TemplaVoila!

Extension Key: **templavoila**

The content of this document is related to TYPO3

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

Table of Contents

Introduction

What does it do?

In short, TemplaVoila is an alternate templating enginge offering a new way of creating and working with design elements. On top of that it provides a new user interface in the backend (aka the “Page Module).

The extension “TemplaVoila” was developed by Kasper Skårhøj and Robert Lemke for a project in a large, french company, Dassault Systemes. TemplaVoila was the result of the innovation that followed some problem solving for the project. In particular the problems that TemplaVoila addresses are how to create more flexible page structures than currently known in TYPO3s concept of “columns”. Further, it integrates traditional templating on the content element level but with a far more flexible point-n-click style than seen before. Finally the development of TemplaVoila also lead to some enhancements of the TYPO3 Core, in particular a concept called FlexForms which allows TYPO3's backend to build hierarchical forms and store their content into XML structures

Sponsorship

The main part of this extension was kindly sponsored by Dassault Systémes, France. Thanks a lot for your incredible attitude towards Open Source and very generous support of TYPO3. Finalisation of version 1.0 was funded by the TYPO3 Association. Thanks to all our supporting members who enable us to develop!

About this manual

With the release of TemplaVoila 1.0 this manual still remains incomplete in some parts. But that doesn't mean that there's no documentation about TemplaVoila ... The following documents will help you getting started and find the essential information for your work:

- TemplaVoila Manual (this document) – This document always contains a complete reference of the configuration options you have. Currently it does not contain further information about setting up TemplaVoila sites, upgrading or migrating from non TemplaVoila sites. But at least an upgrading and migration part is planned.
- Futuristic Template Building – This is a large step-by-step tutorial which explains the most important parts of TemplaVoila and gets you started with your first TV website. At the time of this writing, some information in the tutorial need an update so they reflect the new features which were implemented through the last year. But still it's worth trying it to get an overview!
- Localization Guide – A document with an in-depth explanation of localization and translation with TYPO3 in general. It specifically demonstrates the localization / translation features of TemplaVoila. At the time of this writing (07.04.06) this document is not published at TYPO3.org yet but will be very soon.

Site developer's manual

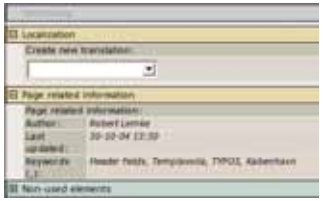
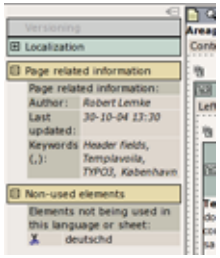
Meant for people developing TYPO3 websites. Does not require PHP knowledge.

Page / User TSconfig reference

The following TypoScript configuration can be used either as Page TSconfig or as User TSconfig.

mod.web_txtemplavoilaM1

Property:	Data type:	Description:	Default:
createPageWizard.fieldNames	list	With this option you may specify a selection of field names of the *pages* table to be displayed in the create-new-page-wizard. **Example:** `mod.web_txtemplavoilaM1.createPageWizard { fieldNames = hidden, title, author, description, abstract }` This will create an editing form like this: 	hidden, title, alias
sideBarEnable	boolean	Defines if the toolbar in the TemplaVoila Page Module is visible or not. By default it is visible as a row of tabs.	TRUE

Property:	Data type:	Description:	Default:
sideBarPosition	string	Defines the position of a toolbar within the TemplaVoila Page Module. Possible values: *toptabs, toprows, left* ***toptabs*** generates a toolbar on top of the page module's content with dynamic tabs which allow you to switch between the different option categories.  ***toprows*** also generates a toolbar on the top of the page, but uses rows which can be expanded and collapsed instead of tabs.  ***left*** instead creates a sidebar at left part of the page module. This sidebar may be shown and hidden by clicking at the little plus / minus sign at the upper right corner of the sidebar.  **Note:** The menu items which are available in this toolbar depend on the extensions you have installed as they provide the functionality.	toptabs
disableContainerElementLocalizationWarning	boolean	Container elements used with TemplaVoila should not be localized. Therefore a warning is displayed if <langDisable> is false for such data structures. If localization was enabled on purpose this warning will be misleading of course and can be disabled by this setting.	FALSE
disableContainerElementLocalizationWarning_warningOnly	boolean	Sometimes you might like to localize container elements with <langChildren> enabled. This is especially the case if the element is more than a container but also has content fields that need localization. The problem is that only the default language values of the reference fields (non-content fields) is recognized by TemplaVoila page module while in the frontend the rendering depends on inheritance and what other references someone might accidentally put into the reference fields of other languages! So, if you use localization for such mixed records, a) make sure inheritance is enabled (so for all languages the references set for default language is used) and b) that no references are localized (leave reference fields for other languages empty).	FALSE

Property:	Data type:	Description:	Default:
translationParadigm	string keyword	If set to “free” the Page module will act according to a translation paradigm called “Free” (opposite to “Bound”) where you use the Page DS <langDisable> to indicate whether or not a page can be localized and there localizations of default language records are linked into separate content structures provided by the data structure either as Inheritance or Separate. You should read the document “Localization Guide” which includes detailed information about these concepts.	
disableDisplayMode	list of keywords	In the “Bound” translation paradigm, you will see a selector box that allows you to filter which languages you see for editing. Here you can disable certain of the available options by setting keywords in this list. Options are: default, selectedLanguage, onlyLocalized	
recordDisplay_tables	list of table names	Comma-separated list of table name to shown in "Record list" tab. This feature requires TYPO3 version >= 4.0.5. Record list allows to edit any record on the page without a need to switch to list module. If this value is not set, record list will not be shown at all.	Not set
recordDisplay_maxItems	integer	Maximum number of records to display. If this values is not set, value from $TCA is used. If value in $TCA is not set, default is used.	10
recordDisplay_alternateBgColors	boolean	If set to 1, forces to alternate background colors for records	false

[tsconfig:mod.web_txtemplavoilaM1]

mod.web_txtemplavoilaM2

Property:	Data type:	Description:	Default:
templatePath	string	Path to directory inside fileadmin/ where templates can be found. Must be accessible by the users file mounts. **Example:** mod.web_txtemplavoilaM2.templatePath = template/main/	

[tsconfig:mod.web_txtemplavoilaM2]

Plugin “pi1” attributes

Property:	Data type:	Description:	Default:
TSconst.[constant-name]	string	Defining constants for Data Structure constants. see “<T3Data Struction> extensions” section	
dontInheritValueFromDefault			
childTemplate	string	Keyword which is used to look for a child-template record. By default “print” is a value you can set. This is also set automatically when “&print=1” is found in the URL The value is matched with the content of “rendertype” so if you want other values than “print” to be available you simply add new items to that selector box and use conditions in the TypoScript Template to detect the circumstance that should set this value. More explainations later	
disableExplosivePreview	boolean	If set, the popup information boxes in the frontend will not appear in FE preview mode.	FALSE
disableErrorMessages	boolean	If set, no error messages will be displayed in the frontend. **Example:** `plugin.tx_templavoila_pi1.disableErrorMessages = 1`	FALSE

[tsref:plugins.tx_templavoila_pi1]

Extension programmer's manual

Who should read this?

If you are an extension programmer and want to find out how to use certain parts of TemplaVoila or take influence on some processing in your own extensions, this is your part of the manual. This section requires PHP knowledge as well as some basic experience with extension programming in TYPO3.

User functions / hooks

Where it made sense, we have implemented some hooks every here and there in TemplaVoila in order to give extension programmers a chance to override or extend certain functionality. Just register your own function and you will take over the control or take influence on that part of TemplaVoila.

If you need to extend a certain part and don't find a way to include your own code, just get in touch with us, we might include some API to implement your own user defined function.

Generally, there are two ways of providing hooks, the ones using t3lib_div::getUserObj() and those using t3lib_div::callUserFunction(). Hooks going the ***getUserObj*** way require a **class name** while ***callUserFunction*** hooks accept a **class name** and **method name**. Here is an example of how to register your own function in both ways:

```
   // The getUserObject way:
$TYPO3_CONF_VARS['EXTCONF']['templavoila'][ sub_key ][subsub_key][] = 'my_class';

   // The callUserFunction way:
$TYPO3_CONF_VARS['EXTCONF']['templavoila'][ sub_key ][subsub_key][] = 'my_class->my_method';
```

Which type of hook was implemented, is specified in the column *type* in the reference below. It also states if only one or multiple userfunctions are allowed for that hook. In the latter case you'll have to add your class name (and method) to an **array of userfunctions**.

Hint: You should read the section about hooks in the *TYPO3 core APIs* document, which is available on TYPO3.org. And of course you should have a look at the source code where the hook is provided before you implement your own userfunction.

Sub key:	Sub-sub key:	Type:	Purpose / description:

cm1	eTypesConfGen	callUserFunction / single	"eTypes" are presets which are use in the click module (cm1) in order to create the field configuration of a data structure. While mapping you may choose between these eTypes, examples are "text", "image", "imagefixed", "ce" and so on.
			
			The "input" eType for example, results in this configuration within the data structure:
			
			`<TCEforms>`
			`  <config>`
			`  <type>input</type>`
			`  <size>30</size>`
			`  <eval>trim</eval>`
			`  </config>`
			`  <label>test</label>`
			`</TCEforms>`
			
			If you want to override the creation of this configuration for a certain eType, you may use eTypesConfGenUserfunctions to specify your user defined function.
			
			Provide a user function for the eType "input", you might specify something like this in your extension's page TSconfig:
			
			`$TYPO3_CONF_VARS['EXTCONF']['templavoila']['cm1']['eTypesConfGen']['input'] = 'tx_myClass->myMethod'`
			
			For more information on how to design your user function have a look at templavoila/cm1/index.php

cm1	eTypesExtraFormFields	callUserFunction / single	(Also see the explanation about eTypes above) Using this hook you may specify a user function which will render certain extra fields for certain eTypes in the mapping dialogues. One popular extra field is the object path for the TypoScriptObject eType. **Example:** `$TYPO3_CONF_VARS['EXTCONF']['templavoila']['cm1']['eTypesExtraFormFields']['input'] = 'tx_myClass->myMethod';`

Sub key:	Sub-sub key:	Type:	Purpose / description:
mod1	renderTopToolbar	callUserFunction / multiple	Use this hook if you want to output some HTML code at the very top of the Edit Page screen in the page module. This was hook was implemented for providing a custom toolbar related to the current page. **Example:** `$TYPO3_CONF_VARS['EXTCONF']['templavoila']['mod1']['renderTopToolbar'][] = 'tx_myClass->myMethod';`

mod1	renderPreviewContent	getUserObj / multiple	Use this hook if you want to render the preview of a custom cType or override the default preview of a certain cType. This is great if you want to provide a preview for your own plugins!
			
			Let's say you wrote a plugin called myext_pi1. Just create a new function your tx_myext_pi1 class and register it in $TYPO3_CONF_VARS (see above). Your own function would look like this:
			
			Example:
			`function renderPreviewContent_preProcess ($row, $table, &$alreadyRendered, &$reference) {`
			`  if (row['CType'] == 'list' && $row['list_type'] == 'myext_pi1') {`
			`  $content = '<strong>MyExt:</strong> '.htmlspecialchars('my custom preview');`
			`  $alreadyRendered = true;`
			`  return $reference->linkEdit($content, $table, $row['uid']);`
			`  }`
			`}`

mod1	renderFrameworkClass	getUserObj / multiple	This function contains the following hook: **renderFrameWork_preProcessOutput** Called just before the final content is compiled for displaying the framework in the edit page screen. An example for using this hook is adding another icon into the title bar of certain content elements or the page.
pi1	renderElementClass	getUserObj / multiple	This function contains the following hook: **renderElement_preProcessRow** Gives you the chance to modify the row currently being rendered for frontend output. One way of using is, is selecting a different template object for a flexible content element, based on certain conditions.

Adding items to the sidebar

[TODO: Explain how other extensions can easily add new items to the sidebar]

```
if (t3lib_extMgm::isLoaded('templavoila'))    {
    require_once (t3lib_extMgm::extPath('templavoila').'mod1/class.tx_templavoila_mod1_sidebar.php');
}
class tx_myext_templavoila_sidebar {
    function init() {
            // Create / get instances:
        $thisObj =& t3lib_div::getUserObj ('&tx_myext_templavoila_sidebar', '');
        $sideBarObj =& t3lib_div::getUserObj ('&tx_templavoila_mod1_sidebar', '');
            // Register sidebar item:
        $sideBarObj->addItem ('tx_myext_templavoila_sidebar_item1', $thisObj, 'renderItem_myext', 'My Extension', 50);
    }
    function renderItem_myext(&$pObj) {
            // Dummy output, just return the current page id:
        return $pObj->id;
    }
}



if (t3lib_extMgm::isLoaded('templavoila'))    {
    require_once (t3lib_extMgm::extPath($_EXTKEY).'class.tx_myext_templavoila_sidebar.php');
    tx_myext_templavoila_sidebar::init();
}
```

TemplaVoila API for extension developers

- Basic concept
- flexformPointer / flexformPointerString
  - sourcePointer / destinationPointer

<T3DataStructure> extensions

Introduction

TemplaVoila extends the Data Structure XML with a set of tags which defines two things related to TemplaVoila:

- **Mapping:** Definition of mapping rules, descriptions, sample data, and field type preset
- **Rendering:** Definition of TypoScript code, Object Path, processing flags and constants

<T3DataStructure> extensions for "<tx_templavoila>"

"Array" Elements:

Element	Description	Sub-elements
<[application tag]>	In this case the application tag is "<tx_templavoila>"	<title> <description> <tags> <sample_data> <sample_order> <eType> <TypoScriptObjPath> <TypoScript> <proc> <ruleConstants> <ruleRegEx> <ruleDefaultElements> <langOverlayMode>
<ROOT><tx_templavoila>	For <ROOT> elements in the DS	<title> <description> <pageModule>
<pageModule>	A bunch of config options which take influence on the rendering in the page module.	<displayHeaderFields> <titleBarColor>
<sample_data>	Sample data, defined in numeric array. Sample data is selected randomly from these options	<n[0-x]>
<sample_order>	For <section>s: Defines a set of array objects to display as sample data. Each value in this numerical array points to a fieldname in the object <el> array.	<n[0-x]>
<TypoScript_constants>		<[constant_name]>
<proc>	Processing options (during rendering)	<stdWrap> <int> <HSC>

"Value" Elements:

Element	Format	Description
<meta><sheetSelector>	string	Defining a file/class with PHP code to evaluation sheet selection in frontend. Its a getUserObject reference a la "EXT:user_myext/class.user_myext_selectsheet.php:&user_myext_selectsheet" where the class user_myext_selectsheet contains a function, selectSheet(), which returns the sheet key, eg. "sDEF" for default sheet. **Notice about using sheets in frontend rendering (pi1):** This feature is fairly advanced and still needs some development and documentation. Here are some points to observe: ● When sheets are defined the template also needs to be remapped! ● If no mapping exists for other keys than "sDEF" then they will default to use the mapping for "sDEF". Thus it can save you a little on mapping the same over and over again if all sheets use the same template. ● When using sheets the local processing XML also needs to be wrapped in eg. "<sheet><sDEF> .... </sheet></sDEF>" ● The selection of sheets should be careful to select only based on parameters that are safely cached. This can be done if parameters are known to be cHash protected - or if the page cache is disabled of course.
<title>	string	The title displayed in the mapping view
<description>	string	Mapping instructions / description, shown in mapping view.

Element	Format	Description
<tags>	string	commalist of tag rules. A tag rule is defined as [tagname]:[mapping-mode]:[attribute] **Examples are:** • table:outer,div,body:inner,td:inner • *:attr:href • a:attr:* • *:inner,a:attr:href,a:attr:src
<eType>	string	Value pointing to a TCEforms preset. Used for building of Data Structures with templavoila. Automatically set and controlled. This tag only used internally by the mapping tool.

<oldStyleColumnNumber>	integer	By setting this tag to an integer value (usually between 0 and 3), you define to which tt_content column number this field relates. This information is used by the list module, frontend editing and all other places which work with the older column paradigm.
		
		If you want to convert a pre-TemplaVoila site to a TemplaVoila site with the migration wizard you also have to make sure setting oldStyleColumnNumber tags for your content areas.
		
		Note: Each value can only be used once in a data structure and this usage makes only sense in page templates!
		
		Background information:
		Before TemplaVoila existed, the content on a page was arranged by assigning each content element to a certain column id. By default four columns were available: "Normal" (id=0), "Left" (id=1), "Right" (id=2) and "Border" (id=3).
		
		Some parts of TYPO3 and some extensions are not aware of the different way TemplaVoila structures content. If you create or move a content element with the List module, the element possibly does not appear at the position where you expect it, because the list module doesn't know which content area reflects the "Normal" column.
		
		Example:
		`<T3DataStructure>`
		`  <ROOT>`
		`  <el>`
		`  <field_maincontent>`
		`  <tx_templavoila>`
		`  <oldStyleColumnNumber>0</oldStyleColumnNumber>`
		`...`

<TypoScriptObjPath>	string	TypoScript object path pointing to a TypoScript Template Content Object which will render the content represented by the element. Very useful if you want to insert a menu which is defined by eg. "lib.myMenu" in the TypoScript Template of a website.

Element	Format	Description

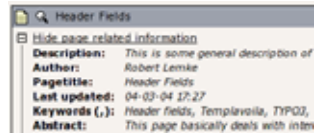
<TypoScript>	string	TypoScript content.
		
		Constants can be inserted
		• which are defined locally in <TypoScript_constants>, see below
		• In the TypoScript template of the website; In the Setup field you can set constants as properties (first level only) in "plugin.tx_templavoila_pi1.TSconst" - those can be inserted by {$TSconst.[constant name]} in the <TypoScript> data!
		
		General example:
		
		`<TypoScript>`
		`<![CDATA[`
		
		`10 = USER`
		`10.userFunc = user_3dsplm_pi2->testtest`
		`10.imageConfig {`
		`  file.import.current = 1`
		`  file.width = 100`
		`}`
		
		`]]>`
		`</TypoScript>`
		
		Access other fields in the same data structure:
		
		`<TypoScript>`
		`  10 = TEXT`
		`  10.field = field_myotherfield`
		`</TypoScript>`
		
		Display the page title:
		
		`<TypoScript>`
		`  10 = TEXT`
		`  10.data = page:title`
		`</TypoScript>`

<[constant_name]>	string	A local TypoScript constant which can be inserted by {$[constant_name]} in <TypoScript> (see above)
		
		Instead of setting a plain value you can also reference object path values from the sites TypoScript template by inserting a value like "{$lib.myConstant}". **Notice**, the value will come from the Templates Setup field.
		
		Example:
		`<TypoScript_constants>`
		`  <backGroundColor>red</backGroundColor>`
		`  <fontFile>{$_CONSTANTS.resources.fontFile}</fontFile>`
		`</TypoScript_constants>`
		
		Here "_CONSTANTS.resources.fontFile" must be an object path with a value in the TypoScript template of the website!

<int>	boolean, 0/1	Pass through intval() before output
<HSC>	boolean, 0/1	Pass through htmlspecialchars() before output

<stdWrap>	string	StdWrap properties as TypoScript, eg:
		
		`<proc>`
		`  <stdWrap>`
		`  trim = 1`
		`  br = 1`
		`  </stdWrap>`
		`</proc>`

Element	Format	Description
<langOverlayMode>	string, keyword	Setting the mode for content fallback when <meta><langChildren> and other languages are used in flexforms. Normally inheritance from default language is enabled by default and globally disabled by the TypoScript setting "dontInheritValueFromDefault" if needed. However through the Data Structure and TO / Local Processing XML you can overrule this per-field by this keyword. In any case it only affects values from other languages than default and only if <langChildren> is enabled (thus using "vDEF" and sibling fields named "vXXX" for localization). **Keywords:** **ifFalse** - Content is inherited if it evaluates to false in PHP (meaning that zero and blank string falls back) **ifBlank** - Content is inherited if it matched a blank string (trimmed) **never** - Content is never inherited from default language! **removeIfBlank** - If the value of this field is blank then the *whole group* of fields (element) is removed! This is a way of removing single elements for localizations in <langChildren>=1 constructions instead of inheriting content from default language. **[default]** - If no keyword matches it uses the global mode.

<displayHeaderFields>	string	A list of page-related fields which should be displayed as a header in the edit page view of the page module. By now, only table "page" is allowed / makes sense.
		
		Note: This tag only takes effect when used in the top-level <tx_templavoila> section, ie. one level below the <ROOT> tag.
		
		 
		
		Example:
		`<T3DataStructure>`
		`  <ROOT>`
		`  <tx_templavoila>`
		`  <pageModule>`
		`  <displayHeaderFields>`
		`  pages.keywords`
		`  pages.mycustomfield`
		`  </displayHeaderFields>`
		`  </pageModule>`
		`...`

<titleBarColor>	color	If you want to help your editors determining which data structure is used for the page they are currently working on, you may specify a color by using this tag. The title bar at the very top of the edit page screen will be displayed in that color.
		
		You may use any value which is allowed in CSS (ie. "red" as well as "#FC2300" etc.)
		
		Note: This tag only takes effect when used in the top-level <tx_templavoila> section, ie. one level below the <ROOT> tag.
		
		 
		
		Example:
		`<T3DataStructure>`
		`  <ROOT>`
		`  <tx_templavoila>`
		`  <pageModule>`
		`  <titleBarColor>orange</titleBarColor>`
		`...`

Extensions to tags in the Data Structure		

Element	Format	Description
<[field-name]><type>	string	In the Data Structure only “array” or blank makes sense. However for TemplaVoila there is additional values possible, “attr” and “no_map”. This is a complete TemplaVoila related overview of the <type> / <section> meanings: • <type>array</type> = Renders an array or objects • <type>array</type> + <section>1</section> = Renders a section which must contain other array-types (without <section> set!) • <type>attr</type> = The object is mapped to a HTML tag *attribute*. • <type>[blank]</type> = The object is mapped to a HTML tag *element*. • <type>no_map</type> = The object is not mappable (only editing in FlexForms eg.)

Sheets and TemplaVoila

TemplaVoila is compatible with definition of sheets. In that case a sheet <ROOT> element is shown in the mapping structure containing each sheet as <ROOT> elements under it.

Accessing “parent” record from DS TypoScript

Note: This feature is experimental and can be changed in future. All users are warned: use it at your own risk! This notice will be removed when this feature becomes stable.

To access “parent” record from “tt_content” or “pages” table in the <TypoScript> section of a field, developer can use special registers. These registers defined only when <TypoScript> section is executed. The following example shows how to use these registers:

```
<TypoScript>
10 = TEXT
10.data = register:tx_templavoila_pi1.parentRec.uid
10.wrap = “uid” field of parent record is |
</TypoScript>
```

Thus any field of parent record is defined as **tx_templavoila_pi1.parentRec.XXX** register, where XXX is replaced by a field name from the corresponding table.

Notice that these registers are undefined for static data structures because static data structures do not have associated parent record. If reference to **tx_templavoila_pi1.parentRec.XXX** appears in the static data structure, result is undefined.

Case: Templating API for use in plugins

Introduction

You can use TemplaVoilas API for templates in your own plugins if you like. As an example of this, lets look at how the “mininews” extension works:

The mininews extension has three displays of content:

- An archive listing of all news in the archive, including a search box and links for browsing to the next page if there are more than 20 news or so.
- A detail display which shows a single news item in full
- A frontpage teaser listing showing the three most recent news with “read more” links.

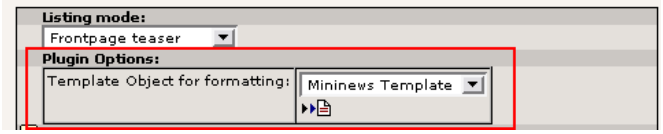
Each of these displays are by default rendered by hardcoded HTML in the plugin. The hardcoded HTML is designed to be sufficient in most cases since you can style it all by CSS styles. Thus you might not need to make an alternative template!

However if you would like to restructure the output more than you can do by the CSS styles on the default HTML you can create a TemplaVoila template. The mininews extension supports this.

Selecting the alternative Template Object

Plugin Options - a FlexForm

The way you select an alternative template is by selecting the template object in the plugin options which are rendered by the standard flexform element for plugins:



For those interested this form is generated by the “flexform_ds.xml” configuration in the “mininews” extension:


```
<T3DataStructure>
   <meta>
      <langDisable>1</langDisable>
   </meta>
   <ROOT>
      <type>array</type>
      <el>
         <field_templateObject>
            <TCEforms>
               <label>LLL:EXT:mininews/locallang_db.php:tt_content.pi_flexform.select_template</label>
               <config>
                  <type>select</type>
                  <items>
                     <n0>
                        <n0></n0>
                        <n1>0</n1>
                     </n0>
                  </items>
                  <foreign_table>tx_templavoila_tmplobj</foreign_table>
                  <foreign_table_where>
                     AND tx_templavoila_tmplobj.pid=###STORAGE_PID###
                     AND tx_templavoila_tmplobj.datastructure="EXT:mininews/template_datastructure.xml"
                     AND tx_templavoila_tmplobj.parent=0
                     ORDER BY tx_templavoila_tmplobj.title
                  </foreign_table_where>
                  <size>1</size>
                  <minitems>0</minitems>
                  <maxitems>1</maxitems>
               </config>
            </TCEforms>
         </field_templateObject>
      </el>
   </ROOT>
</T3DataStructure>
```

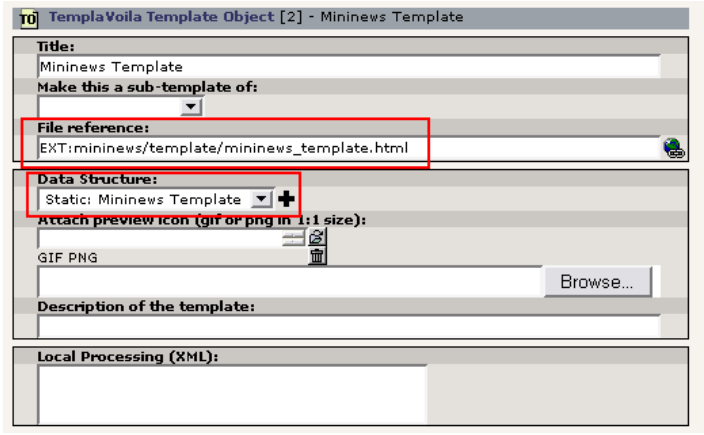
Further “mininews” enables the configuration by setting these lines in the “ext_tables.php”

```
$TCA['tt_content']['types']['list']['subtypes_addlist'][$_EXTKEY.'_pi1']='tx_mininews_frontpage_list;;;;
1-1-1,pi_flexform';
t3lib_extMgm::addPiFlexFormValue($_EXTKEY.'_pi1', 'FILE:EXT:mininews/flexform_ds.xml');
```

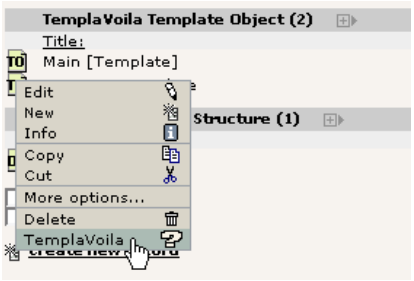
This is in fact all you have to do to select an alternative template. Of course the question is - what is a Template Object and how to we create one? This is answered next.

Creating a Template Object

This is done in the “Storage folder” which should have been configured to the website. Here you create a new Template Object an select the *Data Structure* that the mininews extension provides:



After having done this you close the document here, click the Template Object icon again and select “TemplaVoila”:



Subsequently you can begin the mapping of the Data Structure to the template file (here the example file “mininews/template/mininews_template.html” is used) and after that process you will see something like this:

Notice in particular how each of the three templates are found in the *same* Data Structure as *sheets* where the highest root element named “ROOT of multitemplate” represents the three sheets inside:

After the mapping process is complete the alternative template is in place.

The big questions now are:

- How can I define a data structure for my plugin just like “mininews” has done?
- How can I use any alternative template represented by a Template Object inside my plugin?

These questions are answered next.

Setting up a Data Structure XML file for Template Objects mapping

In “mininews” the data structure that is used for the mapping of templates is found in the file “mininews/template_datastructure.xml”. The contents look like this:

```
<T3DataStructure>
   <sheets>
   <!-- The Archive configuration is so large that we have put it into it's own file,
        and references it from here: -->
      <sArchive>EXT:mininews/template_datastructure_arc.xml</sArchive>

      <!-- Single display of mininews items: -->
      <sSingle>
         <ROOT>
            <tx_templavoila>
               <title>SINGLE DISPLAY</title>
                  <description>Select the HTML element which is the container of the
                  single display of a news article:</description>
                  <tags>div:inner</tags>
            </tx_templavoila>
            <type>array</type>
            <el>
               <field_date>
                  <tx_templavoila>
                     <title>Date</title>
                     <description>News date</description>
                     <tags>*:inner</tags>
                     <sample_data>
                        <n0>6th August 10:34</n0>
                        <n1>29/12 2003</n1>
                     </sample_data>
                  </tx_templavoila>
               </field_date>
               <field_header>
```

```
            <tx_templavoila>
               <title>Header</title>
               <description>Header field.</description>
                  <tags>*:inner</tags>
                  <sample_data>
                     <n0>People on mars!</n0>
                     <n1>Snow in Sydney</n1>
                  </sample_data>
            </tx_templavoila>
         </field_header>
         <field_teaser>
            <tx_templavoila>
               <title>Teaser</title>
               <description>Teaser field.</description>
               <tags>*:inner</tags>
               <sample_data>
                  <n0>Capthurim Chanaan vero genuit Sidonem primogenitum et
                        Heth Iebuseum quoque </n0>
               </sample_data>
            </tx_templavoila>
         </field_teaser>
         <field_bodytext>
            <tx_templavoila>
               <title>Bodytext</title>
               <description>Bodytext field</description>
               <tags>*:inner</tags>
               <sample_data>
                  <n0><![CDATA[
                     <p><strong>Filii Ham Chus et Mesraim Phut et Chanaan</strong> filii autem
Chus Saba et Evila Sabatha et Rechma et Sabathaca porro filii Rechma Saba et Dadan Chus autem genuit
Nemrod iste coepit esse potens in terra Mesraim vero genuit Ludim et Anamim et Laabim et Nepthuim
Phethrosim quoque et Chasluim de quibus egressi sunt Philisthim et.</p>
                     <p>Capthurim Chanaan vero genuit Sidonem primogenitum et Heth Iebuseum
quoque et Amorreum et Gergeseum Evheumque et Aruceum et Asineum Aradium quoque et Samareum et Ematheum
filii Sem Aelam et Assur et Arfaxad et Lud et Aram et Us et Hul et Gothor et Mosoch Arfaxad autem genuit
Sala qui et ipse genuit Heber porro Heber nati sunt duo filii nomen uni Phaleg quia in diebus eius
divisa est terra et nomen fratris eius Iectan Iectan autem genuit Helmodad et Saleph et Asermoth et Iare
Aduram quoque et Uzal et Decla Ebal etiam et Abimahel et Saba necnon et Ophir et Evila et Iobab omnes
isti filii Iectan Sem Arfaxad Sale.</p>
                  ]]></n0>
               </sample_data>
            </tx_templavoila>
         </field_bodytext>
         <field_url>
            <type>attr</type>
            <tx_templavoila>
               <title>"Back" URL.</title>
               <description>Map to a-tags href-attribute of the link back to
                  archive listing.</description>
               <tags>a:attr:href</tags>
               <sample_data>
                  <n0>javascript:alert('You click this link!');</n0>
               </sample_data>
            </tx_templavoila>
         </field_url>
      </el>
   </ROOT>
</sSingle>


<!-- Frontpage display of a few mininews teaser items: -->
<sFrontpage>
   <ROOT>
      <tx_templavoila>
         <title>FRONTPAGE LISTING</title>
         <description>Select the HTML element which is the container of the
                  frontpage listing display of a news articles:</description>
         <tags>div:inner</tags>
      </tx_templavoila>
      <type>array</type>
      <el>
         <field_fpListing>
            <type>array</type>
            <section>1</section>
            <tx_templavoila>
               <title>Archive Listing container</title>
               <description></description>
               <tags>div,table:inner</tags>
            </tx_templavoila>
            <el>
               <element_even>
                  <type>array</type>
```



```
                  <tx_templavoila>
                     <title>Element Container, Even</title>
                     <description></description>
                     <tags>*:outer</tags>
                  </tx_templavoila>
                  <el>
                  <field_date>
                     <tx_templavoila>
                        <title>Date</title>
                        <description>News date</description>
                        <tags>*:inner</tags>
                        <sample_data>
                           <n0>6th August 10:34</n0>
                           <n1>29/12 2003</n1>
                        </sample_data>
                     </tx_templavoila>
                  </field_date>
                  <field_header>
                     <tx_templavoila>
                        <title>Header</title>
                        <description>Header field.</description>
                        <tags>*:inner</tags>
                        <sample_data>
                           <n0>People on mars!</n0>
                           <n1>Snow in Sydney</n1>
                        </sample_data>
                     </tx_templavoila>
                  </field_header>
                  <field_teaser>
                     <tx_templavoila>
                        <title>Teaser</title>
                        <description>Teaser field.</description>
                        <tags>*:inner</tags>
                        <sample_data>
                           <n0>Capthurim Chanaan vero genuit Sidonem primogenitum et
                                 Heth Iebuseum quoque </n0>
                        </sample_data>
                     </tx_templavoila>
                  </field_teaser>
                  <field_url>
                     <type>attr</type>
                     <tx_templavoila>
                        <title>"MORE" URL.</title>
                        <description>Map to a-tags href-attribute of the link pointing
                              to the archive!</description>
                        <tags>a:attr:href</tags>
                        <sample_data>
                           <n0>javascript:alert('You click this link!');</n0>
                        </sample_data>
                     </tx_templavoila>
                  </field_url>
               </el>
            </element_even>
         </el>
      </field_fpListing>
   </el>
   </ROOT>
</sFrontpage>
</sheets>
</T3DataStructure>
```

If you study this long codelisting you will find that it only configures the DS for the templates “SINGLE DISPLAY” and “FRONTPAGE LISTING” - the “ARCHIVE LISTING” is actually found in another file referred to by this line:

```
<sArchive>EXT:mininews/template_datastructure_arc.xml</sArchive>
```

You can study the contents of this file by yourself.

Anyways, the question remains: How does mininews configure that this XML file should be available for Template Objects to point to? This is found in ext_tables.php:

```
// Adding datastructure for Mininews:
$GLOBALS['TBE_MODULES_EXT']['xMOD_tx_templavoila_cm1']['staticDataStructures'][]=array(
    'title' => 'Mininews Template',
    'path' => 'EXT:'.$_EXTKEY.'/template_datastructure.xml',
    'icon' => '',
```

```
    'scope' => 0,
);
```

$_EXTKEY contains the value “mininews” as usual in a ext_tables.php file for an extension.

By this configuration the DS will appear in the Data Structure selector box:

At this point we have:

- FlexForm configuration needed to select template record in the “Insert Plugin” type Content Element
- A Data Structure (DS) in an XML file which can be used for mapping a template HTML-file to the DS.
- You should also have an example template-HTML file which can demonstrate the mapping of your DS.

All that is left is to actually use the template in the plugin.

Getting the value of the Plugin FlexForm field

In the mininews plugin class we first need to detect if a Template Object record is pointed at and if so make sure it is used.

Detecting the Template Object (TO) record

In the “mininews/pi1/class.tx_mininews_pi1.php” file the class contains these two variables:

```
        // TemplaVoila specific:
    var $TA='';                     // If TemplaVoila is used and a TO record is found, this array will
be loaded with Template Array.
    var $TMPLobj='';            // Template Object
```

Later, in the listView function you find this initialization which detects the record. Comments below

```
   1: function listView($content,$conf)    {
   2:
   3:         // Init FlexForm configuration for plugin:
   4:     $this->pi_initPIflexForm();
   5:
   6:         // Looking for TemplaVoila TO record and if found, initialize template object:
   7:     if (t3lib_extMgm::isLoaded('templavoila'))    {
   8:         $field_templateObject = $this->pi_getFFvalue($this->cObj-
>data['pi_flexform'],'field_templateObject');
   9:         if (intval($field_templateObject))    {
  10:             $this->TMPLobj = t3lib_div::makeInstance('tx_templavoila_htmlmarkup');
  11:             $this->TA = $this->TMPLobj->getTemplateArrayForTO(intval($field_templateObject));
  12:             if (is_array($this->TA))    {
  13:                 $this->TMPLobj->setHeaderBodyParts($this->TMPLobj->tDat['MappingInfo_head'],$this-
>TMPLobj->tDat['MappingData_head_cached']);
  14:             }
  15:         }
  16:     }
```

- Line 4 initializes the “pi_flexform” field in $this->cObj->data. This will convert the field from being a string with the XML data to being an array with the same XML converted to PHP array by t3lib_div::xml2array()
- Line 7 checks if TemplaVoila is loaded -which it must be of course!
- Line 8 requests the value of “field_templateObject” in the FlexForm content of “pi_flexform”
- Line 9 sees in that value is an integer - which means it points to a Template Object record “uid”
- In line 10 we create an instance of the “tx_templavoila_htmlmarkup” class which will be our API for merging our *data* from mininews with the *template* from the Template Object record.
- Line 11 loads the Template Array from the TO pointed to by $field_templateObject.
- Line 12 checks if the Template Array was set - this is the case if there was mapping information found in the TO.
- Line 13 will set possible header sections if any should be defined in the TO.

Now, in the rest of the mininews class we can just check if $this->TA is an array and if so use templavoilas API for merging data and template. This is shown next.

Merging Data with Template markup

In order to not make this too lengthy I will just cut out some examples.

Repeated list rows

The first example is how to accumulate content for list rows. This is basically done by a loop, traversing over the elements and for each iteration calling an API function in TemplaVoila with two arguments, the appropriate part of the ->TA variable (Template Array = cached template markup) and an array with the mininews data.

```
   1:     // Create list of elements:
   2: $elements='';
   3: while($this->internal['currentRow'] = mysql_fetch_assoc($res))    {
   4:     $elements.=$this->TMPLobj->mergeDataArrayToTemplateArray(
   5:         $this->TA['sub']['sArchive']['sub']['field_archiveListing']['sub']['element_even'],
   6:         array(
   7:             'field_date' => $this->getFieldContent('datetime'),
   8:             'field_header' => $this->pi_list_linkSingle($this->getFieldContent('title'),$this-
>internal['currentRow']['uid'],1),
   9:             'field_teaser' => nl2br(trim(t3lib_div::fixed_lgd($this-
>getFieldContent('teaser_list'),$this->conf['frontPage.']['teaserLgd'])))
  10:         )
  11:     );
  12: }
```

In this listing you can see that the array with data from mininews has three keys, “field_date”, “field_header” and “field_teaser”. These corresponds with three elements found in the Data Structure XML for the “ARCHIVE LISTING” template:

```
  10:                 <field_archiveListing>
  11:                     <type>array</type>
  12:                     <section>1</section>
  13:                     <tx_templavoila>
  14:                         <title>Archive Listing container</title>
  15:                         <description></description>
  16:                         <tags>div,table:inner</tags>
  17:                     </tx_templavoila>
  18:                     <el>
  19:                         <element_even>
  20:                             <type>array</type>
  21:                             <tx_templavoila>
  22:                                 <title>Element Container, Even</title>
  23:                                 <description></description>
  24:                                 <tags>*:outer</tags>
  25:                             </tx_templavoila>
  26:                             <el>
  27:                                 <field_date>
  28:                                     <tx_templavoila>
  29:                                         <title>Date</title>
  30:                                         <description>News
date</description>
  31:
  32:                                         <tags>*:inner</tags>
  33:                                         <sample_data>
                                                <n0>6th August
10:34</n0>
  34:
  35:                                             <n1>29/12 2003</n1>
  36:                                         </sample_data>
  37:                                     </tx_templavoila>
  38:                                 </field_date>
  39:                                 <field_header>
  40:                                     <tx_templavoila>
  41:                                         <title>Header</title>
  42:                                         <description>Header
field.</description>
  43:
  44:                                         <tags>*:inner</tags>
  45:                                         <sample_data>
  46:                                             <n0>People on mars!</n0>
  47:                                             <n1>Snow in Sydney</n1>
  48:                                         </sample_data>
  49:                                     </tx_templavoila>
  50:                                 </field_header>
  51:                                 <field_teaser>
```

```
  50:                                                     <tx_templavoila>
  51:                                                             <title>Teaser</title>
  52:                                                             <description>Teaser
field.</description>
  53:                                                             <tags>*:inner</tags>
  54:                                                             <sample_data>
  55:                                                                     <n0>Capthurim Chanaan
vero genuit Sidonem primogenitum et Heth Iebuseum quoque </n0>
  56:                                                             </sample_data>
  57:                                                     </tx_templavoila>
  58:                                             </field_teaser>
```

This is a part of the Data Structure which is nested inside of

```
<T3DataStructure><sheets><sArchive><ROOT><el>
```

I point this out because you can see the logic of the variable

```
$this->TA['sub']['sArchive']['sub']['field_archiveListing']['sub']['element_even']
```

in the code listing from this. Basically, if you substitute “sub” with “el” you can almost read that this variable will contain the markup for

```
<T3DataStructure><sheets><sArchive><ROOT><el><field_archiveListing><el><element_even>

$this->TA['sub']['sArchive']['sub']['field_archiveListing']['sub']['element_even']
```

Putting it all together

After having accumulated the list rows (and some other stuff) the values on the outer levels are also composed into a similar API call whose output is finally returned:

```
   1:     // Wrap the elements in their containers:
   2: $out = $this->TMPLobj->mergeDataArrayToTemplateArray(
   3:         $this->TA['sub']['sArchive'],
   4:         array(
   5:             'field_archiveListing' => $elements,
   6:             'field_browseBox_cellsContainer' => $br_elements,
   7:             'field_searchBox_sword' => htmlspecialchars($this->piVars['sword']),
   8:             'field_searchBox_submitUrl' => htmlspecialchars(t3lib_div::getIndpEnv('REQUEST_URI')),
   9:             'field_browseBox_displayRange' => $rangeLabel,
  10:             'field_browseBox_displayCount' => $this->internal['res_count']
  11:         )
  12:     );
  13:
  14: return $out;
```

This time you will see that the accumulated content of the list rows ($elements) is added to the key “field_archiveListing”. For all the other fields you can look them up in the DS as well:

```
...
 105:
 106:                     <!--
 107:                             Defining mappings for the search box:
 108:                     -->
 109:                     <field_searchBox_submitUrl>
 110:                             <type>attr</type>
 111:                             <tx_templavoila>
 112:                                     <title>Search form action</title>
 113:                                     <description>URL of the news-search; Map to the action-attribute
of the search form.</description>
 114:                                     <tags>form:attr:action</tags>
 115:                                     <sample_data>
 116:                                             <n0>javascript:alert('Hello, you pressed the search
button!');return false;</n0>
 117:                                     </sample_data>
 118:                             </tx_templavoila>
 119:                     </field_searchBox_submitUrl>
 120:                     <field_searchBox_sword>
 121:                             <type>attr</type>
 122:                             <tx_templavoila>
 123:                                     <title>Search word field</title>
 124:                                     <description>Search word; Map to the forms input-fields value-
attribute.</description>
 125:                                     <tags>input:attr:value</tags>
 126:                                     <sample_data>
 127:                                             <n0>Strawberry Jam</n0>
 128:                                             <n1>Jack Daniels</n1>
```



```
 129:                                             <n2>Flowers</n2>
 130:                                     </sample_data>
 131:                             </tx_templavoila>
 132:                     </field_searchBox_sword>
 133:
 134:                     <!--
 135:                             Defining mappings for the browse box, display note:
 136:                     -->
 137:                     <field_browseBox_displayRange>
 138:                             <tx_templavoila>
 139:                                     <title>Range</title>
 140:                                     <description>Map to position where "x-y" should be outputted
(showing which records are displayed)</description>
 141:                                     <tags>*:inner</tags>
 142:                                     <sample_data>
 143:                                             <n0>1-10</n0>
 144:                                             <n1>20 to 30</n1>
 145:                                     </sample_data>
 146:                             </tx_templavoila>
 147:                     </field_browseBox_displayRange>
 148:                     <field_browseBox_displayCount>
 149:                             <tx_templavoila>
 150:                                     <title>Count</title>
 151:                                     <description>Map to position where the total number of found
records should be outputted.</description>
 152:                                     <tags>*:inner</tags>
 153:                                     <sample_data>
 154:                                             <n0>123</n0>
 155:                                             <n1>3402</n1>
 156:                                     </sample_data>
 157:                             </tx_templavoila>
 158:                     </field_browseBox_displayCount>
...
```

Thats all!

FAQ: Frequently Asked Questions

Subpages don't inherit datastructure / template object

Q: *In the frontend, pages will render if I select a datastructure and a template object, but if I define some DS/TO for a page's subpages, an error message appears: Couldn't find a Data Structure set for table/row XXX. Please select a Data Structure and Template Object first.*

A: Make sure that you have configured the frontend plugin (templavoila_pi1) correctly in your TypoScript template. The following configuration will produce the error described above:

```
page = PAGE
page.typeNum = 0
page.10 < plugin.tx_templavoila_pi1
```

Instead use the following code and have a look at the configuration section of this manual:

```
page = PAGE
page.typeNum = 0
page.10 = USER
page.10.userFunc = tx_templavoila_pi1->main_page
```

Q: *I made “Content Elements” field, added content elements there but nothing is displayed in FrontEnd. What should I do now?*

A: You should check if you have <TypoScriptObjectPath> entry in that field inside your DS record. If it exists, TemplaVoila will not show content. This issue exists in TemplaVoila versions up to and including 1.0 but will be fixed (removed) in future versions. Inspect DS record and remove any occurrences of <TypoScriptObjectPath> from the content element fields. Do not forget to clear cache before checking results.

Q: *I made a new site but frontend is always empty!*

A: You forgot to add templates from “CSS static content” extension in the template record.