' => array(
            'config' => array(
                'type' => 'passthrough',
            ),
        ),
    ),
);
```

## ['types']['key] section

You have to add *at least* one entry in the "types"-configuration before any of the configured fields from the ['columns'] section will show up in TCEforms.

### Required configuration

Let's take the internal notes (sys\_note) as an example. The input form looks like this:

It corresponds to the following "types" configuration:

```
'types' => array(
    '0' => array('showitem' => 'category;;;2-2-2, author, email, personal, subject;;;3-3-3, message')
)
```

The key "showitem" lists the order in which to define the fields: "category, author, email, personal, subject, message".

### Optional possibilities

The power of the "types"-configuration becomes clear when you want the form composition of a record to depend on a value from the record. Let's look at the "dummy" table from the "examples" extension. The "ctrl" section of its TCA looks like this:

```
$TCA['tx_examples_dummy'] = array(
    'ctrl' => array(
        'title'      => 'LLL:EXT:examples/locallang_db.xml:tx_examples_dummy',
        'label'      => 'title',
        'tstamp'     => 'tstamp',
        'crdate'     => 'crdate',
        'cruser_id'  => 'cruser_id',
        'type'       => 'record_type',
        'default_sortby' => 'ORDER BY title',
        'delete'     => 'deleted',
        'enablecolumns' => array(
            'disabled' => 'hidden',
        ),
        'dynamicConfigFile' => t3lib_extMgm::extPath($EXTKEY) . 'tca.php',
        'iconfile'         => t3lib_extMgm::extRelPath($EXTKEY) .
            'icon_tx_examples_dummy.gif',
    ),
);
```

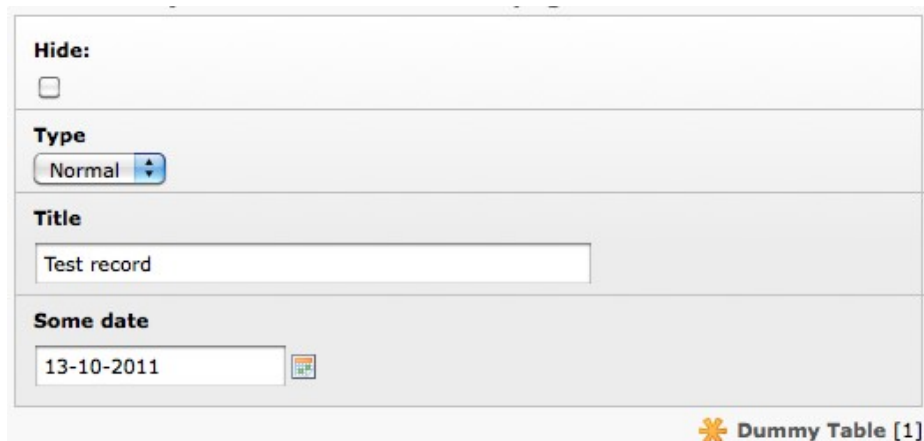
The line in bold indicates that the field called “record\_type” will be used to indicate the “type” of any given record of the table. Let's look at how this field is defined:

```
'record_type' => array(
    'exclude' => 0,
    'label' => 'LLL:EXT:examples/locallang_db.xml:tx_examples_dummy.record_type',
    'config' => array(
        'type' => 'select',
        'items' => array(
            array('LLL:EXT:examples/locallang_db.xml:tx_examples_dummy.record_type.0',
0),
            array('LLL:EXT:examples/locallang_db.xml:tx_examples_dummy.record_type.1',
1),
            array('LLL:EXT:examples/locallang_db.xml:tx_examples_dummy.record_type.2',
2),
        )
    )
),
```

There's nothing unusual here. It's a pretty straightforward select field, with three options. Finally, in the “types” section, we defined what fields should appear and in what order for every value of the “type” field:

```
'types' => array(
    '0' => array('showitem' => 'hidden, record_type, title, some_date '),
    '1' => array('showitem' => 'record_type, title'),
    '2' => array('showitem' => 'title, some_date, hidden, record_type '),
),
```

The result if the following display when type “Normal” is chosen:




Hide: ☐

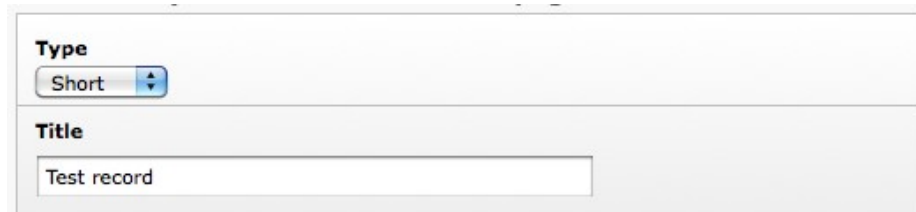
Type: Normal

Title: Test record

Some date: 13-10-2011

 Dummy Table [1]

Changing to type “Short” reloads the form and displays the following:



Type: Short

Title: Test record

And finally, type “Weird” also shows all fields, but in a different order:

Title

Test record

Some date

13-10-2011

Hide:

☐

Type

Weird

## Default values

If no "type" field is defined the type value will default to "0" (zero). If the type value (coming from a field or being zero by default) does not point to a defined index in the "types"-configuration, the configuration for key "1" will be used by default.

**Notice:** You must not show the same field more than once in the editing form. If you do, the field will not detect the value properly.

Key	Datatype	Description
<b>showitem</b>	string (list of field configuration sets)	<p><b>Required.</b> Configuration of the displayed order of fields in TCEforms. The whole string is divided by tokens according to a - unfortunately - complex ruleset.</p> <ul style="list-style-type: none"> <li>#1: Overall the value is divided by a "comma" ( , ). Each part represents the configuration for a single field.</li> <li>#2: Each of the field configurations is further divided by a semi-colon ( ; ). Each part of this division has a special significance. <ul style="list-style-type: none"> <li>Part 1: Field name reference (<b>Required!</b>)</li> <li>Part 2: Alternative field label (string or LLL reference)</li> <li>Part 3: Palette number (referring to an entry in the "palettes" section).</li> <li>Part 4: Special configuration (split by colon ( : )), e.g. 'nowrap' and 'richtext[(list of keys or *)]' (see “Additional \$TCA features”)</li> <li>Part 5: Form style codes (see “Visual style of TCEforms”)</li> </ul> </li> </ul> <p>Notice: Instead of a real field name you can insert "--div--" and you should have a divider line shown. However this is not rendered by default. If you set the <code>dividers2tabs</code> option (see [ctrl] section), each --div-- will define a new tab. Furthermore using a value "newline" for Part 3, will start a newline with this tab.</p> <p><b>Example:</b></p> <pre>'types' =&gt; array(     '0' =&gt; array('showitem' =&gt; 'hidden;;;1-1-1, title;;;2-2-2, poem, filename;;;3-3-3, season;;;4-4-4, weirdness, color, --div--;LLL:EXT:examples/locallang_db.xml:tx_examples_haiku.images, image1, image2, image3, image4, image5'), ),</pre> <p>Another special field name, '--palette--', will insert a link to a palette (of course you need to specify a palette and title then...)</p>
<b>subtype_value_field</b>	string (field name)	Field name, which holds a value being a key in the 'subtypes_excludelist' array. This is used to specify a secondary level of 'types' - basically hiding certain fields of those found in the types-configuration, based on the value of another field in the row.

Key	Datatype	Description
		<b>Example (from sysext/cms/tbl_tt_content.php):</b> <pre> 'subtype_value_field' =&gt; 'list_type', 'subtypes_excludelist' =&gt; array(     '3' =&gt; 'layout',     '2' =&gt; 'layout',     '5' =&gt; 'layout',     ...     '21' =&gt; 'layout' ), </pre>
<b>subtypes_excludelist</b>	array	See "subtype_value_field".  <b>Syntax:</b> "[field value]" => "[comma-separated list of fields (from the main types-config) which are excluded]"
<b>subtypes_addlist</b>	array	A list of fields to add when the "subtype_value_field" matches a key in this array.  See "subtype_value_field".  <b>Syntax:</b> "[value]" => "[ comma-separated list of fields which are added]"  <b>Notice:</b> that any transformation configuration used by TCE will NOT work because that configuration is visible for the TCEforms class only during the drawing of fields. In other words any configuration in this list of fields will work for display only."
<b>bitmask_value_field</b>	string (field name)	Field name, which holds a value being the integer (bit-mask) for the 'bitmask_excludelist_bits' array. It works much like 'subtype_value_field' but excludes fields based on whether a bit from the value field is set or not. See 'bitmask_excludelist_bits'; [+/-] indicates whether the bit [bit-number] is set or not.  <b>Example:</b> <pre> 'bitmask_value_field' =&gt; 'active', 'bitmask_excludelist_bits' =&gt; array(     '-0' =&gt; 'tpl_a_subpart_marker,tpl_a_description',     '-1' =&gt; 'tpl_b_subpart_marker,tpl_b_description',     '-2' =&gt; 'tpl_c_subpart_marker,tpl_c_description' ) </pre>
<b>bitmask_excludelist_bits</b>	array	See "bitmask_value_field"  "[+/-][bit-number]" => "[comma-separated list of fields (from the main types-config) excluded]"

## ['palettes'][key] section

"Palettes" represent a way to move less frequently used form fields out of sight. Palettes are groups of field which are associated with another field in the main form. When this field is activated the palette fields are displayed. In the backend, "palettes" are known as "secondary options".

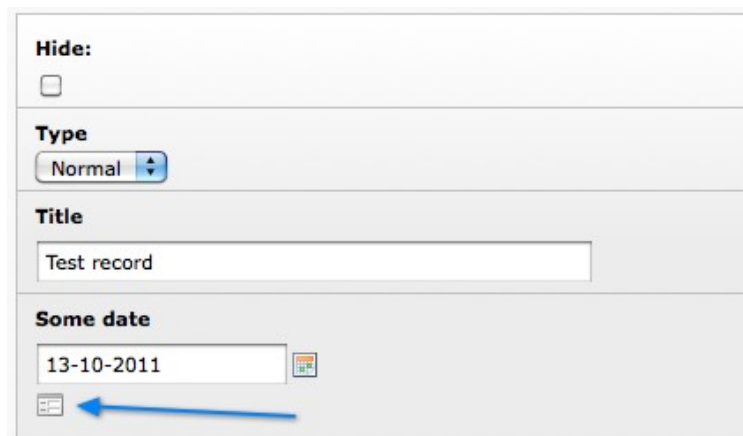
Let's add a palette to the example from the previous section. The palette itself is defined like this:

```
'palettes' => array(
    '1' => array('showitem' => 'enforce_date'),
),
```

Now we change the "types" configuration to link the palette to the "some\_date" field:

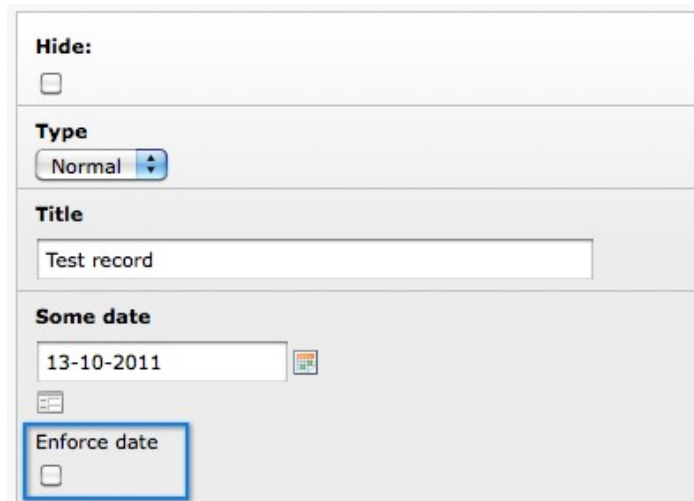
```
'0' => array('showitem' => 'hidden, record_type, title, some_date;;1 '),
```

When a palette exists, an icon appears next to the relevant field:



The screenshot shows a form with several fields: 'Hide' (checkbox), 'Type' (dropdown menu set to 'Normal'), 'Title' (text input with 'Test record'), and 'Some date' (date input with '13-10-2011'). A small icon representing a palette is visible to the right of the date input. A blue arrow points to this icon.

Clicking on this icon, the palette is revealed:



The screenshot shows the same form as before, but the palette is now revealed. It contains a checkbox labeled 'Enforce date' which is currently unchecked. The entire palette section is highlighted with a blue border.

Palette display can be activated permanently by checking the "Show secondary options" box at the bottom of any forms screen:

☒ Show secondary options (palettes)



### Note

This checkbox may be hidden by TSConfig, so it may not appear all the time.



Key	Datatype	Description
<b>showitem</b>	string (list of field names)	<b>Required.</b> Configuration of the displayed order of fields in the palette. Remember that a field name must not appear in more than one palette and not more than one time!. E.g. 'hidden,starttime,endtime'
<b>canNotCollapse</b>	boolean	If set, then this palette is not allowed to 'collapse' in the TCEforms-display. This basically means that if "Show secondary options" is not on, this palette is <i>still</i> displayed in the main form and not linked with an icon.

All fields in a palette are shown on a single line. Since TYPO3 4.3, it is possible to place them on several lines by using the `--linebreak--` keyword.

**Example**

```
'palettes' => array(
    '1' => array('showitem' => 'salutation, firstname, lastname, --linebreak--, mobile,
phone, fax, --linebreak--, email, email_work),
)
```

# Additional \$TCA features

## Special Configuration introduction

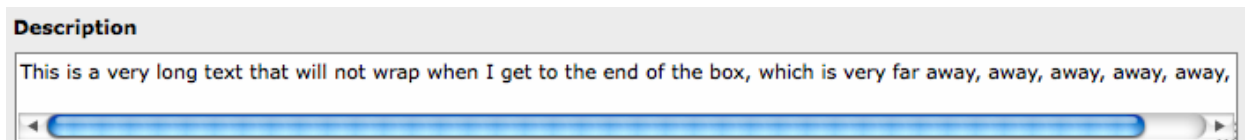
In relation to "types"-configuration it is possible to pass special parameters to a field only for certain "types"-configurations. For instance you can define that a text field should not wrap text lines for certain types. Let's add the "description" field to our previous example, a field which was not displayed until now. The configuration for type "0" becomes:

```
'0' => array('showitem' => 'hidden;;;1-1-1, record_type;;;2-2-2, title;;;3-3-3,
description;;;nowrap, some_date;;1 '),
```

Notice the keyword "nowrap" in position 4 for the field "description". The field itself is defined like this in the columns section:

```
'description' => array(
    'exclude' => 0,
    'label' => 'LLL:EXT:examples/locallang_db.xml:tx_examples_dummy.description',
    'config' => array(
        'type' => 'text',
        'cols' => 50,
        'rows' => 3
    )
)
```

The result is a textarea field where lines are not wrapped automatically when reaching the width of the box:



The point of setting "nowrap" in the "types"-configuration is that under other "types"-configurations the field *will* wrap lines. Likewise you can configure an RTE to appear for a field only if a certain type of the record is set and in other cases not.

## Default Special Configuration (defaultExtras)

Since "types"-configuration does not apply for FlexForms and since a feature available as special configuration is sometimes needed regardless of type value you can also configure the default value of the special configuration. This is done with a key in the ['columns']['field name'] array. Thus, the alternative configuration for the example above could be:

```
'description' => array(
    'exclude' => 0,
    'label' => 'LLL:EXT:examples/locallang_db.xml:tx_examples_dummy.description',
    'config' => array(
        'type' => 'text',
        'cols' => 50,
        'rows' => 3
    ),
    'defaultExtras' => 'nowrap'
)
```

and the "nowrap" parameter doesn't appear in the "types"-configuration anymore:

```
'0' => array('showitem' => 'hidden;;;1-1-1, record_type;;;2-2-2, title;;;3-3-3,
description, some_date;;1 '),
```

This works equally well.

## Special Configuration options

### Keywords

This table lists the options for keywords in special configuration. Each keyword is followed by a *value* wrapped in [] (square brackets).

It is possible to use several keywords. Each must be separated by a colon (:). See examples below.

Keyword	Description	Value syntax	Examples
<b>nowrap</b>	Disables line wrapping in "text" type fields.	[no options]	
<b>richtext</b>	Enables the RTE for the field and allows you to set which toolbar buttons must be shown on top of the existing configuration.	Blank, * or keywords separated by " "	richtext[*] = all RTE options richtext[] = inherit default configuration richtext[cut copy paste] = ensures that cut, copy and paste options are shown regardless of RTE configuration See RTE API definition later for more details.
<b>rte_transform</b>	Configuration of RTE transformations and other options. <i>See table below for a list of the key values possible.</i>	key1=value2 key2=value2 key3=value3 ...	rte_transform[key1=value1 key2=value2 key3=value3]
<b>fixed-font</b>	Use a monospace font in "textarea" type fields.	[no options]	
<b>enable-tab</b>	Enable tabulator inside "textarea" type fields.	[no options]	
<b>rte_only</b>	If set, the field can <i>only</i> be edited with a Rich Text Editor - otherwise it will not show up.	boolean (0/1)	
<b>static_write</b>	This allows to configure a field value to be written to a file. <i>See table below for value of f1-f5</i>	f1 f2 f3 f4 f5	
<b>wizards</b>	Used to specifically enable wizards configured for a field. See option "enableByTypeConfig" in the wizard configuration.	wizard-key1 wizard-key2 ...	wizards[table]

### rte\_transform[] key/value pairs

Keyword	Description	Value syntax	Examples
<b>flag</b>	This points to a field in the row which determines whether or not the RTE is disabled. If the value of the field is set, then the RTE is disabled.	Field name	rte_transform[flag=rte_disable]
<b>mode</b>	Configures which transformations the content will pass through between the database and the RTE application.	Transformation keywords separated by dashes ("-"). The order is calling order when direction is "db". <i>See RTE API section for list of transformations available.</i>	rte_transform[mode=ts-css-images]

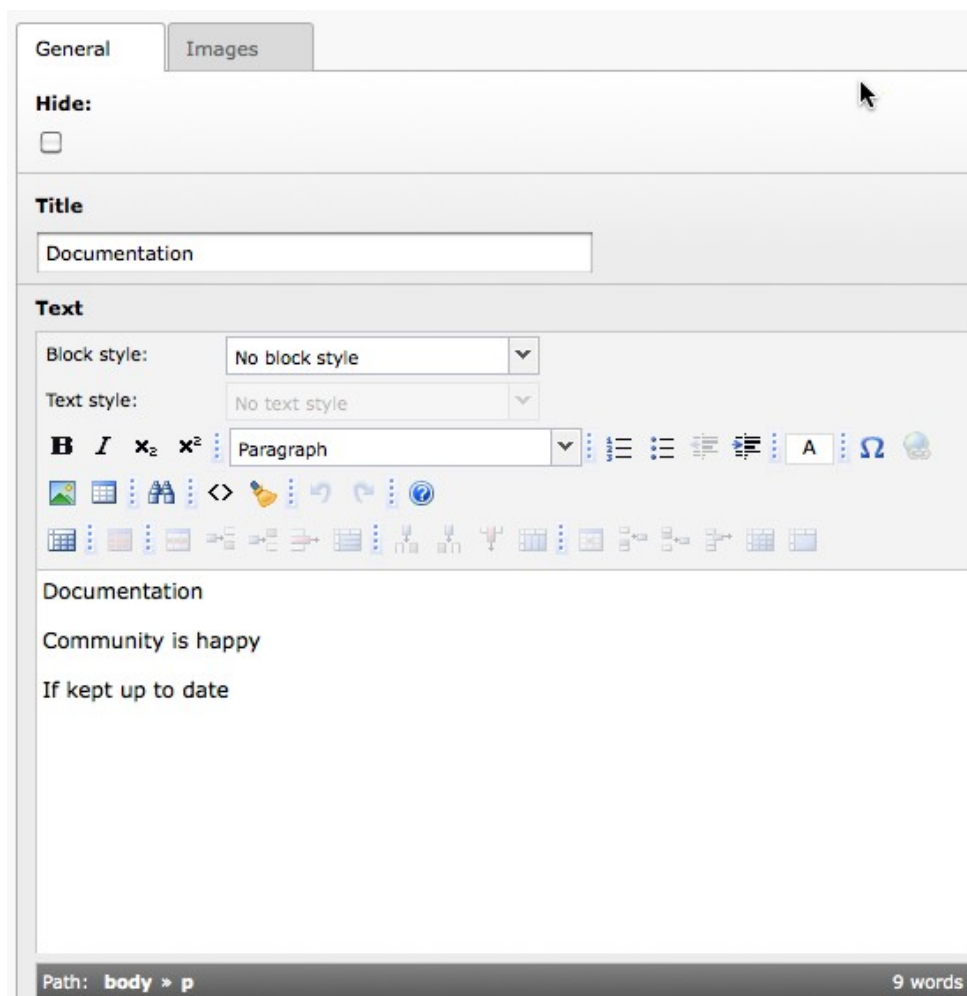
Keyword	Description	Value syntax	Examples
<b>imgpath</b>	This sets an alternative path for Rich Text Editor images. Default is configured by the value <code>TYPO3_CONF_VARS["BE"] ["RTE_imageStorageDir"]</code> (default is "uploads/")	path relative to <code>PATH_site</code> , e.g. "uploads/rte_test/"	

#### Example - Setting up Rich Text Editors

Let's take another table from the "examples" extension to look at how to set up a text will with a RTE. The table is called "tx\_examples\_haiku" and it contains a column called "poem" on which we want to have the RTE. Its configuration looks like this:

```
'poem' => array(
    'exclude' => 0,
    'label' => 'LLL:EXT:examples/locallang_db.xml:tx_examples_haiku.poem',
    'config' => array(
        'type' => 'text',
        'cols' => 40,
        'rows' => 6
    ),
    'defaultExtras' => 'richtext[]:static_write[filename|poem]'
)
```

Concentrate on just the part in bold. This example contains no additional configuration (notice the empty square brackets), meaning the RTE will inherit from the TYPO3-wide configuration (as defined by Page and User TSconfig). This may look like this (depending on your local RTE configuration):



**static\_write[] parameters**

Keyword	Description
<b>f1</b>	The field name which contains the name of the file being edited. This filename should be relative to the path configured in \$TYPO3_CONF_VARS["BE"]["staticFileEditPath"] (which is "fileadmin/static/" by default).  The file <b>must</b> exist and be writable.
<b>f2</b>	The field name which will also receive a copy of the content (in the database). This should probably be the field name that carries this configuration.
<b>f3</b>	The field name containing the alternative subpart marker used to identify the editable section in the file. The default marker is ###TYPO3_STATICFILE_EDIT### and may be encapsulated in HTML comments. There must be two markers, one to identify the beginning and one for the end of the editable section. Optional.
<b>f4</b>	The field name of the record which - if true - indicates that the content should always be loaded into the form from the file and not from the duplicate field in the database.
<b>f5</b>	The field name which will receive a status message as a short text string. Optional.

**Example - Write to static file**

Let's go back to the above example and look at the second part of the "defaultExtras" configuration (in bold):

```

'poem' => array(
    'exclude' => 0,
    'label' => 'LLL:EXT:examples/locallang_db.xml:tx_examples_haiku.poem',
    'config' => array(
        'type' => 'text',
        'cols' => 40,
        'rows' => 6
    ),
    'defaultExtras' => 'richtext[]:static_write[filename|poem]'
)

```

This configuration means that the content of the “pœm” field will be written to the file given in “filename”. It looks like this in the BE:

The screenshot shows the TYPO3 Backend Editor (BE) interface. At the top, there's a 'Title' field with the value 'Documentation'. Below it is the 'Text' field, which has a 'Block style' dropdown set to 'No block style' and a 'Text style' dropdown set to 'No text style'. The 'Text' field itself contains the text 'Documentation', 'Community is happy', and 'If kept up to date'. Below the text field, there's a 'Path' field showing 'body » p' and a word count of '9 words'. At the bottom, there's a 'File' field with the value 'myhaiku.txt'.

Before saving the content of "fileadmin/static/myhaiku.txt" must be:

```
###TYPO3_STATICFILE_EDIT###
###TYPO3_STATICFILE_EDIT###
```

After saving the content of "fileadmin/static/myhaiku.txt" looks like this:

```
###TYPO3_STATICFILE_EDIT###
<p>Documentation</p><p>Community is happy</p><p>If kept up to date</p>
###TYPO3_STATICFILE_EDIT###
```

## Soft References

"Soft References" are references to database elements, files, email addresses, URLs etc. which are found in-text in content. The <link [page\_id]> tag from typical bodytext fields are an example of this.

The Soft Reference parsers are used by the system to find these references and process them accordingly in import/export actions and copy operations. Also, the soft references are utilized by integrity checking functions.

### Default soft reference parsers

The class “t3lib\_softrefproc” contains generic parsers for the most well-known types which are default for most TYPO3 installations. This is the list of the possible keys:

“softref” key	Description
<b>substitute</b>	A full field value targeted for manual substitution (for import /export features)
<b>notify</b>	Just report if a value is found, nothing more.
<b>images</b>	HTML <img> tags for RTE images / images from fileadmin/

"softref" key	Description
<b>typolink</b>	References to page id or file, possibly with anchor/target, possibly comma-separated list.
<b>typolink_tag</b>	As typolink, but searching for <link> tag to encapsulate it.
<b>TSconfig</b>	Processing (filerefs? Domains? what do we know...)
<b>TStemplate</b>	Free text references to "fileadmin/" files. HTML resource links like <a>, <img>, <form>
<b>ext_fileref</b>	Relative file reference, prefixed "EXT:[extkey]/" - for finding extension dependencies
<b>email</b>	Email highlight
<b>url</b>	URL highlights (with a scheme)

These are by default set up in the config\_default.php file:

```
'SC_OPTIONS' => array(
    'GLOBAL' => array(
        'softRefParser' => array(
            'substitute' =>
                't3lib/class.t3lib_softrefproc.php:&t3lib_softrefproc',
            'notify' => 't3lib/class.t3lib_softrefproc.php:&t3lib_softrefproc',
            'images' => 't3lib/class.t3lib_softrefproc.php:&t3lib_softrefproc',
            'typolink' =>
                't3lib/class.t3lib_softrefproc.php:&t3lib_softrefproc',
            'typolink_tag' =>
                't3lib/class.t3lib_softrefproc.php:&t3lib_softrefproc',
            'TSconfig' =>
                't3lib/class.t3lib_softrefproc.php:&t3lib_softrefproc',
            'TStemplate' =>
                't3lib/class.t3lib_softrefproc.php:&t3lib_softrefproc',
            'ext_fileref' =>
                't3lib/class.t3lib_softrefproc.php:&t3lib_softrefproc',
            'email' => 't3lib/class.t3lib_softrefproc.php:&t3lib_softrefproc',
            'url' => 't3lib/class.t3lib_softrefproc.php:&t3lib_softrefproc',
        )
    )
),
```

## User-defined soft reference parsers

Soft References can also be user-defined. It is easy to set them up by simply adding new keys in \$TYP03\_CONF\_VARS['SC\_OPTIONS']['GLOBAL']['softRefParser']. Use key names based on the extension you put it in, e.g. "tx\_myextensionkey".

The class containing the soft reference parser must have a function named "findRef". Please refer to the class "t3lib\_softrefproc" from t3lib/ for the API to use and return.

## Wizards Configuration

Wizards are configurable for some field types, namely "input", "text", "select" and "group" types. They provide a way to insert helper-elements, links to wizard scripts etc.

A well known example of a wizard application is the form wizard:

**Form Structure**

Name: | \*name=input,40 | Enter your name here  
Email: | \*email=input,40  
Your enquiry: | \*Your\_enquiry=textarea  
| formtype\_mail=submit | Submit Feedback  
| html\_enabled=hidden | 1  
| subject=hidden | From Demo Site mail form  
# Example content:

The wizard is configured for the text area field and appears as an icon to the right. Clicking the icon will guide the user to a view where the "cryptic" form code is presented in a more user-friendly

interface:

Forms wizard		
Preview of element:	Element type:	Detailed configuration:
	<b>Name:</b> Type: <input type="text" value="Input field"/> Label: <input type="text" value="Name:"/> Required: <input checked="" type="checkbox"/>	Field: <input type="text" value="name"/> Size: <input type="text" value="40"/> Max: <input type="text"/> Value: <input type="text" value="Enter your name here"/>
	<b>Email:</b> Type: <input type="text" value="Input field"/> Label: <input type="text" value="Email:"/> Required: <input checked="" type="checkbox"/>	Field: <input type="text" value="email"/> Size: <input type="text" value="40"/> Max: <input type="text"/> Value: <input type="text"/>
	<b>Your enquiry:</b> Type: <input type="text" value="Text area"/> Label: <input type="text" value="Your enquiry:"/> Required: <input checked="" type="checkbox"/>	Field: <input type="text" value="Your_enquiry"/> Columns: <input type="text"/> Rows: <input type="text"/> No Wrap: <input type="checkbox"/> Value: <input type="text"/>
Special configuration for mail forms: ?		
	Send button label:	<input type="text" value="Submit Feedback"/>
	HTML mode enabled:	<input checked="" type="checkbox"/>
	Subject:	<input type="text" value="From Demo Site mail form"/>
	Recipient email:	<input type="text" value="introduction-feedback@ty"/>

Another example of wizards are the new / edit / suggest wizards which are available for "group" or "select" type fields:

Include Basis Template:

root\_systemConfiguration

root\_extensionConfiguration

root\_menu

root\_blocks

root\_page

⬆

⬆

⬇

⬇

✕

📁

📄

📄

📄

📄

📄

root\_systemConfiguration [20]

root\_extensionConfiguration [49]

root\_menu [15]

root\_blocks [36]

root\_page [8]

🔍 Find records

✎

+

📄 Template

## Configuration of wizards

The value of the “wizards” key in the field config-array is an array. Each key is yet another array which configures the individual wizards for a field. The order of the keys determines the order the wizards are displayed in. The key-values themselves play no important role (except from a few reserved words listed in a table below).

The configuration for the new / edit / suggest wizards shown above looks like this:

```
'basedOn' => array(
    'label' => 'LLL:EXT:cms/locallang_tca.xml:sys_template.basedOn',
    'config' => array(
        'type' => 'group',
        'internal_type' => 'db',
        'allowed' => 'sys_template',
        'show_thumbs' => '1',
        'size' => '3',
        'maxitems' => '50',
        'autoSizeMax' => 10,
        'minitems' => '0',
        'default' => '',
        'wizards' => array(
            '_PADDING' => 4,
            '_VERTICAL' => 1.
        )
    )
)
```



```

'suggest' => array(
    'type' => 'suggest',
),
'edit' => array(
    'type' => 'popup',
    'title' => 'Edit template',
    'script' => 'wizard_edit.php',
    'popup_onlyOpenIfSelected' => 1,
    'icon' => 'edit2.gif',
    'JSopenParams' =>
'height=350,width=580,status=0,menubar=0,scrollbars=1',
),
'add' => array(
    'type' => 'script',
    'title' => 'LLL:EXT:cms/locallang_tca.xml:sys_template_basedOn_add',
    'icon' => 'add.gif',
    'params' => array(
        'table'=>'sys_template',
        'pid' => '###CURRENT_PID###',
        'setValue' => 'prepend'
    ),
    'script' => 'wizard_add.php',
)
)
)
),

```

The part specific to the wizards configuration is highlighted in bold. The first two lines of this configuration make use of two reserved keywords to define settings for the display of icons.

## Reserved keys

Each wizard is identified by a key string. However some strings are reserved for general configuration. These are listed in this table and as a rule of thumb they are prefixed with an underscore ("\_"):

Key	Type	Description
<b>_ POSITION</b>	string	Determines the position of the wizard-icons/titles. Default is “right”. Possible values are “left”, “top”, “bottom”.
<b>_ VERTICAL</b>	boolean	If set, the wizard icons (if more than one) will be positioned in a column (vertically) and not a row (horizontally, which is default)
<b>_ DISTANCE</b>	int+	The distance in pixels between wizard icons (if more than one).
<b>_ PADDING</b>	int+	The cellpadding of the table which keeps the wizard icons together.
<b>_ VALIGN</b>	string	valign attribute in the table holding things together.
<b>_ HIDDENFIELD</b>	boolean	If set, the field itself will be a hidden field (and so not visible...)
<b>[any other key]</b>	PHP-Array	Configuration of the wizard types, see below.

## General configuration options

This table lists the general configuration options for (almost) all wizard types. In particular the value of the "type" key is important because it denotes what additional options are available.

Key	Type	Description
<b>type</b>	string	Defines the type of wizard. The options are listed as headlines in the table below. <b>This setting is required!</b>
<b>title</b>	string or LLL reference	This is the title of the wizard. For those wizards which require a physical representation – e.g. a link - this will be the link if no icon is presented.
<b>icon</b>	fileref	This is the icon representing the wizard. If the first 3 chars are NOT “./” then the file is expected to be in “t3lib/gfx/”. So to insert custom images, put them in “./typo3conf/” or so. You can also prefix icons from extensions with "EXT:ext/[extension key]/directory.../". Generally, the format is the same as for referring to icons for selector box options.

Key	Type	Description
		If the icon is not set, the title will be used for the link.
<b>enableByTypeConfig</b>	boolean	If set, then the wizard is enabled only if declared in the Special Configuration of specific types (using "wizards[ <i>list of wizard-keys</i> ]"). See wizard section.
<b>RTEonly</b>	boolean	If set, then this wizard will appear only if the wizard is presented for a RTE field.
<b>hideParent</b>	array	If set, then the real field will not be shown (but rendered as a hidden field). In "hideParent" you can configure the non-editable display of the content as if it was a field of the "none" type. The options are the same as for the "config" key for "none" types.

## Specific wizard configuration options based on type

Key	Type	Description
<b>Type: script</b> Creates a link to an external script which can do "context sensitive" processing of the field. This is how the Form and Table wizards are used.		
<b>notNewRecords</b>	boolean	If set, the link will appear <i>only</i> if the record is not new (that is, it has a proper UID)
<b>script</b>	PHP-script filename	If the first 3 chars are NOT "../" then the file is expected to be in "typo3/". So to link to custom script, put it in "../typo3conf/". File reference can be prefixed "EXT:[extension key]/" to point to a file inside an extension. A lot of parameters are passed to the script as GET-vars in an array, P.
<b>params</b>	array	Here you can put values which are passed to your script in the P array.
<b>popup_onlyOpenIfSelected</b>	boolean	If set, then an element (one or more) from the list must be selected. Otherwise the popup will not appear and you will get a message alert instead. This is supposed to be used with the wizard_edit.php script for editing records in "group" type fields.
<b>Type: popup (+colorbox)</b> Creates a link to an external script opened in a pop-up window.		
<b>notNewRecords</b>	boolean	See above, type "script"
<b>script</b>	PHP-script filename	See above, type "script"
<b>params</b>		See above, type "script"
<b>JSopenParams</b>	string	Parameters to open JS window:  <b>Example:</b>  <pre>"JSopenParams" =&gt; "height=300,width=250,status=0,menubar=0,scrollbars=1",</pre>
<b>Type: userFunc</b> Calls a user function/method to produce the wizard or whatever they are up to.		
<b>notNewRecords</b>	boolean	See above, type "script"
<b>userFunc</b>	string	Calls a function or a method in a class.  <b>Methods:</b> [classname]->[methodname]  <b>Functions:</b> [functionname] The function/class must be included on beforehand. This is advised to be done within the localconf.php file. Two parameters are passed to the function/method: 1) An array with parameters, much like the ones passed to scripts. One key is special though: the "item" key, which is passed by reference. So if you alter that value it is reflected <i>back!</i> 2) \$this (reference to the TCEform-object). The content returned from the function call is inserted at the position where the icon/title would normally go.

Key	Type	Description
<p><b>Type: colorbox</b>  Renders a square box (table) with the background color set to the value of the field. The id-attribute is set to a md5-hash so you might change the color dynamically from pop-up- wizard.  The icon is not used, but the title is given as alt-text inside the color-square.</p>		
<b>dim</b>	W x H, pixels	<p>Determines the dimensions of the box. Default is 20 pixels.</p> <pre>"dim" =&gt; "50x20",</pre>
<b>tableStyle</b>	style-attribute content in table-tag	<p>Sets the border style of the table, eg:</p> <pre>"tableStyle" =&gt; "border:solid 1px black;"</pre>
<b>exampleImg</b>	string	<p>Reference to a sample (relative to PATH_typo3 directory). You can prefix with "EXT:" to get image from extension. An image width of 350 is optimal for display.</p> <p><b>Example:</b>  'exampleImg' =&gt; 'gfx/wizard_colorpickerex.jpg'</p>
<p><b>Type: select</b>  This renders a selector box. When a value is selected in the box, the value is transferred to the field and the field (default) element is thereafter selected (this is a blank value and the label is the wizard title).  “select” wizards make no use of the icon.  The “select” wizard's select-properties can be manipulated with the same number of TSconfig options which are available for “real” select-types in TCEFORM.[table].[field]. The position of these properties is “TCEFORM.[table].[field].wizards.[wizard-key]”.</p>		
<b>mode</b>	append, prepend, [blank]	<p>Defines how the value is processed: Either added to the front or back or (default) substitutes the existing.</p>
<b>items, foreign_table_ etc...</b>	Options related to the selection of elements known from “select” form-element type in \$TCA.	<p><b>Example:</b></p> <pre>'items' =&gt; array(     array('8 px', '8'),     array('10 px', '10'),     array('11 px', '11'),     array('12 px', '12'),     array('14 px', '14'),     array('16 px', '16'),     array('18 px', '18'),     array('20 px', '20') )</pre>
<p><b>Type: suggest</b>  This renders an input field next to a select field of type "group" (internal_type=db) or of type "select" (using foreign_table). After the user has typed at least 2 (minimumCharacters) characters in this field, a search will start and show a list of records matching the search word. The "suggest" wizard's properties can be configured directly in TCA or in page TSConfig (TCEFORM.suggest.default, TCEFORM.suggest.[queryTable], see TSconfig manual).  The configuration options are applied to each table queried by the suggest wizard. There's a general “default” configuration that applies to all tables. On top of that, there can be specific configurations for each table (use the table's name as a key). See wizard example below.</p>		
<b>pidList</b>	list of values	<p>Limit the search to certain pages (and their subpages). When pidList is empty all pages will be included in the search (as long as the be_user is allowed to see them).</p> <p><b>Example:</b></p> <pre>\$TCA['pages']['columns']['storage_pid']['config'] ['wizards']['suggest'] = array(     'type' =&gt; 'suggest',     'default' =&gt; array(         'pidList' =&gt; '1,2,3,45',     ), );</pre>

Key	Type	Description
<b>pidDepth</b>	integer	Expand pidList by this number of levels. Has an effect only if pidList has a value.  <b>Example:</b>  <pre>\$TCA['pages']['columns']['storage_pid']['config'] ['wizards']['suggest'] = array(     'type' =&gt; 'suggest',     'default' =&gt; array(         'pidList' =&gt; '6,7',         'pidDepth' =&gt; 4     ), );</pre>
<b>minimumCharacters</b>	integer	Minimum number of characters needed to start the search. Works only in "default" configuration.
<b>maxPathTitleLength</b>	integer	Maximum number of characters to display when a path element is too long
<b>searchWholePhrase</b>	boolean	Whether to do a LIKE=%mystring% (searchWholePhrase = 1) or a LIKE=mystring% (to do a real find as you type), default: 0  <b>Example:</b>  <pre>\$TCA['pages']['columns']['storage_pid']['config'] ['wizards']['suggest'] = array(     'type' =&gt; 'suggest',     'default' =&gt; array(         'searchWholePhrase' =&gt; 1,     ), );</pre>
<b>searchCondition</b>	string	Additional WHERE clause (no AND needed to prepend)  <b>Example:</b>  <pre>// configures the suggest wizard for the field "storage_pid" in table "pages" to search only for pages with doktype=1 \$TCA['pages']['columns']['storage_pid']['config'] ['wizards']['suggest'] = array(     'type' =&gt; 'suggest',     'default' =&gt; array(         'searchCondition' =&gt; 'doktype=1',     ), );</pre>
<b>cssClass</b>	string	Add a CSS class to every list item of the result list.
<b>receiverClass</b>	string	PHP class alternative receiver class - the file that holds the class needs to be included manually before calling the suggest feature (default: t3lib_tceforms_suggest_defaultreceiver), should be derived from "t3lib_tceforms_suggest_defaultreceiver".
<b>renderFunc</b>	string	User function to manipulate the displayed records in the results.
<p><b>Type: slider</b></p> <p>This renders a slider next to the field. It works for either input-type fields or select-type fields. For select-type fields, the wizard will "slide" through the items making up the field. For input-type fields, it will work only for fields evaluated to integer, float and time. It is advised to also define a "range" property, otherwise the slider will go from 0 to 10000.  <b>Note:</b> the range is properly taken into account only as of TYPO3 4.6.1.</p>		
<b>step</b>	integer/float	Sets the step size the slider will use. For floating point values this can itself be a floating point value.
<b>width</b>	pixels	Defines the width of the slider

In the next section the more complex core wizard scripts are demonstrated with examples. Before that, here are a few examples of simpler core wizards.

### Example - Selector box of preset values

You can add a selector box containing preset values next to a field:

The screenshot shows a form with two fields. The first field is labeled 'Season' and has a dropdown menu open showing the options 'Spring', 'Summer', 'Autumn', and 'Winter'. The second field is labeled 'Weirdness' and is an input field with a green border and a green background. To the right of the 'Weirdness' field are two buttons, '+' and '-'.

When an option from the selector box is selected it will be transferred to the input field of the element. The mode of transfer can be either substitution (default) or prepending or appending the value to the existing value.

This is the corresponding TCA configuration:

```
'season' => array(
    'exclude' => 0,
    'label' => 'LLL:EXT:examples/locallang_db.xml:tx_examples_haiku.season',
    'config' => array(
        'type' => 'input',
        'size' => 20,
        'eval' => 'trim',
        'wizards' => array(
            'season_picker' => array(
                'type' => 'select',
                'mode' => '',
                'items' => array(
                    array('LLL:EXT:examples/locallang_db.xml:tx_examples_haiku.season.spring', 'Spring'),
                    array('LLL:EXT:examples/locallang_db.xml:tx_examples_haiku.season.summer', 'Summer'),
                    array('LLL:EXT:examples/locallang_db.xml:tx_examples_haiku.season.autumn', 'Autumn'),
                    array('LLL:EXT:examples/locallang_db.xml:tx_examples_haiku.season.winter', 'Winter'),
                )
            )
        )
    )
),
```

### Example - User defined wizard (processing with PHP function)

The "userFunc" type of wizard allows you to render all the wizard code yourself. Theoretically, you could produce all of the other wizard kinds ("script", "popup", "colorbox", etc.) with your own user function if you wanted to alter their behavior.

In this example the wizard provides to JavaScript-powered buttons that make it possible to increase or decrease the value in the field by 1. The wizard also highlights the field with a background color. This is how it looks:

The screenshot shows a form with a single field labeled 'Weirdness'. The field is an input field with a green border and a green background, displaying the value '-3'. To the right of the field are two buttons, '+' and '-'.

The corresponding configuration is:

```
'weirdness' => array(
    'exclude' => 0,
    'label' => 'LLL:EXT:examples/locallang_db.xml:tx_examples_haiku.weirdness',
    'config' => array(
        'type' => 'input',
        'size' => 10,
        'eval' => 'int',
        'wizards' => array(
            'specialWizard' => array(
                'type' => 'userFunc',
                'userFunc' =>
                    'EXT:examples/class.tx_examples_tca.php:tx_examples_tca->someWizard',
                'params' => array(
                    'color' => 'green'
                )
            )
        )
    )
),
```

```

    )
),

```

Notice the “params” array, which is passed to the user function that handles the wizard. And here’s the code of the user function (from file `class.tx_examples_tca.php` of the “examples” extension):

```

function someWizard($PA, $fObj) {
    // Note that the information is passed by reference,
    // so it's possible to manipulate the field directly
    // Here we highlight the field with the color passed as parameter
    $backgroundColor = 'white';
    if (!empty($PA['params']['color'])) {
        $backgroundColor = $PA['params']['color'];
    }
    $PA['item'] = '<div style="background-color: ' . $backgroundColor . '"; padding:
4px;">' . $PA['item'] . '</div>';

    // Assemble the wizard itself
    $output = '<div style="margin-top: 8px; margin-left: 4px;">';
    // Create the + button
    $onClick = "document." . $PA['formName'] . "['" . $PA['itemName'] . "'].value++;
return false;";
    $output .= '<a href="#" onclick="' . htmlspecialchars($onClick) . '" style="padding:
6px; border: 1px solid black; background-color: #999">+</a>';
    // Create the - button
    $onClick = "document." . $PA['formName'] . "['" . $PA['itemName'] . "'].value--;
return false;";
    $output .= '<a href="#" onclick="' . htmlspecialchars($onClick) . '" style="padding:
6px; border: 1px solid black; background-color: #999">-</a>';
    $output .= '</div>';
    return $output;
}

```

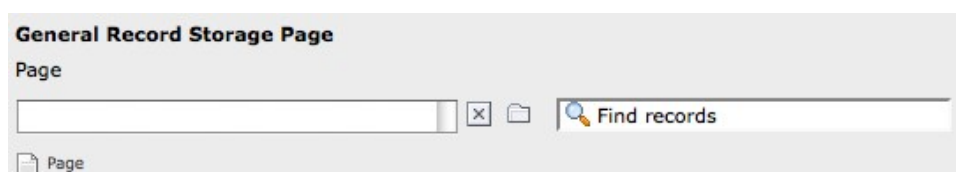
First the HTML code of the field itself is manipulated, by adding a `div` tag around it. Notice how all you need to do is to change the value of `$PA['item']` since that value is passed by reference to the function and therefore doesn’t need a return value - only to be changed. In that `div`, we use the color received as parameter.

After that we create the JavaScript and the links for both the “+” and “-” buttons and we return the resulting HTML code.

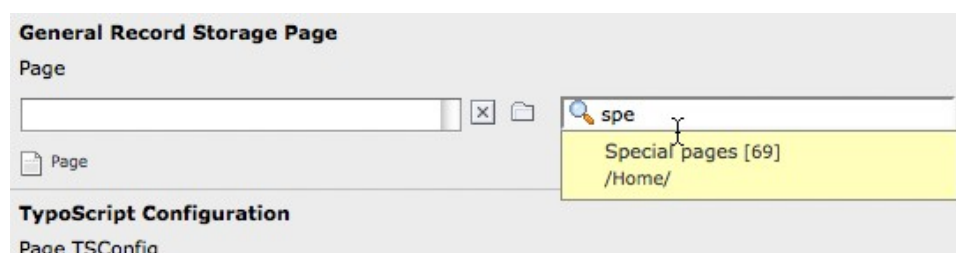
Use the `debug()` function to find more about what is available in the `$PA` array.

### Example - add a suggest wizard

As an example, let’s look at the suggest wizard setup for the “General Record Storage page”. The wizard looks like this:



And here’s the wizard in action:



Here’s the corresponding TCA configuration:

```

$TCA['pages']['columns']['storage_pid']['config']['wizards']['suggest'] = array(
    'type' => 'suggest',

```

```

        'default' => array(
            'searchWholePhrase' => 1,
            'maxPathTitleLength' => 40,
            'maxItemsInResultList' => 5
        ),
        'pages' => array(
            'searchCondition' => 'doktype=1',
        ),
    );

```

The tables that are queried are the ones used in `$TCA['pages']['columns']['storage_pid']['config']['allowed']`.

The wizard can be configured differently for each of these tables. The settings in "default" is applied to all tables. In the example above, there's a special setting for the "pages" table.

#### Example – Add a slider wizard

The "haiku" table in the "examples" extension implements a slider wizard for the "Angle" field. The field configuration looks like this:

```

'angle' => array(
    'exclude' => 0,
    'label' => 'LLL:EXT:examples/locallang_db.xml:tx_examples_haiku.angle',
    'config' => array(
        'type' => 'input',
        'size' => 5,
        'eval' => 'trim,int',
        'range' => array(
            'lower' => -90,
            'upper' => 90
        ),
        'default' => 0,
        'wizards' => array(
            'angle' => array(
                'type' => 'slider',
                'step' => 10,
                'width' => 200
            )
        )
    ),
),

```

Note the range which defines the possible values as varying from -90 to 90. With the step property we indicate that we want to progress by increments of 10. The slider wizard is rendered like this:



## Wizard scripts in the core

The wizard interface allows you to use any PHP-script for your wizards but there is a useful set of default wizard scripts available in the core of TYPO3. These are found in `PATH_typo3` and are all prefixed "wizard\_" (except "browse\_links.php").

Below is a description of each of them including a description of their special parameters and an example of usage.

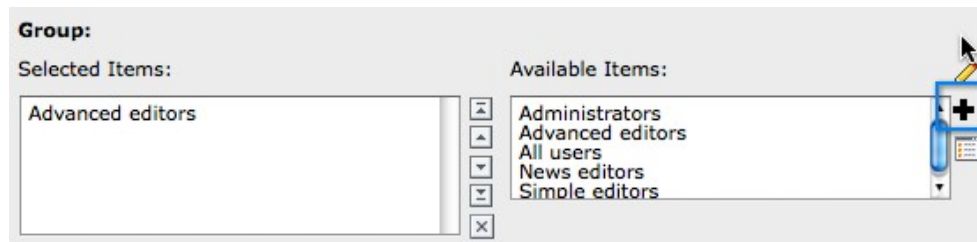
### wizard\_add.php

This script links to a form which allows you to create a new record in a given table which may optionally be set as the value on return to the real form.

Key	Type	Description
<b>table</b>	string	Table to add record in.
<b>pid</b>	int	pid of the new record. You can use the "markers" (constants) as values instead if you wish:

Key	Type	Description
		<code>###CURRENT_PID###</code> <code>###THIS_UID###</code> <code>###STORAGE_PID###</code> <code>###SITEROOT###</code>  (see TCA/select for description)
<b>setValue</b>	"prepend", "set", "append"	"set" = the field will be forced to have the new value on return "append"/"prepend" = the field will have the value appended/prepended. You must set one of these values.

As an example, let's look at BE user records where one can see several wizards in use:





The wizard appears as a “+” icon. When clicked, the user is directed to a form where a new BE user group can be created:

When the new template is saved and the user clicks the close button of the form the new group is automatically inserted as the list of selected groups.

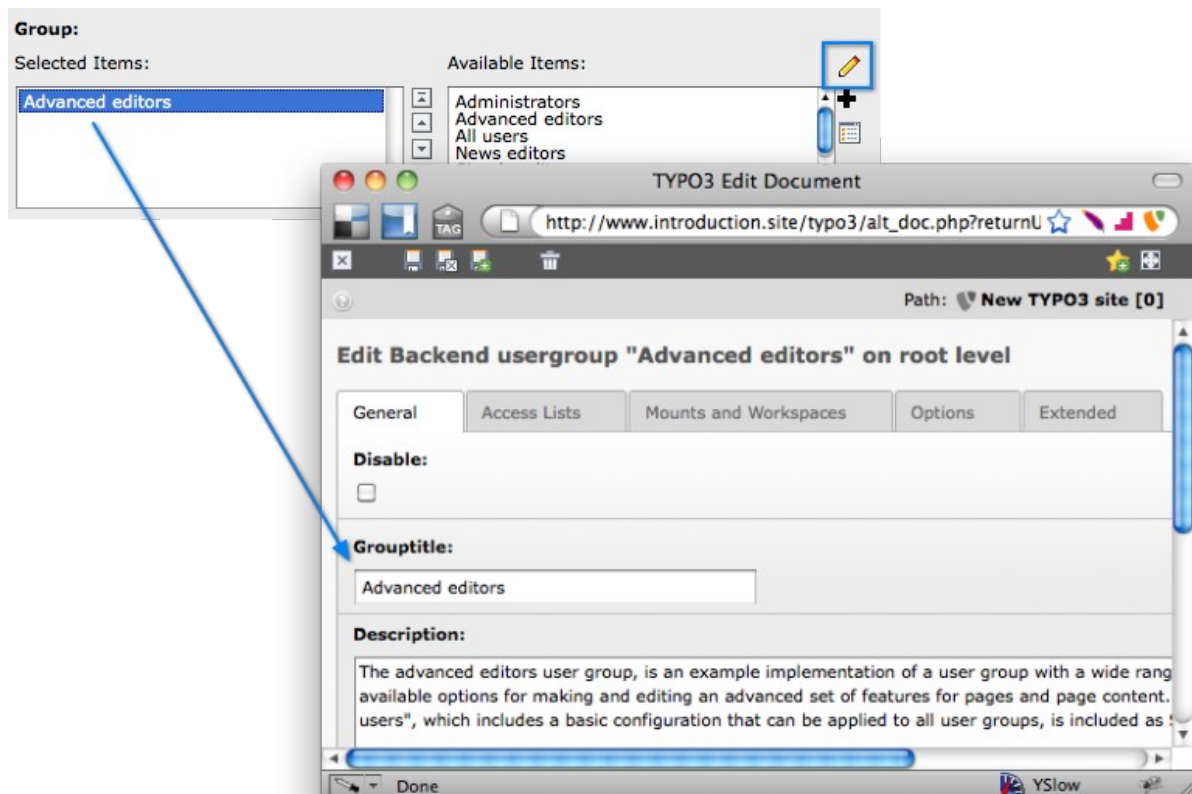
The configuration looks like this:

```
'usergroup' => array(
    'label' => 'LLL:EXT:lang/locallang_tca.xml:be_users.usergroup',
    'config' => array(
        'type' => 'select',
        'foreign_table' => 'be_groups',
        'foreign_table_where' => 'ORDER BY be_groups.title',
        'size' => '5',
        'maxitems' => '20',
        'iconsInOptionTags' => 1,
        'wizards' => array(
            '_PADDING' => 1,
            '_VERTICAL' => 1,
            'edit' => array(
                'type' => 'popup',
                'title' =>
                    'LLL:EXT:lang/locallang_tca.xml:be_users.usergroup_edit_title',
                'script' => 'wizard_edit.php',
                'popup_onlyOpenIfSelected' => 1,
                'icon' => 'edit2.gif',
                'JSopenParams' =>
                    'height=350,width=580,status=0,menubar=0,scrollbars=1',
            ),
            'add' => array(
                'type' => 'script',
                'title' =>
                    'LLL:EXT:lang/locallang_tca.xml:be_users.usergroup_add_title',
                'icon' => 'add.gif',
                'params' => array(
                    'table' => 'be_groups',
                    'pid' => '0',
                    'setValue' => 'prepend'
                ),
                'script' => 'wizard_add.php',
            ),
        ),
        'list' => array(
            'type' => 'script',
            'title' =>
                'LLL:EXT:lang/locallang_tca.xml:be_users.usergroup_list_title',
            'icon' => 'list.gif',
            'params' => array(
                'table' => 'be_groups',
                'pid' => '0',
            ),
            'script' => 'wizard_list.php',
        ),
    ),
),
```

The part in bold is related to the Add-wizard. Note how it points to the "wizard\_add.php" script. The "params" array instructs the Add-wizard how to handle the creation of the new record, i.e. which table, where to store it, etc.. In particular the "setValue" parameter tells the wizard script that the uid of the newly created record should be inserted in the relations field of the original record (the one where we clicked the Add-wizard's icon).

## wizard\_edit.php

The Edit wizard gives you a shortcut to edit references in "select" or "group" type form elements. Again let's look at the BE user records:



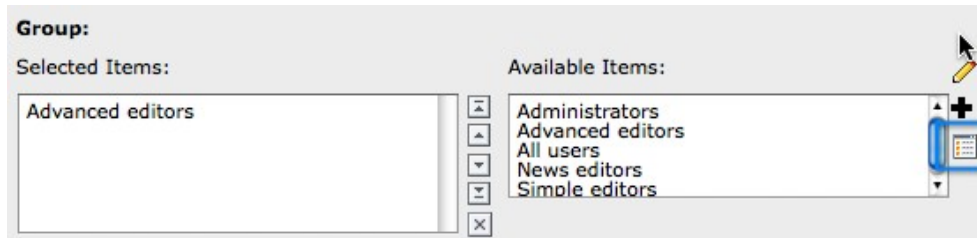
When a record is selected and the Edit-wizard button is clicked, that record opens in a new window for modification. Let's look again at the configuration (just the Edit-wizard part):

```
'usergroup' => array(
    'label' => 'LLL:EXT:lang/locallang_tca.xml:be_users.usergroup',
    'config' => array(
        ...
        'wizards' => array(
            ...
            'edit' => array(
                'type' => 'popup',
                'title' =>
                'LLL:EXT:lang/locallang_tca.xml:be_users.usergroup_edit_title',
                'script' => 'wizard_edit.php',
                'popup_onlyOpenIfSelected' => 1,
                'icon' => 'edit2.gif',
                'JSopenParams' =>
                'height=350,width=580,status=0,menubar=0,scrollbars=1',
            ),
            ...
        )
    )
),
```

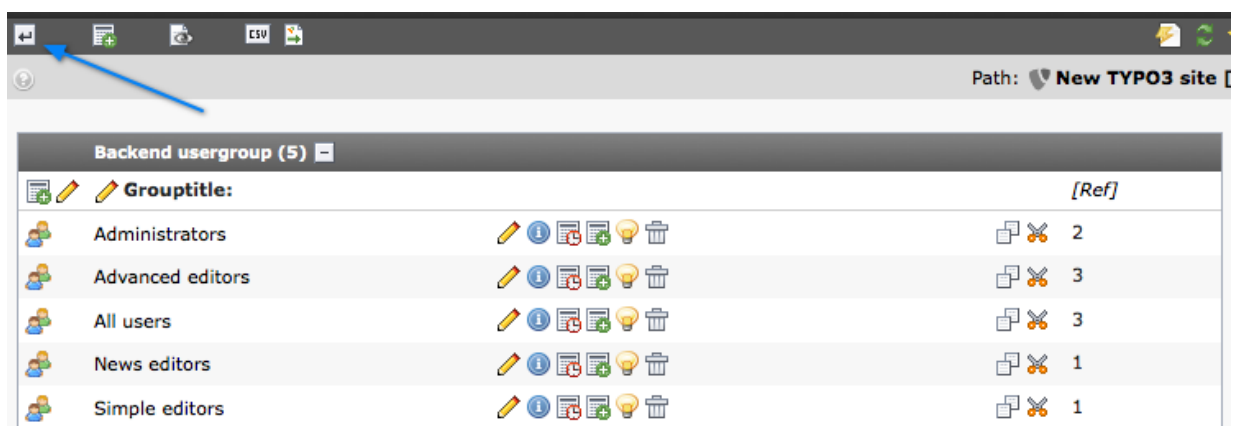
The wizard is set to type "popup" which makes it so that the selected record will open in a new window. There are no parameters to pass along like there were for the Add-wizard.

## wizard\_list.php

This links to the Web>List module for only one table and allows the user to manipulate stuff there. Again, the BE user records have it:



By clicking the icon the user gets taken to the Web>List module. Notice the "Back" link found in the upper left corner, which leads back to the edit form.



This wizard has a few parameters to configure in the "params" array:

Key	Type	Description
<b>table</b>	string	Table to manage records for
<b>pid</b>	int	id of the records you wish to list. You can use the "markers" (constants) as values instead if you wish:  <pre> ###CURRENT_PID### ###THIS_UID### ###STORAGE_PID### ###SITEROOT### </pre> (see TCA/select for description)

For the BE users table, the configuration look like this (just the List-wizard part):

```

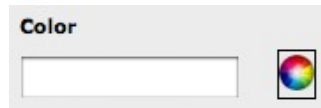
'usergroup' => array(
    'label' => 'LLL:EXT:lang/locallang_tca.xml:be_users.usergroup',
    'config' => array(
        ...
        'wizards' => array(
            ...
            'list' => array(
                'type' => 'script',
                'title' =>
                'LLL:EXT:lang/locallang_tca.xml:be_users.usergroup_list_title',
                'icon' => 'list.gif',
                'params' => array(
                    'table' => 'be_groups',
                    'pid' => '0',
                ),
                'script' => 'wizard_list.php',
            ),
        ),
    ),
),

```

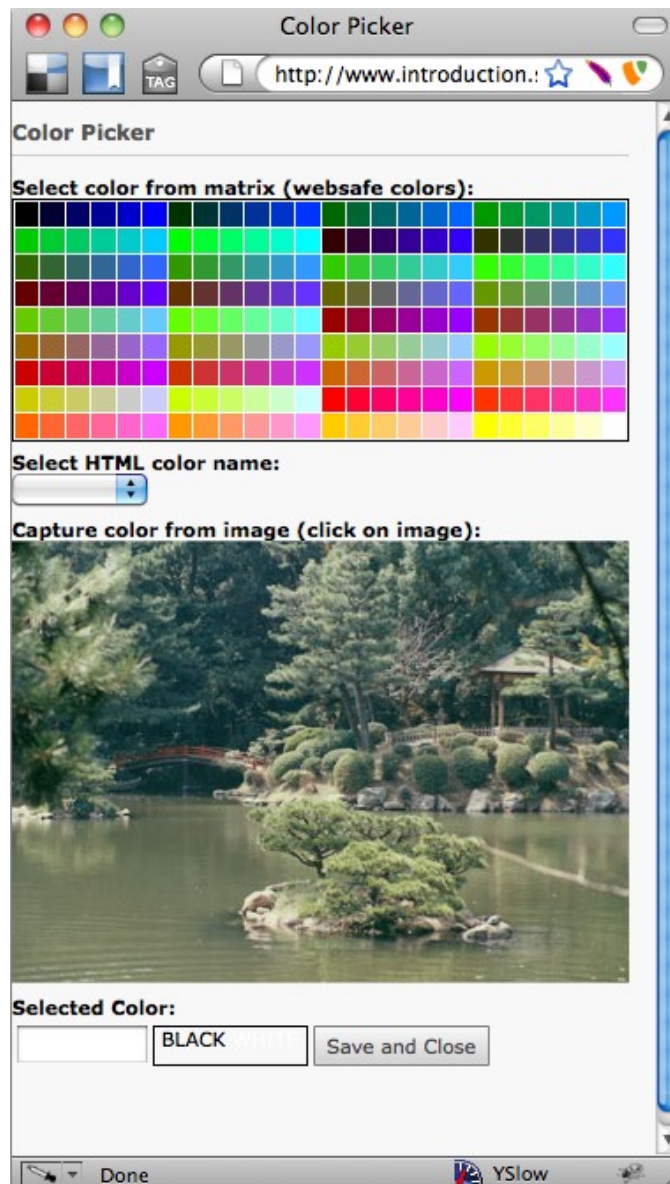
The type is also the "script" type. In the "params" array the table and pid passed to the script is set.

### wizard\_colorpicker.php

The colorpicker wizard allows you to select a HTML color value from a user-friendly pop-up box. The wizard type is "colorbox" which will first of all add a colored box next to an input field. Here's how it looks in a "haiku" record of the "examples" extension:



The color of the box is set to the value of the text field. Clicking the box will open a popup window with the full color picker wizard:



Here you can select from the web-color matrix, pick a color from the sample image or select a HTML-color name from a selector box.

The corresponding TCA configuration looks like this:

```
'color' => array(
    'exclude' => 0,
    'label' => 'LLL:EXT:examples/locallang_db.xml:tx_examples_haiku.color',
    'config' => array(
```

```
'type' => 'input',
'size' => 10,
'eval' => 'trim',
'wizards' => array(
    'colorChoice' => array(
        'type' => 'colorbox',
        'title' =>
'LL:EXT:examples/locallang_db.xml:tx_examples_haiku.colorPick',
        'script' => 'wizard_colorpicker.php',
        'dim' => '20x20',
        'tableStyle' => 'border: solid 1px black; margin-left: 20px;',
        'JSopenParams' =>
'height=600,width=380,status=0,menubar=0,scrollbars=1',
        'exampleImg' => 'EXT:examples/res/images/japanese_garden.jpg',
    )
)
),
```

Notice the wizard type which is "colorbox".

## wizard\_forms.php

The forms wizard is used typically with the Content Elements, type "Mailform". It allows to edit the code-like configuration of the mail form with a nice editor. This is shown in the introduction to Wizards above.

This is the available parameters:

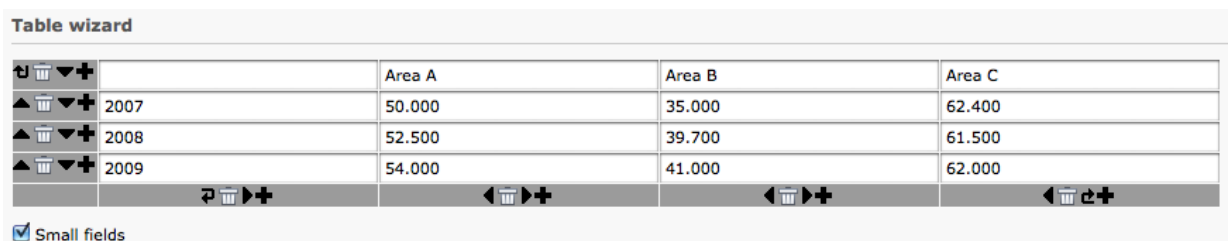
Key	Type	Description
<b>xmlOutput</b>	boolean	If set, the output from the wizard is XML instead of the strangely formatted TypoScript form-configuration code.

The configuration used for the editor in Content Elements looks like this:

```
'forms' => array(
    'notNewRecords' => 1,
    'enableByTypeConfig' => 1,
    'type' => 'script',
    'title' => 'Forms wizard',
    'icon' => 'wizard_forms.gif',
    'script' => 'wizard_forms.php?special=formtype_mail',
    'params' => array('xmlOutput' => 0)
)
```

## wizard\_table.php

The tables wizard is used typically with the Content Elements, type "Table". It allows to edit the code-like configuration of the tables with a visual editor.



This is the available parameters:

Key	Type	Description
<b>xmlOutput</b>	boolean	If set, the output from the wizard is XML instead of the TypeScript table configuration code.
<b>numNewRows</b>	integer	Setting the number of blank rows that will be added in the bottom of the table when the plus-icon is pressed. The default is 5, the range is 1-50.

This is the configuration code used for the table wizard in the Content Elements:

```
'table' => array(
    'notNewRecords' => 1,
    'enableByTypeConfig' => 1,
    'type' => 'script',
    'title' => 'Table wizard',
    'icon' => 'wizard_table.gif',
    'script' => 'wizard_table.php',
    'params' => array('xmlOutput' => 0)
),
```

## wizard\_rte.php

This wizard is used to show a "full-screen" Rich Text Editor field. The configuration below shows an example taken from the Text field in Content Elements:

```
'RTE' => array(
    'notNewRecords' => 1,
    'RTEonly' => 1,
    'type' => 'script',
    'title' => 'LLL:EXT:cms/locallang_ttc.php:bodytext.W.RTE',
    'icon' => 'wizard_rte2.gif',
    'script' => 'wizard_rte.php',
),
```

## wizard\_tsconfig.php

This wizard is used for the TSconfig fields and TypoScript Template "Setup" fields. It is specialized for that particular situations and it is not likely you will need it for anything on your own.

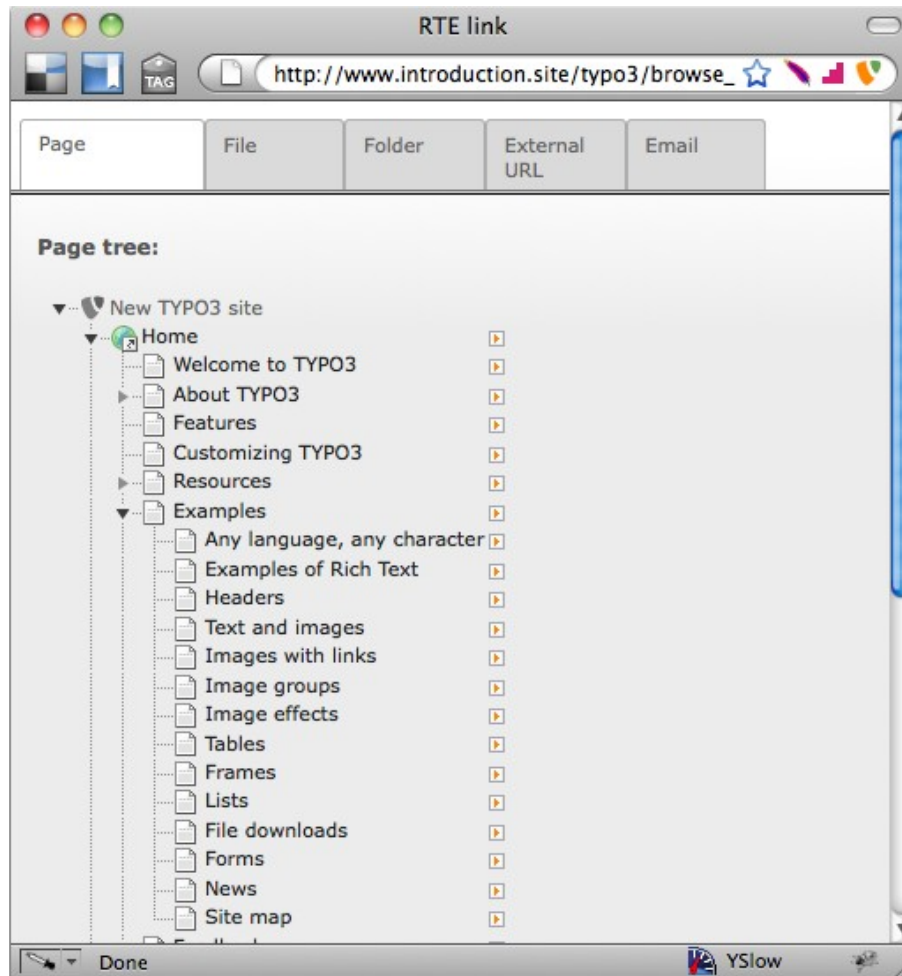
## browse\_links.php

The "Links" wizard is used many places where you want to insert link references.

Key	Type	Description
<b>allowedExtensions</b>	string	Comma separated list of allowed file extensions. By default, all extensions are allowed.
<b>blindLinkOptions</b>	string	Comma separated list of link options that should not be displayed. Possible values are file, mail, page, spec, and url. By default, all link options are displayed.

This works not only in the Rich Text Editor but also in "typolink" fields. Here's an example from tt\_content:

Clicking the wizard icons opens the Element Browser window:



Such a wizard can be configured like this:

```
'image_link' => array(
    'exclude' => 1,
    'label' => 'LLL:EXT:cms/locallang_ttc.php:image_link',
    'config' => array(
        'type' => 'input',
        'size' => '15',
        'max' => '256',
        'checkbox' => '',
        'eval' => 'trim',
        'wizards' => array(
            '_PADDING' => 2,
            'link' => array(
                'type' => 'popup',
                'title' => 'Link',
                'icon' => 'link_popup.gif',
                'script' => 'browse_links.php?mode=wizard',
                'JSopenParams' =>
                'height=300,width=500,status=0,menubar=0,scrollbars=1'
            )
        ),
        'softref' => 'typolink[linkList]'
    )
),
```

Notice how the "browse\_links.php" script requires an extra parameter since it has to return content back to the input field (and not the RTE for instance which it also supports).



# Extending the \$TCA array

Being a PHP array, the Table Configuration Array can be easily extended. TYPO3 also provides APIs for making this simpler.

## Storing the changes

Changes to the \$TCA are generally packaged into extensions and – more precisely – reside in the "ext\_tables.php" file (more details about extension structure can be found in the "Core APIs" manual).

They can also be written in the "typo3conf/extTables.php" file. The name of this file can be changed – if you so wish – by changing the value of the global variable \$typo\_db\_extTableDef\_script in the "typo3conf/localconf.php" file. It's also possible to remove the "typo3conf/extTables.php" file by setting:

```
$typo_db_extTableDef_script = 1;
```

also in "typo3conf/localconf.php". This variable is then stored into the constant TYPO3\_extTableDef\_script.

The advantage of the TYPO3\_extTableDef\_script file is that it is loaded last. This means that you are sure that your changes are not overridden by some other customizations.

The advantage of putting your changes inside an extension is that they are nicely packaged in a self-contained entity which can be easily deployed on multiple servers. The drawback is that the extension loading order cannot be finely controlled, except by editing the loaded extension list manually. At a somewhat coarser level, setting the "priority" property in the "ext\_emconf.php" file can help (a "bottom" extension will load last, but its exact load order may vary if there are several "bottom"-priority extensions).

## Customization examples

Many extracts can be found throughout the manual, but this section provides more complete examples.

### Example 1: extending the fe\_users table

The "examples" extension adds two fields to the "fe\_users" table. Here's the complete code:

```
$temporaryColumns = array (
    'tx_examples_options' => array (
        'exclude' => 0,
        'label' => 'LLL:EXT:examples/locallang_db.xml:fe_users.tx_examples_options',
        'config' => array (
            'type' => 'select',
            'items' => array (
                array('LLL:EXT:examples/locallang_db.xml:fe_users.tx_examples_options.I.0', '1'),
                array('LLL:EXT:examples/locallang_db.xml:fe_users.tx_examples_options.I.1', '2'),
                array('LLL:EXT:examples/locallang_db.xml:fe_users.tx_examples_options.I.2', '--div--'),
                array('LLL:EXT:examples/locallang_db.xml:fe_users.tx_examples_options.I.3', '3'),
            ),
            'size' => 1,
            'maxitems' => 1,
        )
    ),
    'tx_examples_special' => array (
        'exclude' => 0,
        'label' => 'LLL:EXT:examples/locallang_db.xml:fe_users.tx_examples_special',
        'config' => array (
            'type' => 'user',
            'size' => '30',
            'userFunc' => 'EXT:examples/class.tx_examples_tca.php:tx_examples_tca->specialField'
        )
    )
);
```



```
t3lib_div::loadTCA('fe_users');
t3lib_extMgm::addTCAcolumns('fe_users', $temporaryColumns,1);
t3lib_extMgm::addToAllTCAtypes('fe_users', 'tx_examples_options;;;1-1-1,
tx_examples_special');
```

First of all, the fields that we want to add are detailed according to the \$TCA syntax for columns. This configuration is stored in the \$temporaryColumns array.

After that come three precise steps:

- first we load the full \$TCA for the "fe\_users" table. This is necessary so that all columns definition are loaded. Otherwise the new columns cannot be added properly.
- second the columns are actually added to the table by using t3lib\_extMgm::addTCAcolumns().
- lastly the fields are added to the "types" definition of the "fe\_users" table by using t3lib\_extMgm::addToAllTCAtypes(). It is possible to be more fine-grained.

This does not create the corresponding fields in the database. The new fields must also be defined in the "ext\_tables.sql" file of the extension:

```
CREATE TABLE fe_users (
    tx_examples_options int(11) DEFAULT '0' NOT NULL,
    tx_examples_special varchar(255) DEFAULT '' NOT NULL
);
```



### Caution

The above statement uses the SQL CREATE TABLE statement. This is the way TYPO3 expects it to be. The Extension Manager will automatically transform this into a ALTER TABLE statement when it detects that the table already exists.

By default new fields are added at the bottom of the form when editing a record from that table. If the table uses tabs, new fields are added at the bottom of the "Extended" tab (this tab is created if it does not exist). The following screenshot shows the placement of the two new fields when editing a "fe\_users" record:

The next example shows how to place a field more precisely.

## Example 2: extending the tt\_content table

In this second example, we will add a "No print" field to all content element types. First of all, we add its SQL definition in "ext\_tables.sql":

```
CREATE TABLE tt_content (
    tx_examples_noprint tinyint(4) DEFAULT '0' NOT NULL
);
```

Then we add it to the \$TCA in "ext\_tables.php":

```
$temporaryColumn = array(
    'tx_examples_noprint' => array (
        'exclude' => 0,
        'label' => 'LLL:EXT:examples/locallang_db.xml:tt_content.tx_examples_noprint',
        'config' => array (
            'type' => 'check',
        )
    )
);
t3lib_extMgm::addTCAcolumns('tt_content', $temporaryColumn, 1);
t3lib_extMgm::addFieldsToPalette('tt_content', 'visibility', 'tx_examples_noprint',
    'after:linkToTop');
```

The code is mostly the same as in the first example, but the last line is very different and requires an explanation. Since TYPO3 4.5 the "pages" and "tt\_content" input forms were totally reorganized for increased usability. For reasons of flexibility, palettes were used intensively for all fields and not just for secondary options. This led to the introduction of new API methods for manipulating the content of palettes. The syntax is similar to the one we saw in the first example, but we have to additionally specify the palette's key, in this case "visibility".

The result is the following:

Because we added the field into an existing palette and after a specific field (as per the "after:linkToTop" directive), it gets added "inside" the form and not in the "Extended" tab.

Obviously this new field will not magically exclude a content element from being printed. For it to have any effect, it must be used during the rendering by modifying the TypoScript used to render the "tt\_content" table. Although this is outside the scope of this manual, here is an example of what you could do, for the sake of showing a complete process.

Assuming you are using "css\_styled\_content" (which is installed by default), you could add the following TypoScript to your template:

```
tt_content.stdWrap.outerWrap = <div class="noprint">|</div>
tt_content.stdWrap.outerWrap.if.isTrue.field = tx_examples_noprint
```

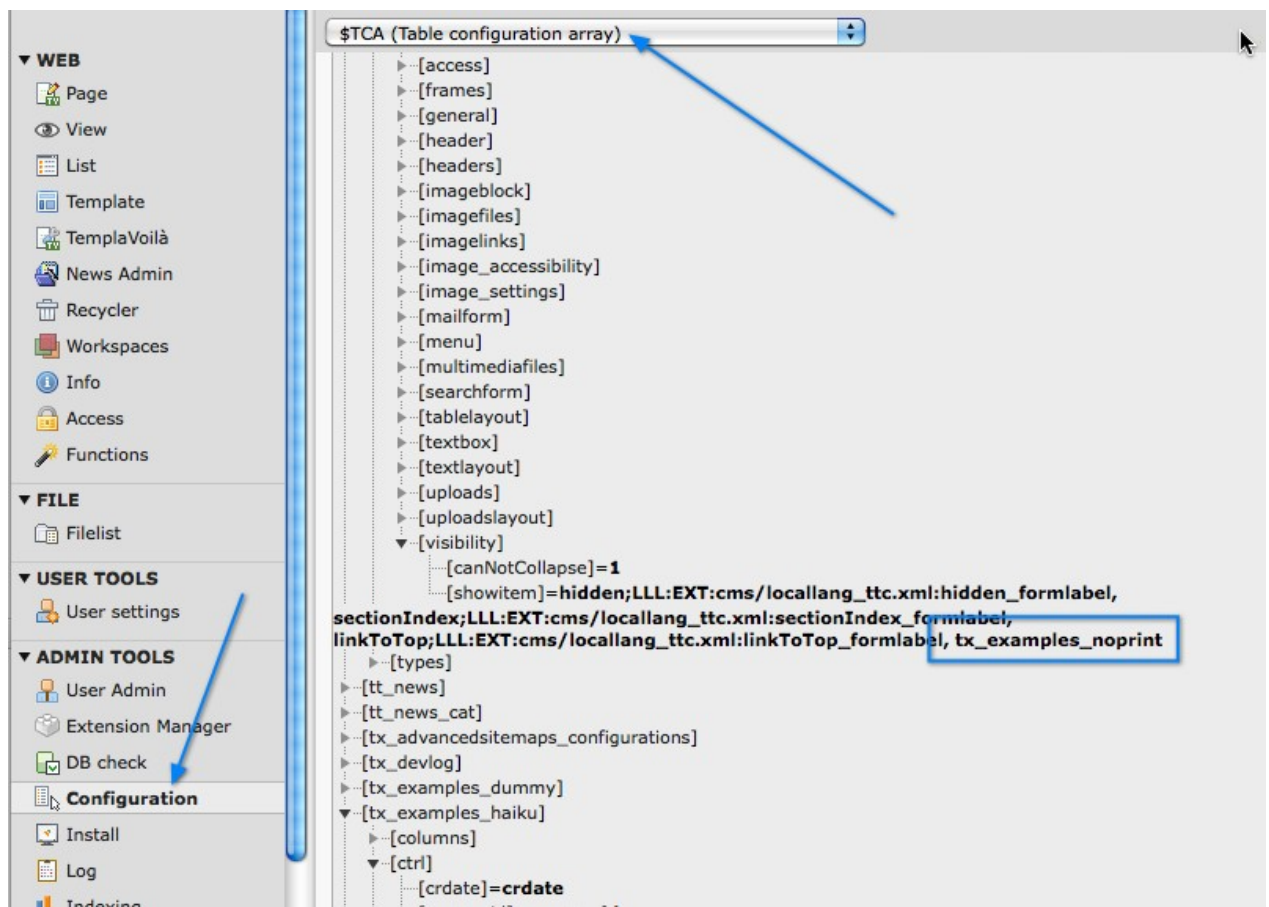
This will wrap a "div" tag with a "noprint" class around any content element that has its "No print" checkbox checked. The final step would be to declare the appropriate selector in the print-media CSS file so that "noprint" elements don't get displayed.

This is just an example of how the effect of the "No print" checkbox can be ultimately implemented. It is meant to show that just adding the field to the \$TCA is not enough.

## Verifying the \$TCA

You may find it necessary – at some point – to verify the full structure of the \$TCA in your TYPO3 installation. The Admin Tools > Configuration module makes it possible to have an overview of the complete \$TCA, with all customizations taken into account.

If you cannot find your new field, it probably means that you have made some mistake.



This view is also useful when trying to find out where to insert a new field, to explore the combination of types and palettes that may be used for the table that we want to extend.

# Appendix A – Skinning considerations

The new reference for skinning is the "Core Skinning Guidelines" manual. However the reference below has been kept as an appendix since it contains information that is not yet available elsewhere, even though a part of it is obsolete. So read it if you are interested but note that not everything may work as described below.

## Visual style of TCEforms

The design of the auto-generated forms in TYPO3 (typically referred to as "TCEforms") can be controlled down to fine detail. The fifth parameter in the \$TCA/types configuration is used for this.

The value consists of three integer pointers separated by a dash ("-"). The first parameter points to a color scheme, the second points to a style scheme for the form elements and the third points to the a border scheme for the table surrounding all form elements until the next border is defined.

The integer pointers refer to entries in the global \$TBE\_STYLES variable. Here the definitions for each pointer is configured.

### \$TBE\_STYLES entries related to TCEforms

The array \$TBE\_STYLES is a part of the skinning API in TYPO3 and therefore the full description is found in the [section about skinning](#). However the definition of the three entries related to TCEforms will be explained in detail here below.

#### Default integer pointers


The "[0-x]" values in the "Subkeys" column in the table below represents the integer pointers that you use in the types-configuration of \$TCA. You can set up any positive integer key you like, but TYPO3s core parts already implements the keys from 0-5 with a certain meaning which you are encouraged to follow as well:


Int. pointer	Title	Description
0	Default	Default index. Always used on main-palettes in the bottom of the forms.
1	Meta fields	Typically used for "Hidden", "Type" and other primary "meta" fields
2	Headers	For fields related to header information
3	Main content	For main content
4	Extras	For extra content, like images, files etc.
5	Advanced	For special content

Even if these pointers are used in the core of TYPO3 the default configuration as found in t3lib/stdtdb/tables.php includes only a definition of the default "0" (zero) pointer:

```
$TBE_STYLES = array(
    'colorschemes' => array(
        '0' => '#E4E0DB,#CBC7C3,#EDE9E5',
    ),
    'borderschemes' => array(
        '0' => array('border:solid 1px black;',5)
    )
);
```

**Reference table:**

Key	Sub-keys	Description
<b>colorschemes</b>	[0-x]	<p>This value is a comma separated list of five color/class definitions. The meaning of each color/class is defined as:</p> <p>[general cell] , [header cell] , [palette header cell] , [header label] , [palette header label]</p> <p>Each composite color/class value is split with a " " (vertical bar). The first part is a color value, typically setting a background color or font color. The second part is a class attribute value which will be set either for the table cell (td) or the span-tag around text</p> <p>For both color and class values these facts apply:</p> <ul style="list-style-type: none"> <li>● Omitting a color (blank value) will use the default value (from index "0" and if index "0" is not defined, based on the general mainColors in \$TBE_STYLES)</li> <li>● Setting a color value to dash ("-") will make it transparent (or just not set).</li> </ul> <p>Class attributes are set only if there was a class value set. There are no default class values.</p> <p><b>Example:</b></p> <pre>\$TBE_STYLES['colorschemes'] [0]='red,yellow,blue,olive,green';</pre>  <p><b>Example:</b></p> <pre>\$TBE_STYLES['colorschemes'][0]='- class-red,-  class-yellow,- class-blue,- class-olive,-  class-green';</pre> <p>This sets class attribute values instead. If you add this to the stylesheet you will get the same result as entering the real color values:</p> <pre>TABLE.typo3-TCEforms .class-red { background- color: red; } TABLE.typo3-TCEforms .class-yellow { background-color: yellow; } TABLE.typo3-TCEforms .class-blue { background- color: blue; } TABLE.typo3-TCEforms .class-olive { color: olive; } TABLE.typo3-TCEforms .class-green { color: green; }</pre>

Key	Sub-keys	Description
<b>styleschemes</b>	[0-x][elementKey]	<p>This value is the content of the "style" attribute of a form element (defined by "elementKey").</p> <p>If the value is prefixed "CLASS:" then it will set the class attribute instead to the value after the prefix.</p> <p>"elementKey" is the value of a [columns][field name][config] / TYPE (e.g. "text", "group", "check", "flex" etc.) or the string "all" (for defining a default value)</p> <p><b>Example:</b></p> <pre>\$TBE_STYLES['styleschemes'][0]['all'] = 'background-color:#F7F7F3;'; \$TBE_STYLES['styleschemes'][0]['check'] = '';</pre> <p>This (above) sets the background-color CSS attribute of all form elements <i>except</i> checkboxes!</p> <p><b>Example:</b></p> <pre>\$TBE_STYLES['styleschemes'][0]['all'] = 'CLASS: formField';</pre> <p>This will set the class attribute to 'formField' for all elements. The associated stylesheet could look like:</p> <pre>TABLE.typo3-TCEforms .formField { background- color: #F7F7F3; }</pre>
<b>borderschemes</b>	[0-x][key]	<p>This value defines the border style of the group of fields. Technically the group of fields are wrapped into a table.</p> <p>"key" is an index defining various values:</p> <ul style="list-style-type: none"> <li>● "0" : "style" attribute of the table wrapping the section</li> <li>● "1" : Distance in pixels after the wrapping table</li> <li>● "2" : "background" attribute of table wrapping the section: Reference to background image is relative to typo3/ folder (prefixed with -&gt;backPath)</li> <li>● "3" : "class" attribute of table wrapping the section.</li> </ul> <p><b>Example:</b></p> <pre>\$TBE_STYLES['borderschemes'][0][0] = 'border:solid 1px black;'; \$TBE_STYLES['borderschemes'][0][1] = 5; \$TBE_STYLES['borderschemes'][0][2] = '../typo3conf/freestyler_transp.gif';</pre> <p>This renders the form fields like this:</p>  <p>(Black border, the distance to the next section is 5 pixels and there is a background image)</p> <p><b>Example:</b></p> <pre>\$TBE_STYLES['borderschemes'][0]= array('',' ',' ','wrapperTable');</pre> <p>With an associated stylesheet you can get the same result (image not included):</p> <pre>TABLE.typo3-TCEforms .wrapperTable { border: 1px solid black; margin-top: 5px; }</pre>

See next chapter for examples of how to configure your TCEforms.

## Style pointers in the "types" configuration

The following is examples of how to use the styling features of TCEforms in real life. These examples will give you a chance to figure out how the features described in the reference table above is implemented.

In the examples below the \$TBE\_STYLES configuration includes the following:

```
$TBE_STYLES['colorschemes'] = Array (
    '0' => '#F7F7F3,#E3E3DF,#EDEDE9',
    '1' => '#94A19A,#7C8D84,#7C8D84',
    '2' => '#E4D69E,#E7DBA8,#E9DEAF',
    '3' => '#C2BFC0,#C7C5C5,#C7C5C5',
    '4' => '#B2B5C3,#C4C6D1,#D5D7DE',
    '5' => '#C3B2B5,#D1C4C6,#DED5D7'
);
$TBE_STYLES['styleschemes'] = Array (
    '0' => array('all'=>'background-color: #F7F7F3;border:#7C8D84 solid 1px;', 'check'=>'),
    '1' => array('all'=>'background-color: #94A19A;border:#7C8D84 solid 1px;', 'check'=>'),
    '2' => array('all'=>'background-color: #E4D69E;border:#7C8D84 solid 1px;', 'check'=>'),
    '3' => array('all'=>'background-color: #C2BFC0;border:#7C8D84 solid 1px;', 'check'=>'),
    '4' => array('all'=>'background-color: #B2B5C3;border:#7C8D84 solid 1px;', 'check'=>'),
    '5' => array('all'=>'background-color: #C3B2B5;border:#7C8D84 solid 1px;', 'check'=>'),
);
$TBE_STYLES['borderschemes'] = Array (
    '0' => array('border:solid 1px black;',5),
    '1' => array('border:solid 1px black;',5),
    '2' => array('border:solid 1px black;',5),
    '3' => array('border:solid 1px black;',5),
    '4' => array('border:solid 1px black;',5),
    '5' => array('border:solid 1px black;',5)
);
```

### Examples

First, lets look at a plain types-configuration which merely renders a list of fields:

```
'types' => Array (
    '0' => Array('showitem' => 'title;;1,photodate,description,images,fe_cruser_id')
),
```

It renders this form:

Now I modify the types config to include the fifth parameters (in red):

```
'types' => Array (
  '0' => Array('showitem' => 'title;;;1;;1--0,photodate;;;-4-,description;;;2-
0-,images;;;1--0,fe_cruser_id')
```

And this looks like:

To understand how the style pointers works, lets organize them into a table. This is the "types"-configuration string:

```
title;;;1;;1--0,photodate;;;-4-,description;;;2-0-,images;;;1--0,fe_cruser_id
```

Splitting this information into a table looks like this:

Fieldname	5th param:	'colorscheme' pnt:	'stylescheme' pnt:	'borderscheme' pnt:
title	1--0	1	[blank]	0
photodate	-4-	[blank]	4	[blank]
description	2-0-	2	0	[blank]
images	1--0	1	[blank]	0
fe_cruser_id	[blank]	[blank]	[blank]	[blank]

Explanation:

- "colorscheme" : The pointer is set to "1" for the first field ("title" field). This gives a green style (according to definitions in \$TBE\_STYLES['colorscheme'][1]) which is active until the "description" field is rendered. Here the pointer is changed to "2" which gives the yellow style. Immediately after the pointer is set back to "1" and that is active throughout the form.
- "stylescheme" : The pointer starts by being blank. Since no previous value is set, the pointer is implicitly "0" (zero) then. At the field "photodate" the pointer is set to "4" which means the style attribute gets the value "background-color: #B2B5C3;border:#7C8D84 solid 1px;" (according to the current configuration of \$TBE\_STYLES['stylescheme'][4]). This gives the blueish background of the date field. Immediately after the pointer is back at "0" again and that lasts for the rest of



the fields.

- "borderscheme" : The pointer is set to "0", then blank for three fields and then set to "0" again for the last two fields. In effect we get the form divided into two sections. As you can see setting the borderscheme pointer explicitly - *even if set to the same value!* - breaks up the form each time into a new section. Setting the first pointer to the default border scheme was actually not necessary but served to illustrate that the same border was applied twice.

It should also be clear now, that setting an empty pointer (blank string) will just let the former value pass through.

The three schemes are designed to go in pairs. It is most likely that all three pointers should be set each time you apply the fifth parameter value. Example:

```
'types' => Array (
  '0' => Array('showitem' => 'title;;;1;;1-1-1,photodate;;;2-2-2,description;;;3-3-3,images,fe_cruser_id;;;5-5-5')
),
```

# Appendix B – Performance considerations

This appendix contains some old musings of Kasper about performance when loading the full \$TCA. The content is not entirely up to date and some code constructs don't exist anymore, but the general meaning is still interesting.

## Loading the full \$TCA dynamically

You may load table descriptions dynamically (as needed) from separate files using the function `t3lib_div::loadTCA($tablename)` where `$tablename` is the name of the table you wish to load a complete description of.

Dynamic tables must always be configured with a *full* [ctrl]-section (and [feInterface] section if needed). That is, it must be represented by `$TCA[$table]['ctrl']`. If the table is dynamic, the value of [ctrl] [dynamicConfigFile] points to an includefile with the full array in.

The loadTCA-function determines whether a table is fully loaded based on whether `$TCA[$table][columns]` is an array. If it is found to be an array the function just returns - else it loads the table if there is a value for “dynamicConfigFile”

The table “pages” must not be dynamic. All others can be in principle. You can also define more than one table in a dynamicConfigFile - as long as the \$TCA array is correctly updated with table information it doesn't matter if a file contains configuration for more than the requested table - although the requested table should of cause always be configured, because it's expected to be. In fact there is not much error checking; The function loadTCA simply includes the file in blind faith that this action will fully configure the table in question.

## Locating places where t3lib\_div::loadTCA call is needed

To find places in your backend code where this should probably be implemented search for:

**"each(\$TCA)"** - This is potentially dangerous in a construction like this:

```
while(list($table,$config)=each($TCA))
```

where \$config would obtain non-complete content. Hopefully there are none left. Instead it should look like:

```
while(list($table)=each($TCA)) {
    t3lib_div::loadTCA($table);
    $config=$TCA[$table]
    ...
}
```

`[/?(palettes|types|columns|interface)?/]` (regex) - to find places where the palettes, types, columns and interfaces keys are used - which would require the whole array to be loaded!

It's recommended to always call the function `t3lib_div::loadTCA()` before using the non-[ctrl] sections of the \$TCA array. The function returns immediately if the table is already loaded, so the overhead should be small and the importance is great.

## Benchmarks on dynamic tables:

Module	tables.php with all configuration		Dynamic loading	
	Cache	No cache	Cache	No cache
<b>Web&gt;List (loads all)</b>	173 ms	322 ms	177 ms	328 ms
<b>Web&gt;Info (loads none)</b>	72 ms	174 ms	66 ms	136 ms

Benchmarks on a PIII/500 MHz Linux PHP4.1.2/Apache, 256 MB RAM. PHP-Cache is PHP-accelerator. All figures are parse times in milliseconds.

## Analysis:

What we see is, when showing a page in Web>List where all tables are loaded, the dynamic loading of tables includes a little overhead (177-173=4 ms) regardless of script-caching. This seems fair, probably due to file operations. It's also evident that the script-caching boosts the parsing considerably in both cases, saving approximately 150 ms in parse time!

The Web>Info module does not load any tables (at least not in the mode, this was tested). This is the whole point of all this - that the full table definitions are loaded only if needed (as they were by the Web>List module). Again the point of caching is clear. But the main thing to look at is, that the Web>Info module is loaded in 66/136 seconds (cache/non-cache) with dynamic loading (was later tested to 60/118 ms when tt\_content was not loaded by default) which is LOWER than if the whole tables.php was included (72/174 ms).

At this point the performance gain is not significant but welcomed. However the mechanism of dynamic loading of tables provides the basis for much greater number of tables in TYPO3. Testing 31 duplicates of the tt\_content table added to the default number of configured tables (total of 62 tables configured) gave this benchmark:

Module	Dynamic loading	
	Cache	No cache
<b>Web&gt;List (loads all)</b>	580 ms	1090 ms
<b>Web&gt;Info (loads none)</b>	67 ms	139 ms

This shows once again the work of the caching (1090-580 ms gained by PHPA) but clearly demonstrates the main objective of dynamic loading; The Web>Info module is not at all affected by the fact that 31 big tables has been added.

The serialized size of the \$TCA in this case was measured to approx 2MB. The total number of KB in table-definition PHP-files was approx. 1.7 MB.

Extreme tests of this configuration has also been done.

A duplicate of tt\_content was added x number of times and yielded these results:

Number of tt_content dupl.	Serialized size of \$TCA	Max size of httpd process (from "top")	Parse time of the included documents
100	5,9 MB	23 MB	380 ms
250	14,5 MB	52 MB	12000 ms
500	28,8 MB	100 MB	x

The configuration of tt\_content is approx. 52 kb PHP code. The testing was done just loading the content into \$TCA - no further processing. However serializing the \$TCA array (when that was tested) gave a double up on the amount of memory the httpd process allocated. This was to expect of course.

From this table we learn, that PHP does not crash testing this. However it makes not much sense to use 500 tables of this size. 250 tables might be alright and 100 tables is a more realistic roof over the number of tables in TYPO3 of the size of tt\_content!

Conducting the same experiment with a table configuration of only 8 kb with 9 fields configured (a reduced configuration for the tt\_content duplicate - which represents a more typical table) yielded these results:

Number of tables	Serialized size of \$TCA	Max size of httpd process (from "top")	Parse time of the included documents	Web>List listing
1	240 kB	12 MB	0 ms	174 ms (12 MB)
100	1,0 MB	12 MB	77 ms	550 ms (12 MB)
250	2,4 MB	12 MB	200 ms	1050 ms (12 MB)
500	4,7 MB	22 MB	450 ms	1900 ms (20 MB)

Number of tables	Serialized size of \$TCA	Max size of httpd process (from "top")	Parse time of the included documents	Web>List listing
1000	9,3 MB	33 MB	900 ms	5000 ms (34 MB)
2000	18,6 MB	51 MB	2000 ms	18000 ms (60 MB)