Maquettage avec TemplaVoila

Tables des Matières

Extension Pin Map - Utilisation.....1
Introduction.....1
Création du fond de carte.....1
Création de l'afficheur.....2
Création des points de la carte.....2

Templating sous typo3 avec l'extension templa voilà :

I – Réalisation du template HTML – Mise en place des zones de contenu.

A) Résumé sur les outils que l'on peut utiliser : PhotoShop - Dreamweaver/GoLive

B) Mettre en place des zones claires

II – Quelques notions générales sur le fonctionnement de Templa Voilà

En quelques lignes….

Toutes ces notions permettent de bien comprendre le niveau de séparation en le contenu et le contenant .

A) notion de Datastructure

Une « Data structure » DS est une représentation XML structurant les contenu.

Celle ci permet de paramétrer un ensemble de champ de contenu, offrant à l'utlisateur les formulaires de saisie adapté à l'utilisateur finale. Cette structure XML est montée sous forme de nœud permettant d'extraire, de placer ceux ci de manière indépendante.

B) Notion de template Object

Un « template object » TO est une mise sous forme XML la structure HTML issue du fichier d'entrer mis en place sous templa voilà..

Cet object permet de pouvoir placer des donner ou des nœud de donné de La DS dans toutes zones (déterminé par les balises HTML).

C) Notion de « template page »

Un template page est un template principal pour une ou plusieurs pages, il contient un TO issue d'un fichier template HTML complet et un DS modélisant ses données. Il est obligatoirement utilisé pour la présentation des pages en template de plus haut niveau.

Il est necessaire pour faire l'afichage primaire à l'écran client et afficher les différentes zones et charte graphique. Attention

plusieurs template page peuvent être défénit au sein d'un site afin d'avoir plusieurs présentation graphique différentes ou d'avoir plusieurs structure de donnée différentes ou les deux.

D) Notion de « flexible content »

La flexibme content est un sous template templa voilà, il contient un TO issue d'une parti d'unn fichier HTML et une DS modélisant sa structure de donnée.

Ce sous template va être ensuite comme un nouveaux type pour les contenu et pourra être placée au sein même d'un template de plus haut c'est a dire un « template page ».

III - Prérequis pour le fonctionnement

Typo3 historiquement possède un moteur de template basé sur un language (resemblant au XML, mais lors de sa création le XML démarrait juste et n'étais pas valider et donc encore moin répandu). Aves le projet de site internet le concepteur lui – même Kasper développa pour se projet une extension de templating. Cette extension révolutionnaire, complètement XML permet de faire de la réalisaition de template avec des structures de contenu et des structure de contenant permettant une véritable séparation contenant contenu.

Cette nouvel génération de templating étant sous forme d'extension celle ci est complètement séparée du corprs de typo3 et demande donc un prérequis apres installation pour son fonctionnement.

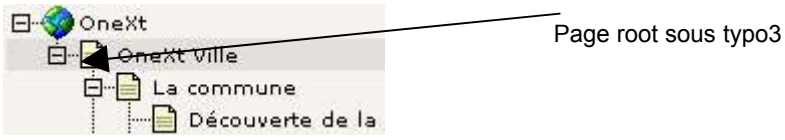
A) Installation de l'extension static_info_tables

Installation de cette extension permet à templa voilà de faire les correspondance entre les différents code de (région, langue, monnaie) de typo3 et les codes officiels ISO, UTF8 ect…

B) Code TS necessaire à l'appel de TV

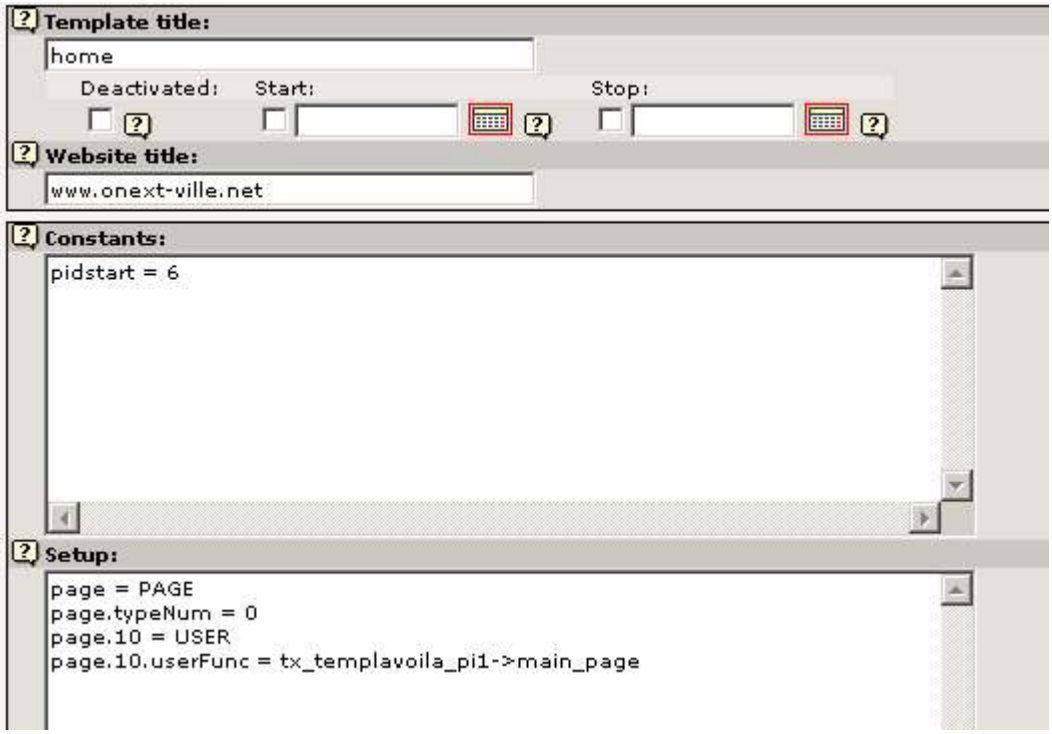
Code TS permettant d'appeler TV. Celui ci est à mettre dans un gabarit le setup d'un gabarit TS, qui doit être placée dans la page « root » de l'arborescance typo3.

Etablir une page root :



Exemple de code typo script à insérer dans le gabarit « root » :

Création d'un gabarit dans la page root.



C) Mise en place du sysfolder, object récipient des objetc templavoila (TO et DS)

1 – créer un espace objet récipient des objets TV

2 – indiquer grace au champ « enregistrement général » une liaison avec la page récipient créer préalablement.

D) Interraction avec les oject de contenu typo3 :

Sous typo3 pour la réalisation des templates et l'appel des conetnu situé dans la base de données, il y a une boite à outils qui est un ensemble de code typoscript prédéfinis. On les appels les static templates, il permettent au développeurs et intégrateurs de l'utlisé de manière simple afin d'intégrer rapidement des contenu et fonctionnalités dans la charte graphique.

2 possibilités :

i. utilisation des static template (tt_content et styles.content)
ii. utilisation de l'extension CSS styles content get

IV – Intégrer un template HTML principal avec templa voilà

A) Comment liés le template HTML à templa voilà
   a. Liaison template html -> template object
   b.
B) Mapping des zones

   a. L'objet « Contenair for element »
   b. Présentation des différents types de zones prédéfinie
   c. Le type « Content »
   d. Le placement dans les balises du template (inner/outer)
   e. Intégration de zone de type « TypoScript Object Path »

C) Intégration des éléments issue des TypoScript

a. Exemple sur des menus

Exemple d'un menu simple en TS menu vertical ou chaque item est encapsulé par un div qui indique une classe:

Menu peut être soit mis directement dans le template principal ou dans un sous template.

b. Exemple sur des liens spécifiques (Impression – Date)

c. Exemple sur des objets Typo Script issue des extensions

```
monmenu = HMENU
monmenu {
    special = directory
    special.value = id_page_démarrage
    1 = TMENU
    1 {
                NO.linkWrap = <div class= 'menu_niv1'>|</div>
                ACT = 1
                ACT.linkWrap = <div class= 'menu_n1-act'>|</div>
     }
     2 = TMENU
     2 {
          NO.linkWrap = <div class= 'menu_niv2'>|</div>
                 ACT = 1
                 ACT.linkWrap = <div class= 'menu_n2-act'>|</div>
    }
}
monmenu.wrap = <div class='menu'>|</div>
```

MINI TUTORIAL : Mise en place d'un template simple avec Templa Voilà.

1 – Créer un template HTML simple avec peu de tableaux imbriqué, en utilisant les balises préconiser par le W3C, le XHML et respectant les standard d'accécibilité. La feuille de style CSS associé doit aussi être réaliser préalablement.

2 – Uploder dans fileadmin ou dans un répertoire de fileadmin le fichier HTML

3 – Mettre en place les prérequis :

b) Aller dans le module « extension Manager »

c) Créer un gabarit dans la page home ou root ou premiere page du site

d) Mettre le Typo Script suivant dans le gabarit créer si dessus

4 – Liés le template HTML avec templa voilà

a) Aller dans fileadmin

b) Cliquer pour faire aparaitre le menu contextuel sur le fichier html puis cliquer sur l'option Templa Voilà

c) Vous arriver sur la fenêtre de mapping

5 – Maaping des zones avec templa Voilà

a) créer le map CO root :

b) créer une première zone

c) créer une zone contenu

d) créer une zone header

e) créer une zone typoScript Object path

f) mapping des tag du header de la page HTML

g) Enregistrement de la data Structure et du template object

Description de la data Structure XML pour les développeur

<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.
<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. Never mind...
<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.

<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 “plugins.tx_templavoila_pi1.TSconst” - those can be inserted by {$TSconst.[constant name]} in the <TypoScript> data!
		
		Example:
		`<TypoScript>`
		`<![CDATA[`
		
		`10 = USER`
		`10.userFunc = user_3dsplm_pi2->testtest`
		`10.imageConfig {`
		`  file.import.current = 1`
		`  file.width = 100`
		`}`
		
		`]]>`
		`</TypoScript>`

Element	Format	Description

<ruleConstants>	string	A kind of mapping table for abstracting CTypes into “constants”. In this particular case, constants mean a single character (a, b, c ...) which will be used as a placeholder for the CType in the regular expression defining the rule.
		
		Example:
		
		`<ruleConstants>`
		`  a = text`
		`  b = templavoila_pi1,3`
		`  c = textpic`
		`</ruleConstants>`
		
		That means: The character “a” will be used as a token for the “text” content element type, the character “b” stands for a flexible content element with the datastructure having the uid “3” and finally “c” is used as a “text with image” CType.

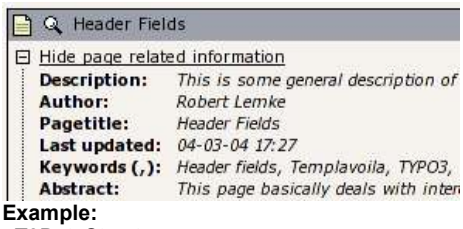
<ruleRegEx>	string	Defines a regular expression which describes a rule for the order of content elements which apply to a certain element. You have to use “constants” (ie. letters like “a” or “d”) in your expression. Each constant reflects a certain CType (content element type). See <ruleConstants> for more information about constants.
		
		Example:
		
		`<ruleRegEx>`
		`  (ab)*b|bac`
		`</ruleRegEx>`
		
		If the constants from the above <ruleConstants> example are used, this rule will accept::
		
		- Either one text and one flexible content with uid 3 once or more followed by exactly one flexible content with uid 3
		- Or one flexible content with uid 3, followed by one text element, followed by one text with image element.

<ruleDefaultElements>	string	Defines which elements are automatically created when the parent element (ie. a page using this datastructure if it's a page template or a flexible content element) is created.
		
		Note: By now this only works with the Create New Page Wizard. But will be implemented for other ways of creating pages / elements as well.
		
		Example:
		
		`<ruleDefaultElements>`
		`  bac`
		`</ruleDefaultElements>`

<[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.)