Modern Template Building, Part 1 [FR]

Extension key: **doc_tut_templselect**

Tables des Matières

Notes de traduction

Gabarit : contenu Typo3 constituant la maquette du site. Elle est constituée de TypoScript et peut s'appuyer sur une maquette HTML statique. Le terme Anglais utilisé est Template. Cependant le document original utilise le terme template aussi bien pour les gabarits TypoScript que pour les maquettes HTML statiques. Le terme technique Template correspondant aux maquettes TypoScript a donc été remplacé au sein de ce document par Gabarit. Nb : dans la version française de l'interface d'administration de Typo3 3.5.0, le terme utilisé est Gabariat bien que son caractère français soit discutable.

Maquette : dans ce document la maquette fait généralement référence à la maquette statique HTML qui contiendra des balise spécifiques (###BALISE###)qui seront remplacées par du contenu conformément aux instructions du gabarit.

Introduction

Ce dont il est question.

Cette extension est un tutoriel complet sur la construction d'un site Web à l'aide d'un CMS (Typo3) basé sur ungabarit (une maquette) HTML.

Ce site web que vous allez construire ressemblera à cela :

Ce site est composé d'une zone contenant le menu dynamique (gauche) et d'un contenu également dynamique (droite) – le reste du contenu est un design statique provenant de la maquette HTML.

But :

Le but de ce tutoriel est de vous donner l'état de l'art de la création des sites Typo3. Il vous permettra de mettre le pied à l'étrier rapidement et vous vous donner une compréhension globale des éléments impliqués dans la création d'un site web avec Typo3.

Si vous trouvez ce tutoriel trop long et souhaitez quelque chose de plus court, nous vous conseillons de choisir un autre CMS car un outil aussi puissant et complexe que Typo3 ne pourra être expliqué dans un document de quelques pages. De plus ce tutoriel ne fait que tracer les grandes lignes de l'ensemble.

Le fait que Typo3 ne vous coûte rien financièrement ne signifie nullement qu'il ne vous demandera pas de temps d'apprentissage ! Il demande un certain temps d'apprentissage et un certain investissement – comme ses concurrents commerciaux. Soyez prévenus. Piloter un avion demande un peu de talent et du temps pour apprendre. Heureusement, ce tutoriel vous mettra permettra un décollage aussi rapide que possible.

Niveaux :

Ce tutoriel est divisé en quatre sections.

Les bases – Une introduction pour les novices couvrant la construction de sites web avec Typo3, les maquettes, le TypoScript et les Content Objects (cObjects). Toute personne souhaitant développer un site avec Typo3 devrait êtrefamilière avec les concepts développés dans cette partie.

Partie 1: Intégration d'une maquette HTML – Cette partie s'adresse spécifiquement aux webdesigners ayant une bonne connaissance du HTML avec quelques connaissances techniques.

Partie 2: Créer un Sélecteur de Maquettes – Cette partie s'adresse aux développeurs web possédant une bonne connaissance du PHP, du SQL et des concepts de programmation en général.

Partie 3: Etendre le schéma d'accès interne – pour les développeurs avancés Typo3/PHP.

Note : les parties 2 et 3 se trouvent dans un autre document, dans l'extension "doc_tut_templselect2"

Vous pouvez bien évidemment aller directement à la partie qui vous intéresse. Cependant, si vous suivez ce tutoriel de bout en bout, vous constaterez que les différentes parties se suivent et vous guideront pas à pas.

L'extension :

Tous les fichiers de ce tutoriel sont contenus dans une extension Typo3. Les extensions contiennent habituellement des ressources et des scripts qui permettent d'étendre les fonctionnalités de Typo3. Si vous installez l'extension de ce tutoriel, vous constaterez qu'elle n'interagit pas avec le coeur de Typo3. Elle sert simplement à installer les fichiers nécessaires pour suivre ce tutoriel sur votre serveur.

Afin de suivre ce tutoriel, commencez par installer le dummy-package (Voir la partie*Les Bases*) puis importez l'extension "doc_tut_templselect" depuis TER (TYPO3 Extension Repository) en utilisant l'EM (Extension Manager). Vous disposerez ainsi de tous les fichiers sur votre serveur lorsque vous en aurez besoin.

Ce document peut également être consulté en ligne ou téléchargé sous format SXW (Openoffice.org Writer).

Prix :

Vous n'avez pas à payer pour la lecture de ce document. Cependant, une semainecomplète de préparation et d'écriture à son concepteur, Kasper Skårhøj. Ce travail n'a pas été rétribué, Si vous ou votre entreprise le trouvez utile et s'il vous permet de concevoir des sites plus performants pour vos clients, songez à faire un faire un don. Une simple congratulation pour le travail effectué ne nourrit malheureusement pas son homme. Sinon, il se pourrait que ce tutoriel soit le dernier.

Les Bases

Introduction

Cette section expliquera rapidement aux débutants le fonctionnement du frontend Typo3, ce que sont les gabarits, les "content objects", une maquette HTML. Si vous connaissez bien ces bases, vous pouvez sauter cette section et aller directement à la suivante : "Intégration d'une maquette HTML". Cependant vous devrez le dummy-package et y créer quelques pages. Ceci est expliqué dans cette section.

Niveau et pré requis :

Public: Tous les développeurs débutant en Typo3 indépendamment de leurs compétences dans d'autres domaines.

Certaines parties font référence à des technologies comme SQL, HTML, CSS et PHP. Le contenu est généralement un peu technique puisqu'il traitera de l'installation du dummy-package et expliquera les bases dumoteur de templates de Typo3.

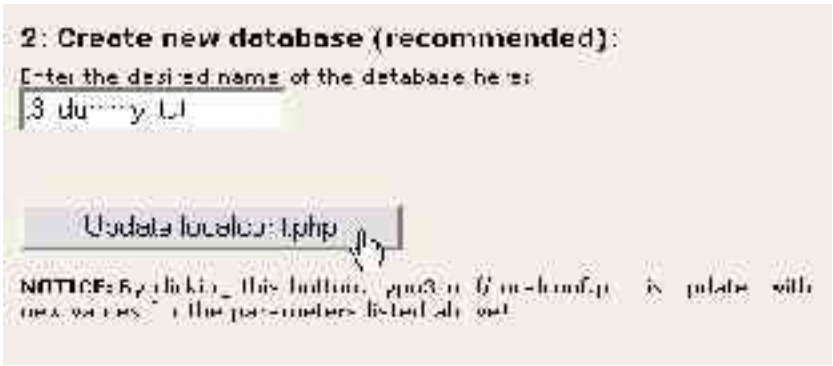
Débutons - installation du "dummy" package ("*tout en un pour les nuls*")

Afin de débuter rapidement, nous allons installer une structure vide et une base de données pour Typo3. La meilleur façon de procéder est d'utiliser le "dummy-package". Un "package" contient les sources générales de Typo3 plus des fichiers pour un site local et éventuellement un contenu de base de données, l'ensemble forme un site web Typo3 complet destiné à une utilisation particulière. Dans le cas du "dummy-package", le site web sera vide, cependant il sera particulièrement utile pour débuter un nouveau projet.

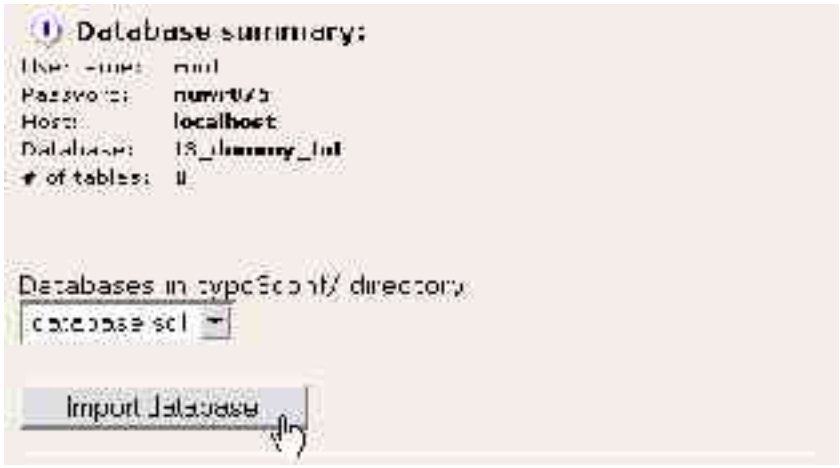
Donc, prenez le dummy-package et installez-le sur votre serveur. Après l'avoir dézippé, ouvrez la page index.php dans un navigateur et vous laissez vous guidez par les différentes étapes de l'assistant d'installation.

Entrez les information concernant la base de données :

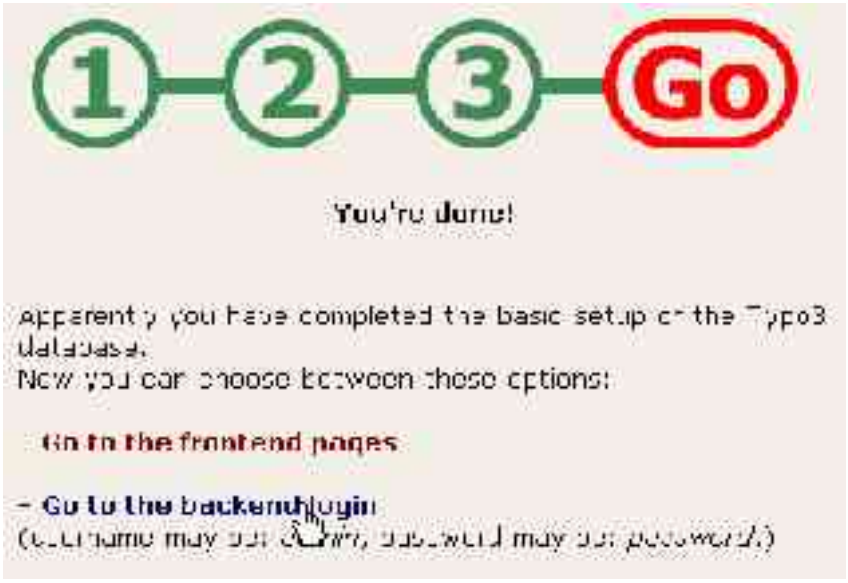
Créer une nouvelle base de données :



Importer la dummy database, "database.sql":



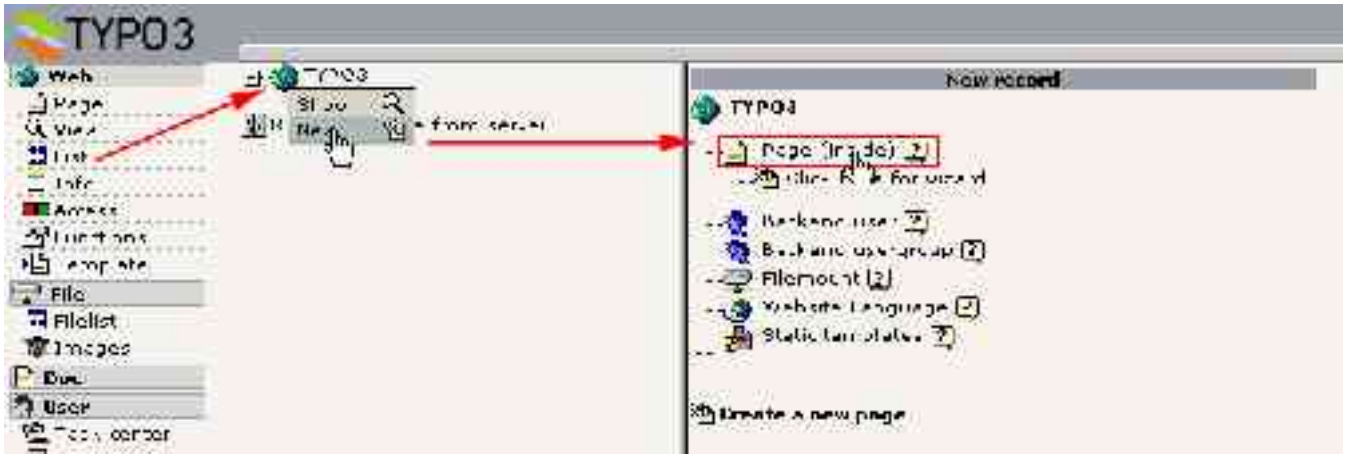
Aller sur le backend (interface d'administration):



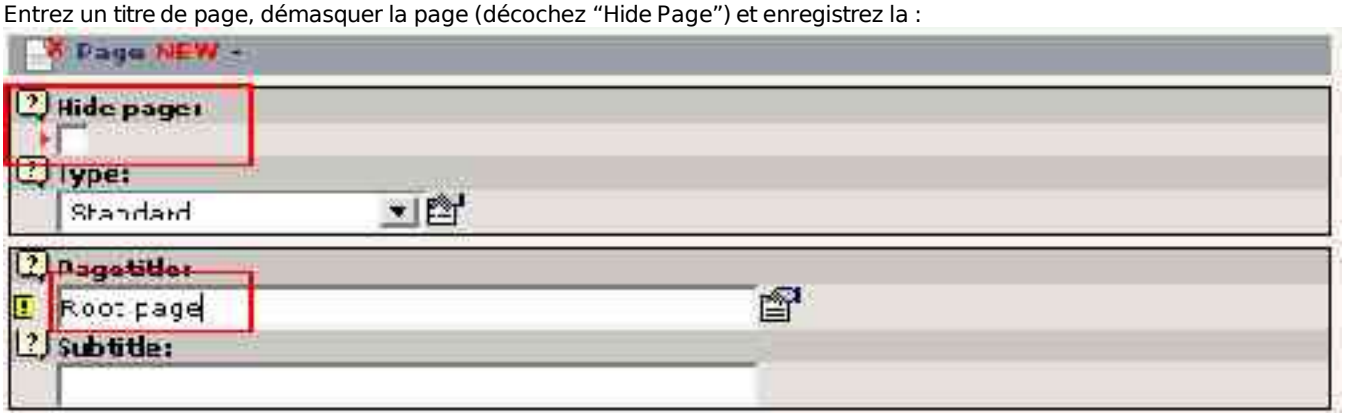
Afin de suivre ces étapes, il est indispensable que le serveur Web puisse écrire dans le fichier localconf.php (vous serez avertis si tel n'est pas le cas) et devez disposer du nom de l'utilisateur et de son mot de passe pour la base de données.

Créer l'arborescence de pages :

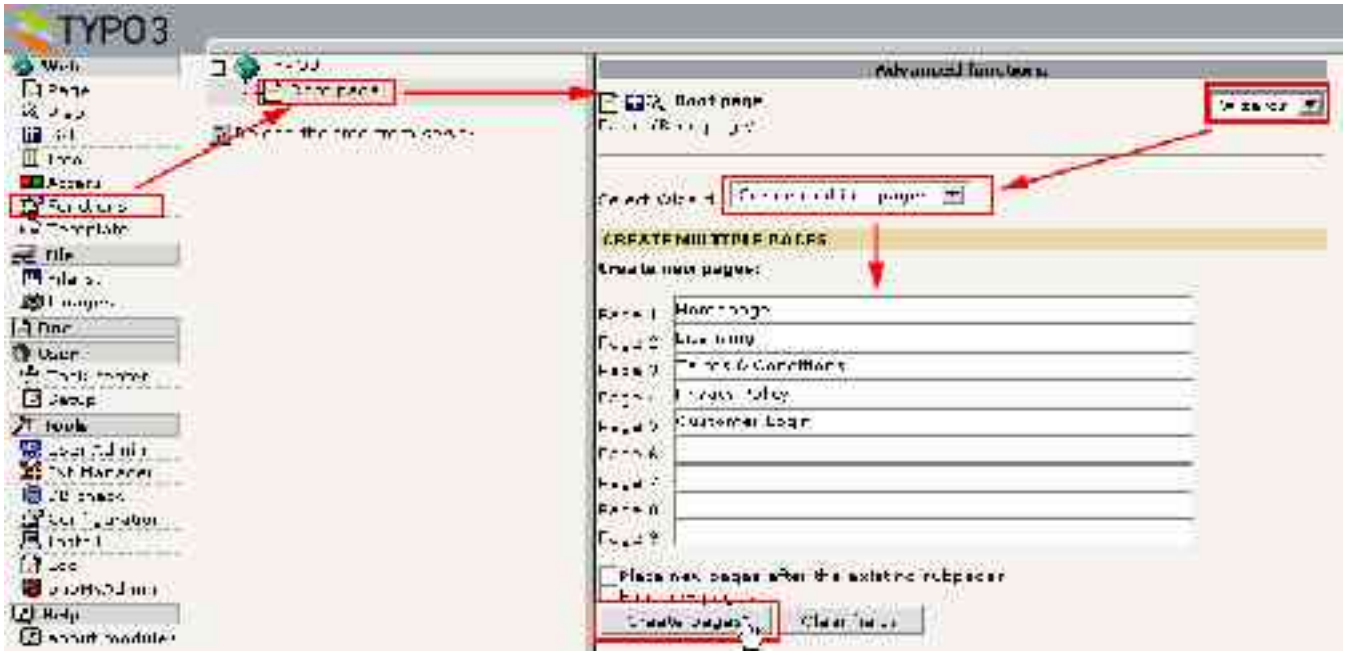
La première chose à faire maintenant est de créer l'arborescence des pages (la structure du site) que nous allons utiliser dans ce tutoriel. Une bonne structure de page doit avoir au moins 2 niveaux.Néanmoins, nous devons dans touts les cas commencer par une page racine :



Entrez un titre de page, démasquer la page (décochez “Hide Page”) et enregistrez la :

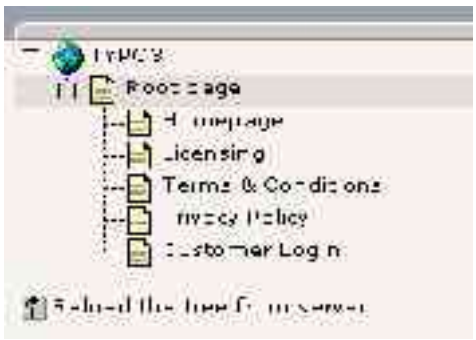


Vous pourriez procéder de cette manière pour créer l'ensemble de vos pages, cependant il existe un assistant qui vous facilitera la tâche. Allez dans le module “Functions”, sélectionnez les assistants (menu déroulant à droite : “wizards”) puis l'assistant “Create Multiple Pages”. Entrez plusieurs titres de pages puis validez avec le bouton “Create Page”.

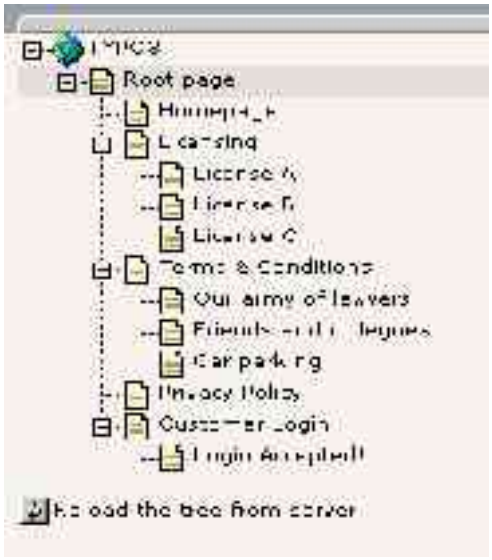


Attention, l'assistant “Create Multiple Pages” impose que l'extension “wizard_crpages”soit installée et activée !

Vous devriez maintenant disposer d'une arborescence de pages comme suit :

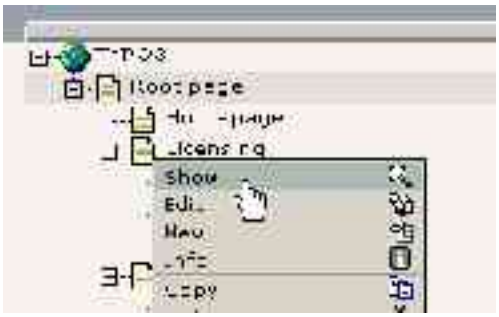


Maintenant créez quelques pages supplémentaires afin d'obtenir ceci :



L'arborescence et les Gabarit

Cliquez sur une icône de page et sélectionner “Show” :



Le résultat de la visualisation (“Show”) de la page “Licensing” sera un écran ressemblant à ceci :

La page “Licensing” a l'identifiant (uid=5) et est visualisé en ajoutant le paramètre “id=5”à l'appel de la page index.php dans le Frontend.

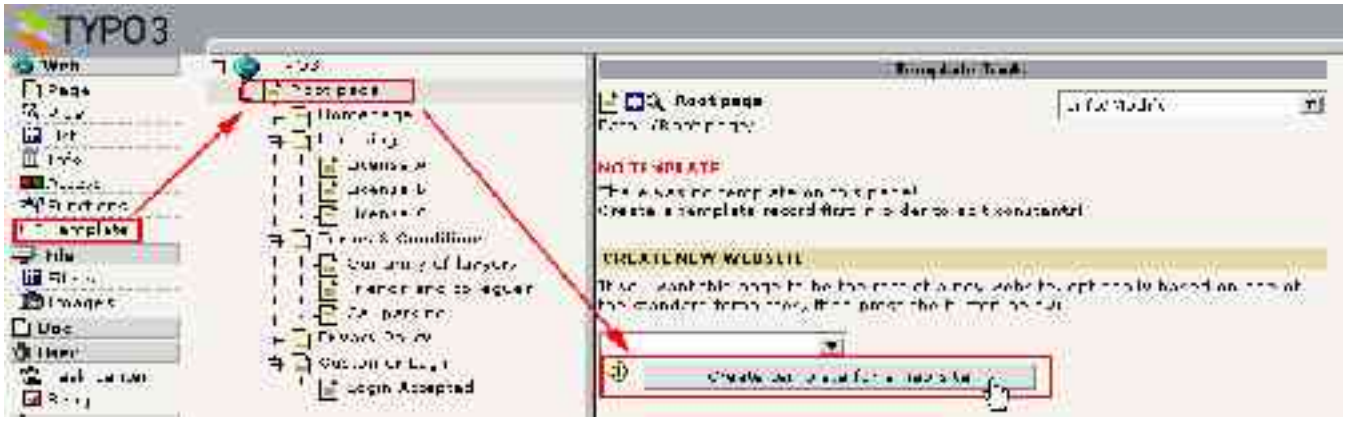
Le message affiché est “No template found”. Ceci signifie que le moteur de frontend n'a pu trouver aucune maquette dans la *rootline* (la chaîne de page de la page actuelle (id=5) à la page racine du site). Ceci demande quelques explications. Typo3 est un CMS multi-site. Cela signifie que vous pouvez avoir plusieurs sites hébergés dans une même arborescence (une même structure) de pages. Chaque site dans cette arborescence doit avoir une page racine (*root page*). La caractéristique de *root page* est positionné dans la maquette de la page racine en cochant la case «Rootlevel». Lorsque la valeur *Rootlevel* est vraie pour une page, cela signifie «A partir d'ici et en dessous, commence un nouveau site Web» .

L'expression *root line* est parfois appelé le chemin de la page. Supposons que la page "Licence A" dans mon arborescence à le chemin suivant : " TYPO3 > Root page > Licensing > License A ". Si nous envoyons l'id de la page "Licence A"au script

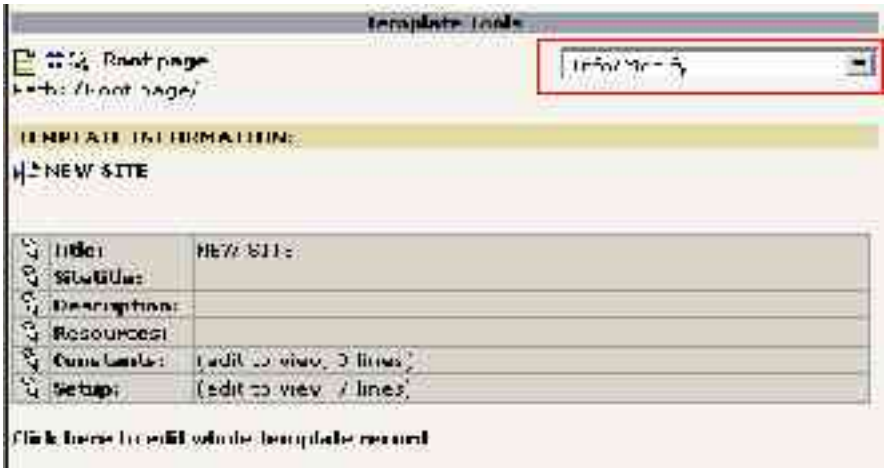
index.php, le moteur de template cherchera dans le chemin*(rootline)* depuis la page "licence A" jusqu'à la page racine (Root Page) pour un contenu de type Gabarit. C'est ce gabarit qui décidera comment toute cette branche de l'arborescence sera affichée.

Créer un gararіat

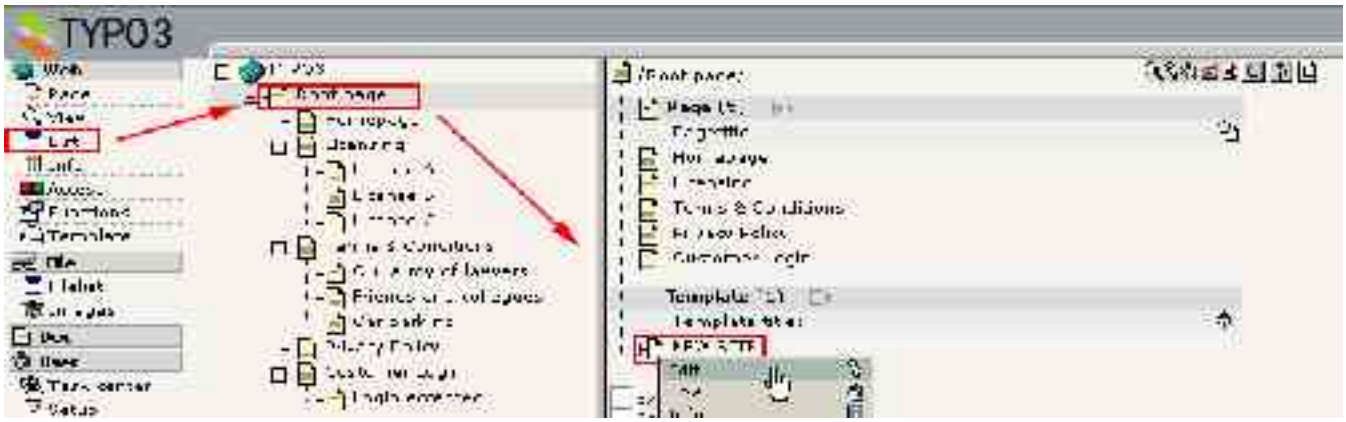
Nous allons maintenant créer un contenu de type Gabarit dans la page "Root page". Le plus simple est d'utiliser le module "Template".



L'écran suivant nous montre un vue du module "Template". Vous utiliserez cet écran pour modifier le gabarit par la suite :



Regardez maintenant la page "Root page" en utilisant le module "List" :

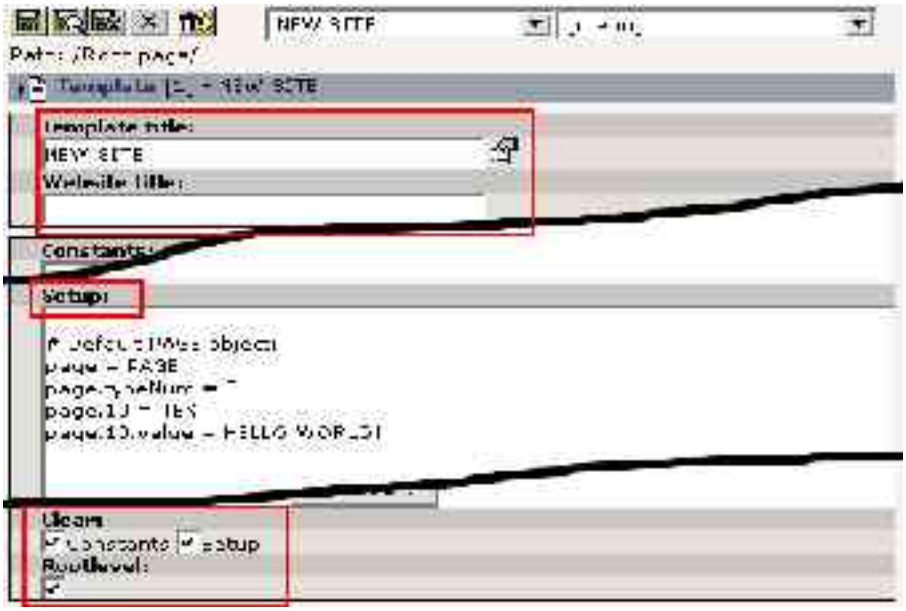


Vous voyez qu'un contenu de type "Template" ("Gabariat dans l'interface FR") a été créé dans la page "Root page".

Ce gabarit contient des informations de base pour le moteur de front end comme le rendu à effectuer pour le site web commençant sur cette page, quelles fonctionnalités sont activées, avec quels paramètres ... Un gabarit est**toujours**

nécessaire pour commencer un site Typo3. Vous pourrez par la suite décider ce que le gabarit contrôlera dans votre site. Vous pourriez parfaitement programmer l'ensemble de votre site depuis le gabarit, vous pourriez également demander à la maquette d'appeler immédiatement un script PHP personnel qui chercherait et afficherait l'ensemble du contenu de la page (en utilisant par exemple du XML /XSLT, d'autres bases de données ... à votre convenance). Vous pourriez également mélanger les 2 approches afin de tirer le meilleur des 2 solutions. C'est cette approche qui sera utilisée dans ce tutoriel.

En examinant le contenu de la maquette, on remarque plusieurs champs intéressants.



Le champs "Template title" est simplement le titre de la template. Il n'est affiché nulle part.

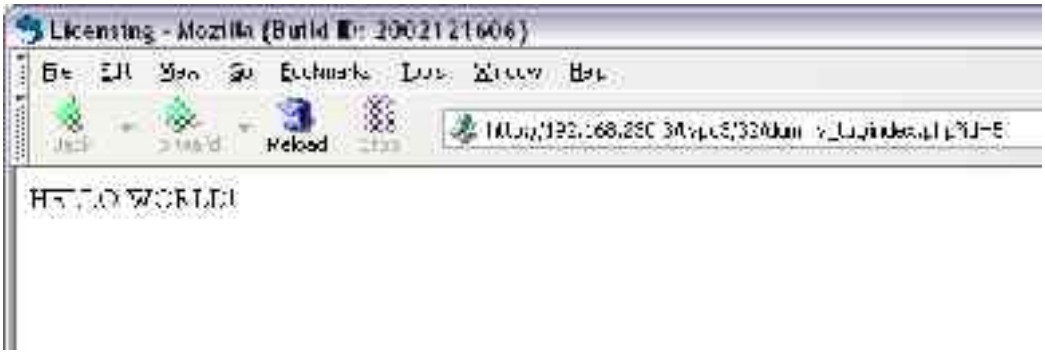
Le champ "Website title" sera le préfixe utilisé devant le titre de la page et affiché entre les tags <title>.Par exemple, si le titre du site est "Mon site", la page "Ma page" aura pour titre (affiché dans la barre de navigation et dans le tag HTML <title>: "Mon site :: Ma page") .

Dans le champ Setup, vous entrerez les instructions de configuration du moteur de front-end. Ces instructions sont décrites dans une structure hiérarchique définie en utilisant la syntaxe TypoScript. C'est pour cette raison que les gabarit sont souvent appelés "TypoScript templates".

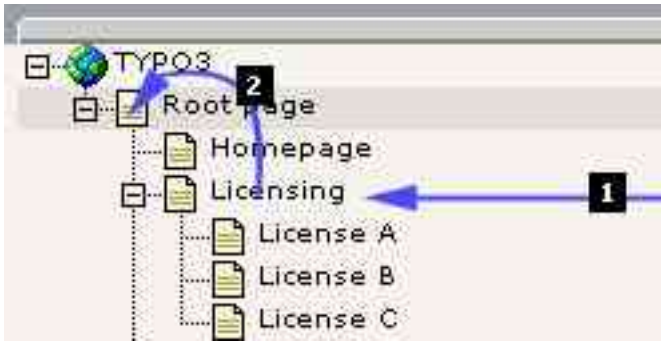
Les cases "Clear constants" et "Clear setup" signifient que le TypoScript ne sera pas hérité des pages parentes (situées au dessus, plus près de la racine). Ces cases devraient être systématiquement cochées pour les gabarit servant de maquette principales pour un nouveau site web (Rootlevel).

La case "Rootlevel" signifie simplement : "Débutes un nouveau site web à partir de cette page".

Revenons à la page "Licensing":



Différentes choses sont apparaues. Voici ce qu'il s'est passé :



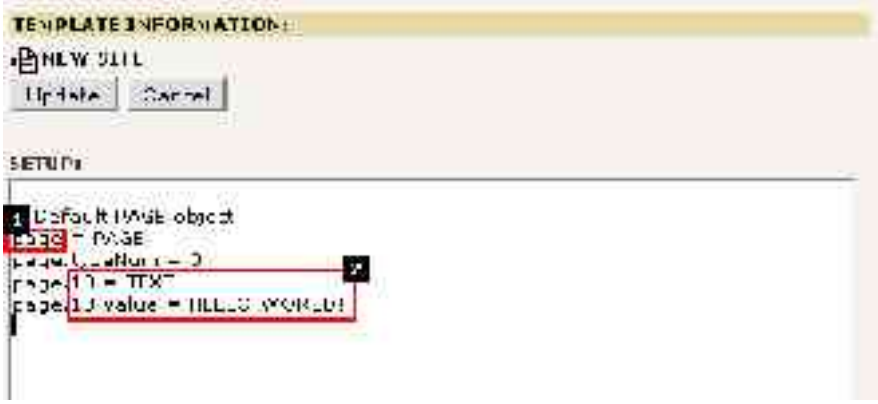
- Appelle la page avec l'uid "5" (étape #1)
- Sélectionne le premier gabarit contenu dans cette page (pid=5); Pas de gabarit dans cette page = pas de résultat !
- Comme il n'y a pas de gabarit dans la page d'uid "5", vérifie la page précédente dans la root line.... (#2)
- La page précédente (parente de uid=5) est la page "Root Page" (uid=1). Dans cette page trouve le premier gabarit. Un gabarit (au moins) est présent !
- Puisque nous avons un gabarit dans la page uid=1, vérifie que la case "Rootlevel" de ce gabarit est coché
- La case "Rootlevel" est effectivement cochée, le gabarit signale un début de site donc :
  - Analyse la configuration Typoscript décrite dans le champ Setup.
  - Commence l'exécution des instructions du champ Setup.

Les bases des instructions TypoScript (Champ Setup)

Au sein d'un gabarit, le principal point d'intérêt est le champ "Setup". Dans ce champ on trouvera un ensemble d'instructions qui détermineront ce qui se passera lorsque nous visualiserons cette page dans le front-end.

Le meilleur moyen d'éditer ce champ est par le module Template et utilisant la vue Info/Modifiy :

Cliquez sur l'icône "edit" :



Quelques explications :

L'objet PAGE (#1)

Nous commençons par définir un objet de premier niveau [appelé *Toplevel Object*" (TLO)]. Le type d'objet est "PAGE" (vous pouvez également définir un objet de type "FRAMESET" -cf le manuel de référence de TypoScript : TSRef-). A lieu de "page" pour nommer d'objet, il est possible d'utiliser un autre nom à l'exception des mots réservés aux TLO (voir TSRef). Comme propriété de notre objet page nous définissons la propriété obligatoire "typeNum" en lui affectant la valeur 0 (zéro).

Cela signifie que l'objet de premier niveau "page" sera maintenant l'objet par défaut acceptant les requête pour l'ensemble des pages et sous-pages de l'arborescence dont notre gabarit signale le début d'un nouveau site.

Nous allons maintenant donner à notre objet page des instructions sur ce qu'il doit faire. Nous allons pour cela consulter l'ensemble des propriétés possibles pour un objet de type "PAGE". Certaines de ces propriétés sont des méta-propriétés comme "config" ou "includeLibs", d'autres sont des propriétés indiquant quel code HTML l'objet page doit produire.

L'ensemble des propriétés les plus significatives des objets PAGE est représenté par l'ensemble des content objects (cObjects). Chacun de ces cObjects peut produire une chaîne HTML dont l'assemblage produira le corps de la page web (entre <body> et </body>). Ces cObjects sont regroupés dans un tableau numérique hiérarchisé.

L'objet Content (#2)

Dans l'exemple ci-dessus, un objet très simple est défini -de type TEXT-. Il porte l'index "10" dans le tableau hiérarchique des objets de contenu (ie. dans l'arborescence des cOjects de page). Ce cObject a sa propriété "value" positionnée à "HELLO WORLD !". Cela a pour conséquence que le contenu généré par cet objet sera "HELLO WORLD !" et le code HTML de la page générée sera :

Vous trouverez l'ensemble des propriétés du cObject TEXT ici.

Autres exemples concernant les objets PAGE et cObjects

Cette section présentera quelques exemples supplémentaires sur les PAGE et cObjects de manière à vous aider à comprendre le concept pleinement. La compréhension de ce point vous permettra de comprendre les "TypoScript templates" aux structures plus complexes.

Deux cObjects?

Que se passe-t-il si nous ajoutons un autre cObjetc ?

```
# Default PAGE object:
page = PAGE
page.typeNum = 0

# Content object one:
page.10 = TEXT
page.10.value = HELLO WORLD!

# Content object two:
page.20 = TEXT
page.20.value = HELLO UNIVERSE!
```

(Notez que les lignes débutant par "#" ou "/" sont des commentaires.)

Sortie :

```
<body bgcolor="#FFFFFF">
HELLO WORLD!HELLO UNIVERSE!
</body>
```

Donc logiquement, les objets peuvent être concaténés, additionnés les uns aux autres.

Changer l'ordre ?

Que se passe-t-il si l'ordre des définitions est modifié ?

```
# Default PAGE object:
page = PAGE
page.typeNum = 0

# Content object two:
```

```
page.20 = TEXT
page.20.value = HELLO UNIVERSE!

# Content object one:
page.10 = TEXT
page.10.value = HELLO WORLD!
```

Sortie :

```
<body bgcolor="#FFFFFF">
HELLO WORLD!HELLO UNIVERSE!
</body>
```

Le résultat est le même. L'ordre de génération dépend en fait de l'index dans le tableau des cObjects (10,20,30 ...).

Contenu dynamique ?

Comment insérer un cObject TEXT renvoyant le titre de la page ?

Il faut consulter le "Typoscript Reference" (TSRef) afin de savoir si le cObject TEXT peut accepter une propriété permettant d'afficher cette information. Outre la propriété "value", il existe au même niveau les propriétés "stdWrap" qui peuvent être utilisées pour récupérer des données et les traiter automatiquement.

Puisque le rendu du cObjet TEXT est effectué au sein de l'objet PAGE, le *current record* (l'enregistrement courant, celui dont les données seront analysées) sera l'enregistrement de la page courante (*table pages de la base de données*). En spécifiant un champ au moyen de la propriété "field" de stdWrap, on récupérera son contenu pour la page courante. Par exemple, le champ "title" contient le nom de la page. Bien évidemment ceci suppose de savoir quels sont les champs contenus dans la table page !

Exemple :

```
# Default PAGE object:
page = PAGE
page.typeNum = 0

# Content object outputting current page title:
page.10 = TEXT
page.10.field = title
```

Sortie HTML :

```
<body bgcolor="#FFFFFF">
Licensing
</body>
```

Comme nous sommes sur la page "Licensing", c'est ce titre qui sera affiché (objet page.10)

Propriétés stdWrap :

Maintenant que nous savons utiliser la propriété "stdWrap", nous allons allons voir comment l'encapsuler , lui appliquer divers traitements ...

```
# Default PAGE object:
page = PAGE
page.typeNum = 0

# Content object outputting current page title:
page.10 = TEXT
page.10 {
  field = title
  crop = 8 | ...
  case = upper
  wrap = This is the truncated page title: <b> | </b>
}
```

(Notez le changement de formatage du TypoScript afin que les propriétés soient incluses dans des accolades. Cette syntaxe permet d'affecter plus facilement des plusieurs propriétés au même objet ou niveau. Le résultat final est le même qu'en préfixant toutes les propriétés par le chemin complet de l'objet : "page.10.").

Résultat :

```
<body bgcolor="#FFFFFF">
This is the truncated page title: <b>LICENSIN...</b>
</body>
```

Notez que la chaîne d'encapsulation (*wrap = This is the truncated page title: | *) a été ajoutée de chaque coté du titre et que celui-ci est passé en majuscules (uppercase) et à été tronqué à 8 lettres en ajoutant "..." à la fin !

Nous aurions pu obtenir le même résultat en utilisant un autre cObjetc : "HTML". L'unique différence est que toutes les propriétés stdWrap de ce type d'objet sont des sous propriétés de "value" et non des propriétés de l'objet lui même :

```
# Default PAGE object:
page = PAGE
```

```
page.typeNum = 0

# Content object outputting current page title:
page.10 = HTML
page.10.value {
  field = title
  crop = 8 | ...
  case = upper
  wrap = This is the truncated page title: <b> | </b>
}
```

Le choix entre l'utilisation de cObjects HTML ou TEXT est une question de goûts personnels.

Contenus depuis des scripts PHP

Vous pouvez aisément insérer des scripts PHP si vous le souhaitez. En fait c'est même recommandé si vous souhaitez obtenir des résultats conditionnels dont les clauses dépassent les limites du TypoScript. Les cObjects et leurs propriétés ne disposent pas de "programmation procédurale", ce sont justes des objets préprogrammés réagissant à des conditions spécifiques.

Dans les cas complexe, vous aurez besoin du cObject USER.

Tout d'abord, il faut créer le fichier PHP (dans **fileadmin/userfunctions.php** pour cet exemple) :

```
<?php

class user_functions {

	/**
	 * Multiplies the current page ID with $conf["factor"]
	 */
	function multiplyTest($content,$conf)		{
		$currentPageUid = $GLOBALS['TSFE']->id;
		$factor = intval($conf['factor']);

		return $currentPageUid * $factor;
	}
}
?>
```

Ensuite, il faut configurer un cObjet de type USER afin d'appeler la fonction avec son unique paramètre : "factor" :

```
# Default PAGE object:
page = PAGE
page.typeNum = 0
page.includeLibs.some_random_id_string = fileadmin/userfunctions.php
page.config.admPanel = 1

# Content object outputting current page title:
page.10 = HTML
page.10.value = The page ID, {field:uid}, multiplied with 15 is:
page.10.value.insertData = 1
page.10.value.wrap = <b> |</b> <br />

page.20 = USER
page.20.userFunc = user_functions->multiplyTest
page.20.factor = 15
```

Le résultat obtenu sera :

Comme vous le constatez, nous avons défini quelques "méta-propriétés" qui ne sont pas directement liées au contenu obtenu.

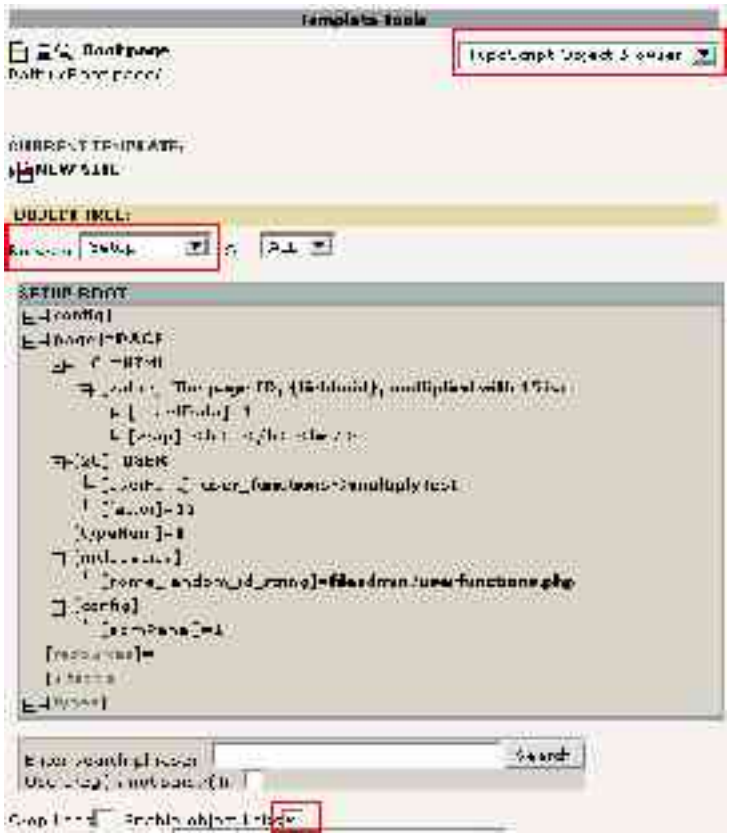
```
page.includeLibs.some_random_id_string = fileadmin/userfunctions.php
page.config.admPanel = 1
```

"includeLibs" est une propriété vous permettant de spécifier une liste de fichiers PHP (des bibliothèques de classes et de fonctions) à inclure AVANT la génération de la page. "config" correspond à de nombreuses options de configuration du comportement général de l'objet PAGE. Dans le cas présent, il ajoute un panneau d'administration en bas de page.

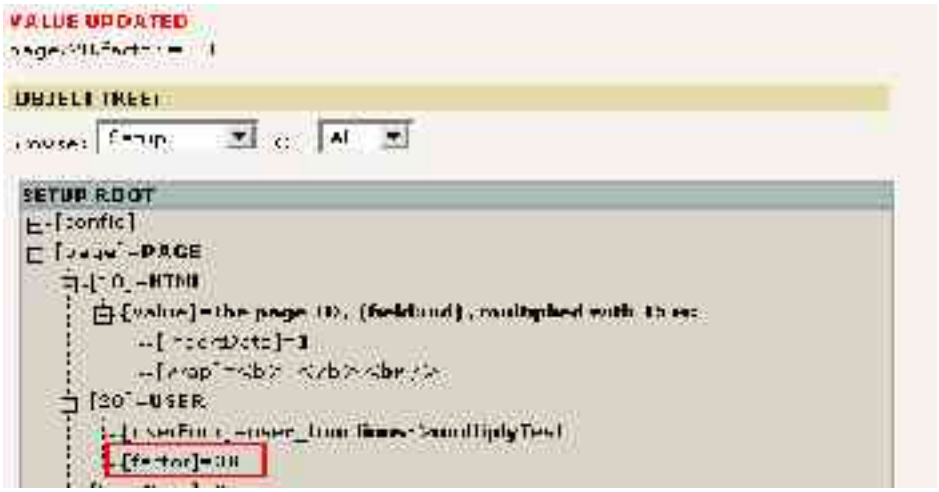
Prenez quelques minutes pour analyser attentivement l'exemple. Notez comment la propriété "factor" de l'objet USER est disponible dans la fonction PHP. Notez également comment la fonction PHP récupère l'ID de la page et comment l'"admin panel" est inclus. Cet exemple est assez instructif et vous devriez l'étudier jusqu'à en posséder une bonne maîtrise.

Le navigateur d'objets (Object Browser)

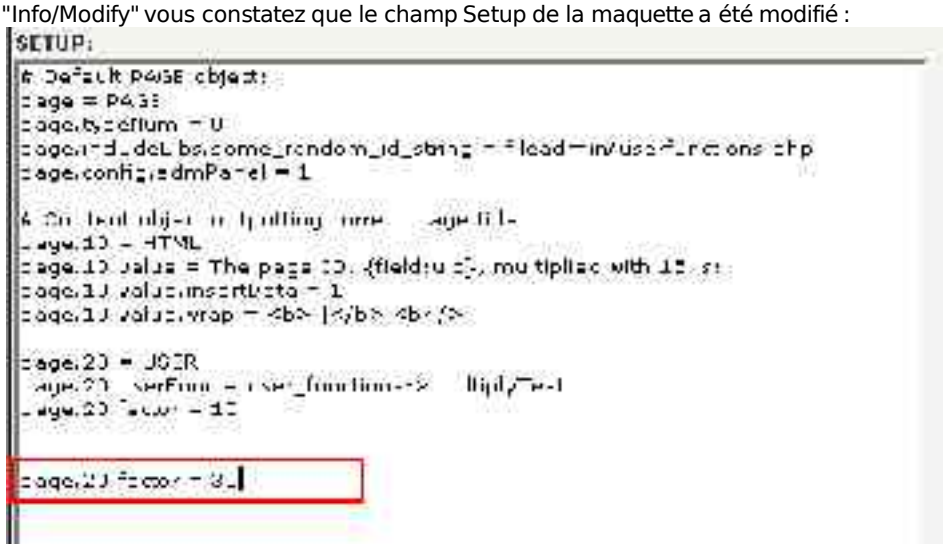
Le navigateur d'objets dans le module Template (gabarit en français) est un outil indispensable pour analyser la structure de vos maquettes :



Comme vous le voyez, le TypoScript est analysé en une arborescence d'objets très structuré. Cet outil vous permet de vérifiez que chaque propriété que vous avez saisie est correctement placée, voire d'éditer ces propriétés si la case "Enable object links" est cochée :



En revenant dans "Info/Modify" vous constatez que le champ Setup de la maquette a été modifié :



Le navigateur d'objets n'est cependant pas suffisamment intelligent pour retrouver et changer la ligne "page.20.factor = 15". Il insère simplement une nouvelle ligne à la fin du Setup qui écrasera les valeurs spécifiées auparavant. Vous pourrez supprimer la valeur inutile manuellement. Le navigateur d'objet fournit une manière sûre de spécifier les valeurs de propriétés puisque la totalité l'objet est reprise.

Poser ses conditions ?

Nous pourrions utiliser ce cas de figure pour illustrer les "conditions". Une condition très simple vérifiant le type du navigateur utilisé par le visiteur :

```
# Default PAGE object:
page = PAGE
page.typeNum = 0
page.includeLibs.some_random_id_string = fileadmin/userfunctions.php
page.config.admPanel = 1

# Content object outputting current page title:
page.10 = HTML
page.10.value = The page ID, {field:uid}, multiplied with 15 is:
page.10.value.insertData = 1
page.10.value.wrap = <b> |</b> <br />

page.20 = USER
page.20.userFunc = user_functions->multiplyTest
page.20.factor = 15

[browser = msie]

page.20.factor = 30
page.10.value = The page ID, {field:uid}, multiplied with 30 is:

[global]
```

Dans Microsoft Internet Explorer la page ressemblera à ceci :

Dans les autres navigateurs (ici Mozilla) on obtiendra :

L'objet PAGE revisité

Le but n'est pas de vous donner un cours complet sur la manière de réaliser des maquettes en se fondant uniquement sur les cObjects. Ce manuel ne fait qu'utiliser les cObjects TEMPLATE, USER et HMENU cObjects en les combinant avec du TypoScript et des maquettes statiques afin d'arriver à nos fins. L'ouvrage de référence pour le TypoScript, les objets et leurs propriétés est TSref et si vous souhaitez plus d'exemples pratiques sur l'utilisation des différents types d'objets veuillez vous reporter au manuel TypoScript by Example.

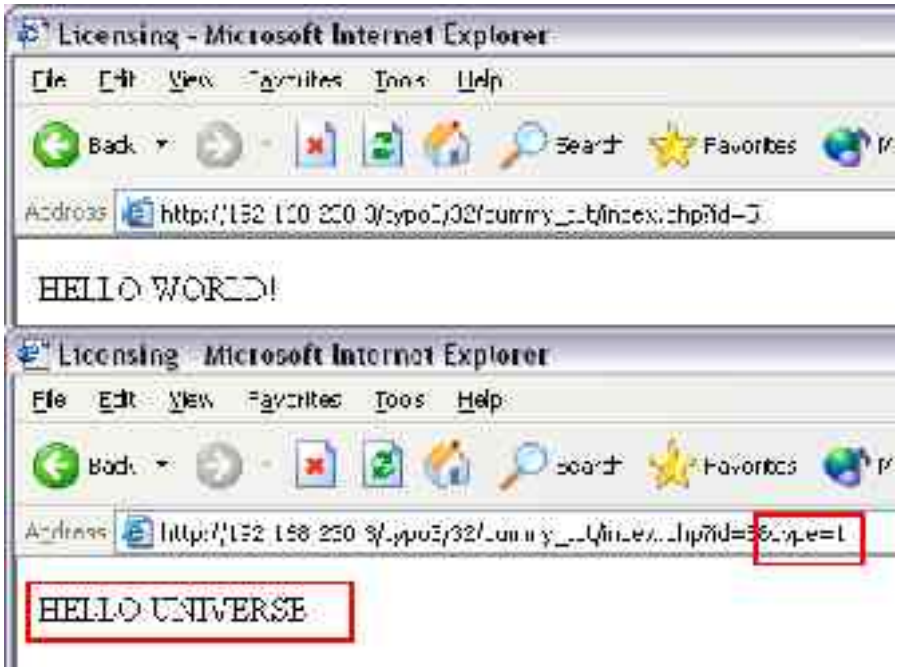
Cependant je souhaiterais conclure sur deux points :

La signification de "&type="

Dans cet exemple, il n'y a pas qu'une seule page de définie. L'objet de premier niveau (TLO) "another_page" a également été définie comme un objet PAGE mais avec un "typeNum" à "1". Ceci signifie que l'objet "page" de type PAGE représente la page par défaut de tandis que l'objet PAGE "another_page" recevra les requêtes où le paramètre "&type=1" sera présent en plus du paramètre id.

```
page = PAGE
page.typeNum = 0
page.10 = TEXT
page.10.value = HELLO WORLD!

another_page = PAGE
another_page.typeNum = 1
another_page.10 = TEXT
another_page.10.value = HELLO UNIVERSE
```

Le résultat sera :



et l'arborescence d'objets :

Copier des objets

Il devient rapidement pratique de diviser son code TypoScript en blocs puis de les assembler à la fin de la maquette en les copiant aux positions adéquates. Cela peut même aller jusqu'à la séparation des blocs dans une hiérarchie de sous-maquettes (par inclusion) ou de maquettes statiques permettant la réutilisation du TypoScript (nous en reparlerons plus loin).

Cette possibilité repose sur une fonctionnalité de Typoscript permettant de copier une branche de l'arborescence d'objets dans une autre branche :

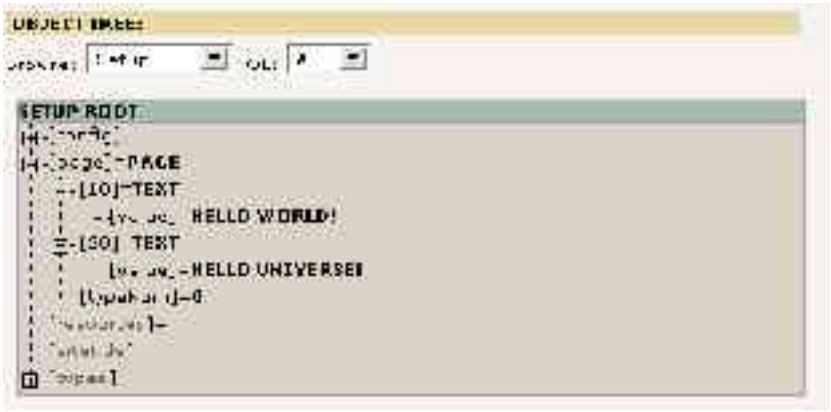
```
# Make temporary version of the first cObject:
temp.world = TEXT
temp.world.value = HELLO WORLD!


# Make temporary version of the second cObject:
temp.universe = TEXT
temp.universe.value = HELLO UNIVERSE!


# Default PAGE object:
page = PAGE
page.typeNum = 0

page.10 < temp.world
page.20 < temp.universe
```

Le résultat sera bien évidemment le suivant :

```
<body bgcolor="#FFFFFF">
HELLO WORLD!HELLO UNIVERSE!
</body>
```

Si vous regardez dans le navigateur d'objets, vous verrez que les objets "temp..." ont été copiés dans les branches "page.10" et "page.20". Cependant, nous aurions pu nous attendre à voir les TLO "temp..." dans l'arborescence. Ce n'est pas le cas. En

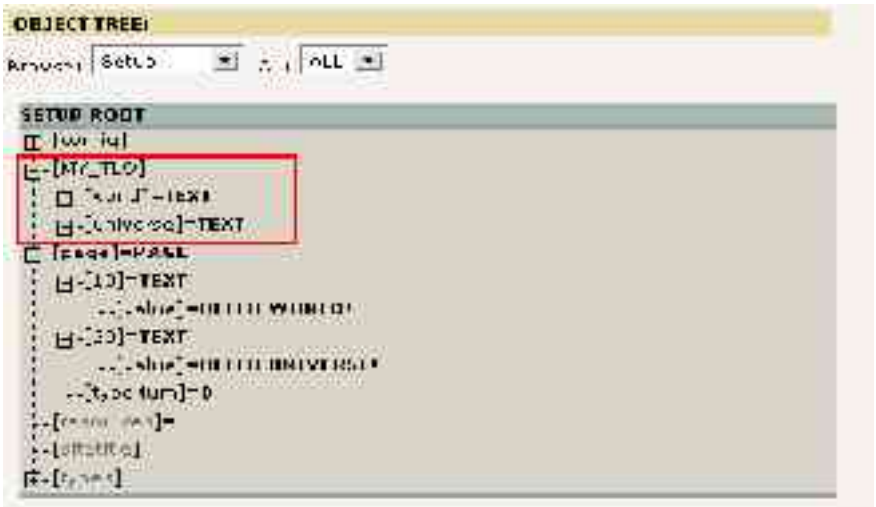
effet, les TLO "temp." et "styles." sont retirés après l'analyse du TypoScript. Ils sont réservés à des objets temporaires disponibles pendant l'analyse du TypoScript mais pas après.



Si nous souhaitons conserver ces objets dans l'arborescence, il nous faut utiliser d'autres TLO :

```
# Make temporary version of the first cObject:
MY_TLO.world = TEXT
MY_TLO.world.value = HELLO WORLD!


# Make temporary version of the second cObject:
MY_TLO.universe = TEXT
MY_TLO.universe.value = HELLO UNIVERSE!


# Default PAGE object:
page = PAGE
page.typeNum = 0

page.10 < MY_TLO.world
page.20 < MY_TLO.universe
```

... et le TLO MY_TLO est correctement conservé :



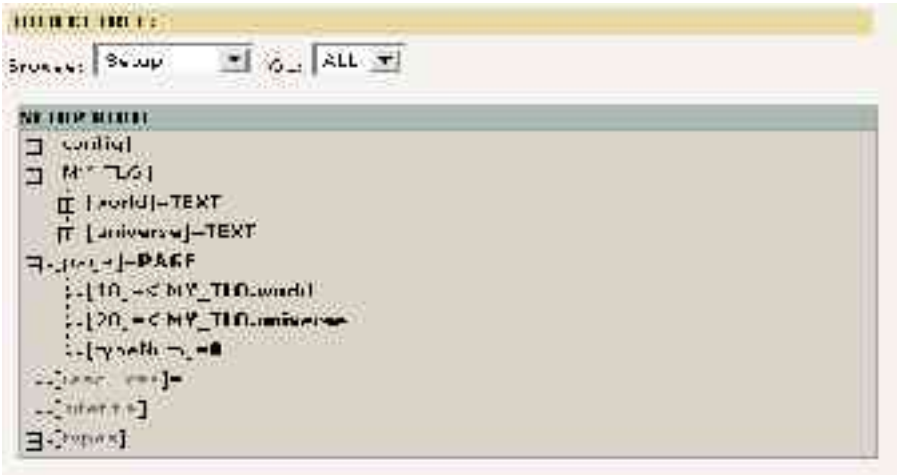
Des Références et non plus des copies

Cela signifie également que nous pouvons créer une référence aux objets de MY_TLO. Établir des références à des cObjets suppose que ceux-ci existent dans la structure de la page après que l'analyse du TypoScript ai été effectuée (contrairement aux TLO "temp." et "style."). Nous pouvons donc réécrire le script comme suit :

```
# Make temporary version of the first cObject:
MY_TLO.world = TEXT
MY_TLO.world.value = HELLO WORLD!


# Make temporary version of the second cObject:
MY_TLO.universe = TEXT
MY_TLO.universe.value = HELLO UNIVERSE!
```

```
# Default PAGE object:
page = PAGE
page.typeNum = 0

page.10 =< MY_TLO.world
page.20 =< MY_TLO.universe
```

Le changement a l'air minime mais l'impact est très important puisque les objets ne sont plus copiés mais référencés :



Il y a plusieurs conséquences pratiques et des inconvénients à cette méthode. Cependant, l'avantage des référence est que le cObject défini à un point de l'arborescence peut être utilisé plusieurs fois par référence au sein de la structure de page, sans être dupliqué en mémoire. Il devient aisé également de modifier les propriétés de cet objet puisqu'il n'est pas nécessaire de retrouver tous les duplicata. Un exemple de ce problème est disponible à la fin de la partie 2 de ce tutoriel.

Note: *Ce type de références n'est pas en soi une syntaxe TypoScript. C'est une fonctionnalité interne des cObjets. Par voie de conséquence, ces références ne peuvent être utilisées qu'en faisant référence à des cObjets à l'exclusion de tout type d'objets ou propriétés (sauf précisions).*

Vider le Cache

Enfin vous devez vous rappeler que lorsque vous effectuez une modification via le module Template, la totalité du cache est vidé. Ceci est nécessaire puisque lorsque Typo3 analyse une maquette, la structure obtenue est stockée dans le cache afin de la rappeler plus rapidement lorsque la page sera rappelée. Ceci signifie également que les modifications effectuées directement dans l'enregistrement de la maquette en utilisant le module "List" ne vident pas le cache. Il est alors nécessaire de le faire en utilisant le lien en bas du menu gauche ou le menu déroulant en haut du formulaire de la maquette.

D'une manière générale, avant d'annoncer un bug face à un comportement inattendu, veuillez vider les caches !!

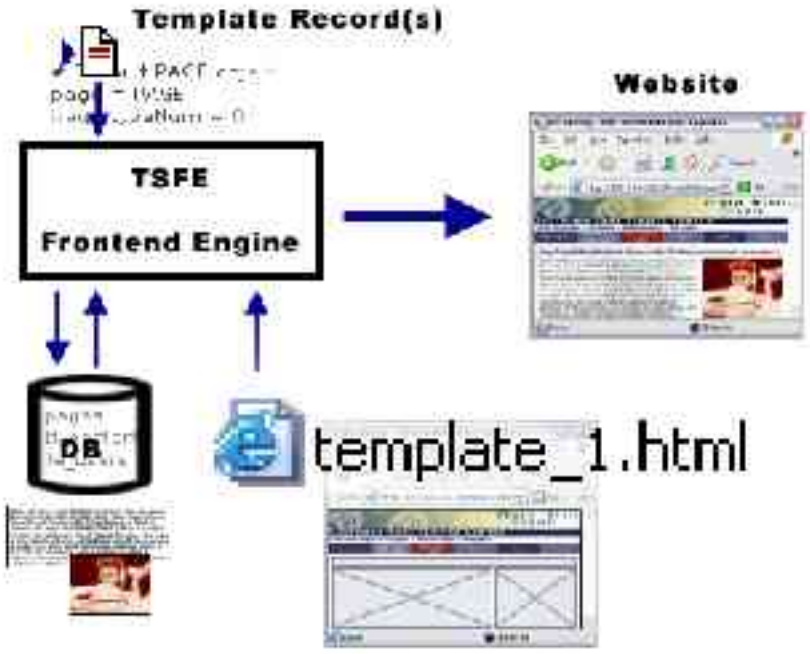
Partie 1 : Intégration d'une maquette HTML

Implémentation d'un CMS

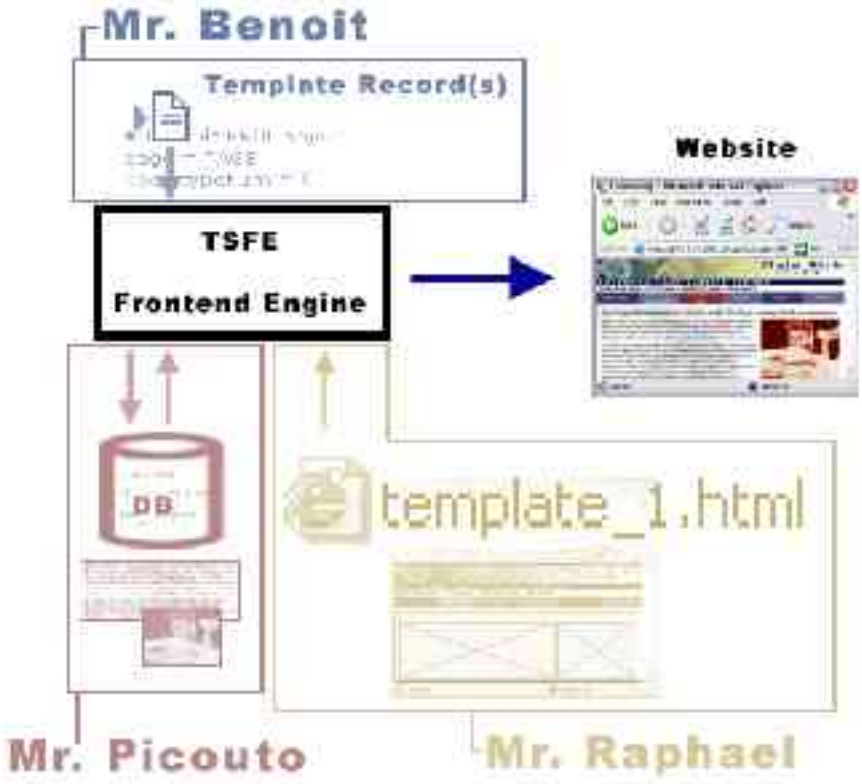
La création d'un site web avec un CMS implique plus de composants qu'un site composé de pages statiques en pur HTML. L'idée sous-jascente d'un CMS est que le contenu est stocké séparément de la présentation. Certains CMS stockent ces contenus dans des fichiers XML, d'autres utilisent une base de données. Chaque technique a ses avantages et ses inconvénients, mais l'idée maîtresse reste la même : "Séparer les contenus des couleurs".

Quand Typo3 génère une page, le moteur de frontend combine du contenu non formaté provenant de la source de données (la base de données) avec une maquette HTML qui défini le formatage. Dans ce processus, c'est le gabarit (l'enregistrement correspondant à la maquette) qui représente le chef d'orchestre et qui indique au moteur comment comment combiner les éléments.

Sur le schéma suivant, vous pouvez voir que le gabarit est l'élément contrôlant -le "programme"- le moteur de frontend qui va lui rechercher le contenu dans la base de données, lire la maquette statique, insérer le contenu aux emplacements réservés dans la maquette statique et, finalement, produire une jolie page web !



Dans la plupart des agences web, plusieurs personnes travaillent de concert pour produire un site web. Dans ce groupe, nous trouverons un graphiste, un développeur et un contributeur. Chacune de ces personnes a des compétences spécifiques et de ce fait interviendra à sur des parties différentes du cycle de production du site :



Dans l'illustration ci-dessus, les différentes composants d'un site web gérés par un CMS sont assignées à une personne différente de l'équipe web :

- **Mr. Raphaël** est l'artiste du groupe. Raphaël a de fortes compétences en design et en arts graphiques. Il connaît GIMP (ou Photoshop), Quanta (ou Dreamweaver), les CSS, le HTML ... Raphaël jongle avec les films en Flash mais pas avec le PHP, le Typoscript, le SQL ... Raphaël se chargera donc de créer les maquettes statiques HTML !
- **Mr. Benoît** est un développeur, il aime le code, les instructions de compilation, les expressions régulières, la logique, le PHP, le SQL – bientôt il adorera le Typoscript. Cependant, s'il adore son bon vieux jean comme tous les programmeurs et designers, les couleurs, la typographie, l'ergonomie et les designs "hype" sont absents de son horoscope. Par conséquent, Benoît sera en charge du Typoscript dans les gabarit.
- **Mr. Picouto** est lui préoccupé par le contenu. C'est le gars du marketing et il est stupéfait par les merveilles que font Raphaël et Benoît pour transporter les idées du papier sur l'écran. Picouto n'est ni un programmeur, ni un designer, par contre, il a un message à faire passer. Il créera le contenu qu'il souhaite en utilisant l'interface de contribution (backend) de Typo3 sans avoir besoin d'autre compétence technique que de savoir utiliser un traitement de texte.

Nous avons donc 3 personnes possédant chacune des compétences spécifiques. Ceci nous conduit à la conclusion que la création de sites web n'est pas une promenade de santé si vous ne possédez pas les 3 compétences citées ci-dessus : Visuelle, Technique, Marketing. C'est généralement vrai dans les équipes web, plus rarement pour les individus, soyez en conscients. (Si vous n'avez pas toutes ces compétences, vous aurez vraisemblablement soit à apprendre beaucoup, soit à utiliser les maquettes standard qui sont des mises en pages fixes que vous pourrez mettre en place avec un minimum de configuration (ce qui n'est pas expliqué dans ce manuel)).

Dans ce tutoriel, je vous montrerais comment Raphaël, Benoît et Picouto devront coopérer afin de créer un site web moderne, correspondant à tous les designs possibles et facilement maintenable, où chaque participant pourra travailler librement sur son domaine réservé, en utilisant ses outils préférés. Et ce, grâce à Typo3.

Niveau, compétences et prérequis.

Webdesigner intermédiaire connaissant HTML/CSS sans être nécessairement une programmeur web.

Pour suivre cette partie du tutoriel, vous aurez besoin d'une bonne connaissance de HTML et CSS. Vous aurez également besoin d'une base de données TYPO3 fonctionnelle contenant l'arborescence que nous avons créé dans les chapitres précédents. Enfin, vous pourrez avoir besoin de connaître les concepts de programmation afin de comprendre en profondeur cette partie. Ne vous inquiétez pas, je vous indiquerais précisément quoi faire et comment, prenez le temps de comprendre les exemples et suivez précisément les instructions.

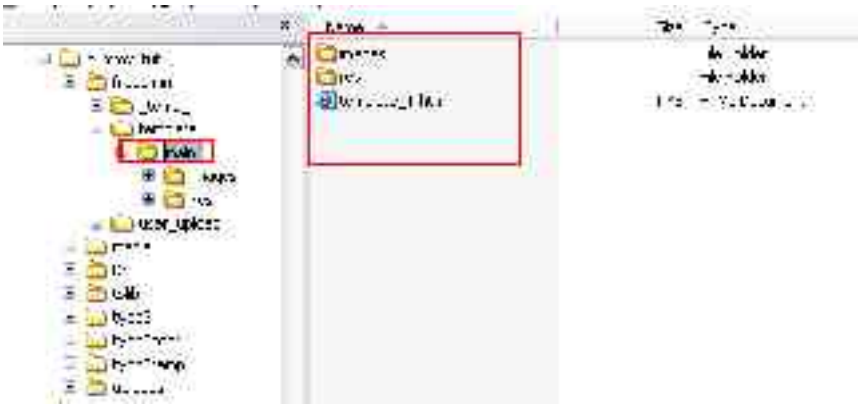
La maquette HTML statique

L'équipe Web a un nouveau client - Main Dish & Son – et Raphaël, l'artiste du groupe a créé une maquette du site web sous la forme d'une page HTML classique :


Main Dish
& son
Buy PaperShredder(tm) Gizmo with 30-days money back guarantee!

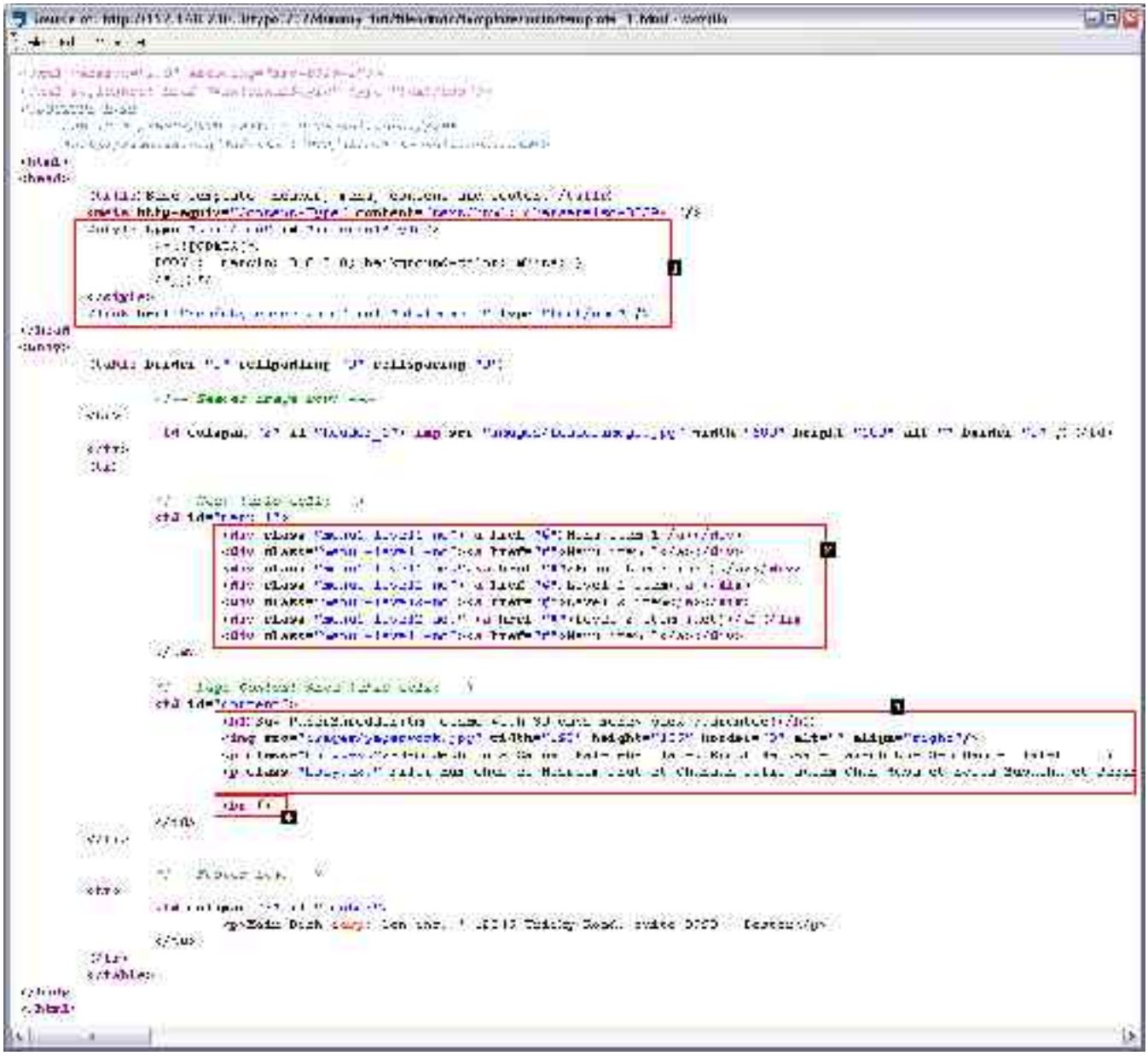
Cette maquette est placée dans le répertoire "fileadmin/template/main/" de l'installation Typo3 (le dummy-package).

Afin de suivre ce guide, vous devriez copier le contenu du dossier "part1/" de l'extension correspondant à ce guide dans le dossier "fileadmin/template/". Si vous n'avez pas encore importé l'extension "doc_tut_templselect" depuis le TER (Typo Extension Repository) vous devriez le faire dès maintenant !



Revenons a travail de Raphaël. La maquette statique HTML est un page HTML normale. Mais lorsque Typo3 utilisera cette maquette, l'objectif est de rendre certaines parties dynamiques, notamment le menu à gauche ainsi que la partie avec le contenu d'exemple au centre droit.

En examinant le code source de la maquette, on trouve un simple document XHTML se référant à une feuille de style et utilisant une simple table pour positionner les différents éléments dans la page :



Quelques commentaires sur cette maquette et sur les problèmes qui nous attendent :

1. Cette section du "header" doit être inclus dans notre page web puisqu'il fait référence à la feuille de style utilisée. Difficulté : nous devons nous assurer d'extraire cette partie et l'inclure dans la partie "header" qui est générée par le moteur de frontend !
2. Le menu à gauche est constitué d'un par objet du menu. Chacune de ces a une classe (au sens CSS) qui lui est assignée. Grâce à ce *class* la présentation de l'élément est contrôlé par la feuille de style.
   C'est une façon astucieuse de créer un menu puisque chaque élément représente un minimum de code HTML (ce qui est bon pour l'implémentation Typoscript), élément qui peut facilement être répété (cela sera nécessaire lorsque le menu sera dynamique.
   Difficulté : nous devons remplacer le menu d'exemple qui est ici statique par un menu dynamique, généré par Typo3.
3. Le content d'exemple que Raphaël a inséré dans la maquette sert juste à donner une visualisation réaliste de la maquette. Notez comment ce contenu et formaté en utilisant les balises <h1> et <p> (utilisant la classe "bodytext"). C'est une bonne approche puisque le contenu dynamique généré par Typo3 utilisera plus tard les mêmes balises et classes pour sa présentation ! (J'ai l'impression que Raphaël a quelque peu triché en se documentant sur Typo3, non ?)
   Difficulté : Nous devons remplacer le contenu d'exemple par une contenu généré dynamiquement.
4. Ce tag *
* crée un espace sous le contenu afin que le pied de page ne soit pas collé au corps de la page.

Enfin, vous noterez que les cellules du tableau qui contiennent le menu et le contenu ont été marquées avec un identifiant (*id-attribute*). Ceci n'est pas seulement utilisé avec les feuilles de style ! il y a une très bonne raison à l'utilisation de ces *id-attribute* ! Mais avant cela un peu de théorie sur les maquettes HTML :

Les bases du cObject TEMPLATE

Commençons par créer un nouveau fichier, fileadmin/template/test.html, avec ce contenu :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
      <title>Untitled</title>
</head>

<body>

<!-- ###DOCUMENT_BODY### -->
      <h1>
            <!-- ###INSIDE_HEADER### -->
                  Header of the page
            <!-- ###INSIDE_HEADER### -->

      </h1>
<!-- ###DOCUMENT_BODY### -->

</body>
</html>
```

(Ce fichier peut être trouvé dans le répertoire de l'extension : "misc/test.html")

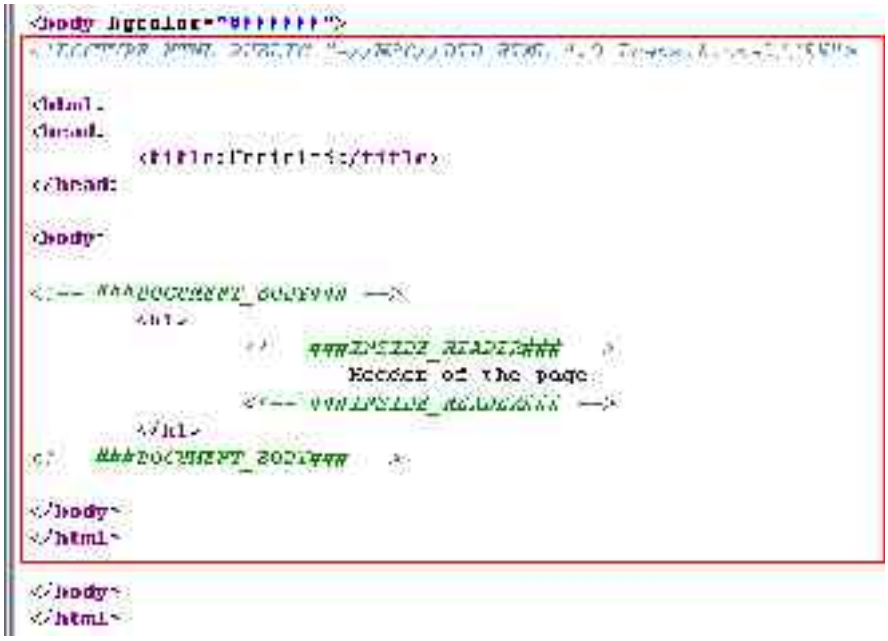
Ensuite, mettez ceci dans le champ Setup de votre gabarit :

```
# Template content object:
temp.mainTemplate = TEMPLATE
temp.mainTemplate {
  template = FILE
  template.file = fileadmin/template/test.html
}


# Default PAGE object:
page = PAGE
page.typeNum = 0

page.10 < temp.mainTemplate
```

Ceci créera un cObject de type "TEMPLATE" à la position "page.10". La propriété "template" de ce cObject TEMPLATE est défini comme étant un autre cObject de type "FILE". Ce cObject lit le fichier créé précédemment, "fileadmin/template/test.html". Les propriétés des cObject TEMPLATE peuvent être trouvées ici.

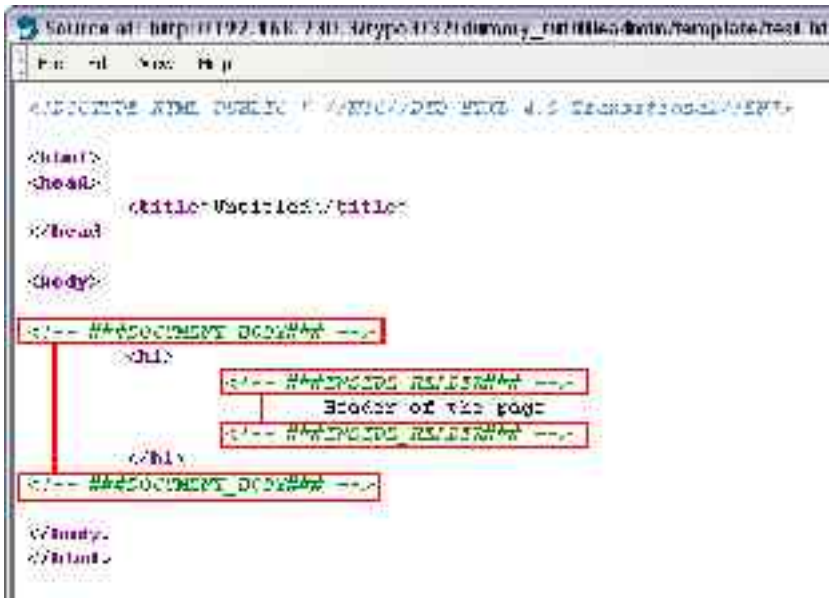
Si vous enregistrer les changements dans le champ Setup et visualiser la page en frontend vous verrez ceci :

En examinant les sources de la page, on constate que le cObject TEMPLATE se content de lire le fichier et le renvoie directement :



Maintenant examinons la principale fonctionnalité du cObject TEMPLATE. Il ne se contente pas de lire le fichier HTML mais nous permet d'en extraire des parties (*subparts*) et de les remplacer par un contenu dynamique !

Une *subpart* est définie comme le contenu compris entre deux marqueurs similaires entourés de ###et contenus dans un commentaire HTML (bien que cette dernière condition soit optionnelle). Le fichier "test.html" a donc deux*subpart*, la sous-partie "DOCUMENT_BODY" et la sous-partie "INSIDE_HEADER". Comme vous le constatez, les marqueurs de sous-parties sont encapsulées dans des commentaires HTML afin qu'il ne soient pas visibles.



Essayons maintenant de changer le champ Setup du gabarit en ceci :

```
# Template content object:
temp.mainTemplate = TEMPLATE
temp.mainTemplate {
  template = FILE
  template.file = fileadmin/template/test.html
  workOnSubpart = DOCUMENT_BODY
  subparts.INSIDE_HEADER = TEXT
  subparts.INSIDE_HEADER.value = HELLO WORLD!
}


# Default PAGE object:
page = PAGE
page.typeNum = 0

page.10 < temp.mainTemplate
```

Rechargez la page et le code source devrait ressembler à ceci :

Les changement appliqués au cObjet TEMPLATE ont produit les choses suivantes :

- Tout d'abord, le cObject TEMPLATE a reçu l'instruction de ne travailler QUE sur la sous-partie "###DOCUMENT_BODY###" – ainsi, le header et le tag body sont retirés (ils seraient redondants dans la page).
- Dans un deuxième temps, la sous-partie délimitée par "###INSIDE_HEADER###"a été remplacée avec le contenu généré par le cObject TEXT défini dans la propriété "subparts.INSIDE_HEADER".

Suite à cet exemple simpliste, nous devons maintenant simplement pointer vers la maquette de Raphaël, fileadmin/template/main/template_1.html, insérer des marqueurs de sous-partie similaires et enfin remplacer le menu et le contenu d'exemple par un menu dynamique et le vrai contenu de la page.

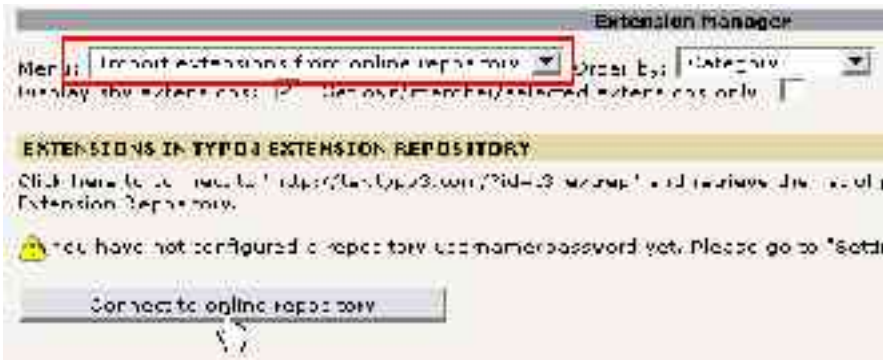
L'extention Template Auto-parser (Analyseur de maquettes)

Le plus simple serait d'éditer la maquette de Raphaël. Cependant, d'après mon expérience, vous ne pouvez faire confiance aux éditeurs HTML pour qu'il n'enlèvent pas ou ne réarrangent pas les commentaires HTML lorsque Raphaël retouchera la maquette par la suite. Il est parfaitement possible que Raphaël effectue quelques changements avec Dreamweaver et que les marqueurs de sous-partie soient retirés de manière silencieuse par le logiciel, la maquette cesserait immédiatement de fonctionner ! De plus, les chemins pointant vers la feuille de style et les images à l'intérieur de la maquette utilisent des chemins relatifs à l'emplacement de cette maquette ([domain]/fileadmin/template/main/), or le moteur de frontend affiche la page en utilisant le chemin [domain]/. Tous les chemins utilisés devront donc être préfixés !

L'ensemble de ces problèmes impose une meilleure solution que la simple édition de la maquette. Importez et installez maintenant l'extension "Template Auto-parser".

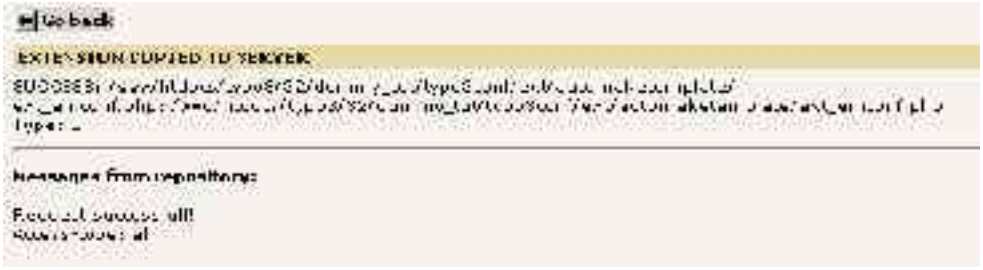
Installer l'extension

Allez dans le module "Extension Manager", et importez l'extension depuis le dépôt en ligne (online repository) :



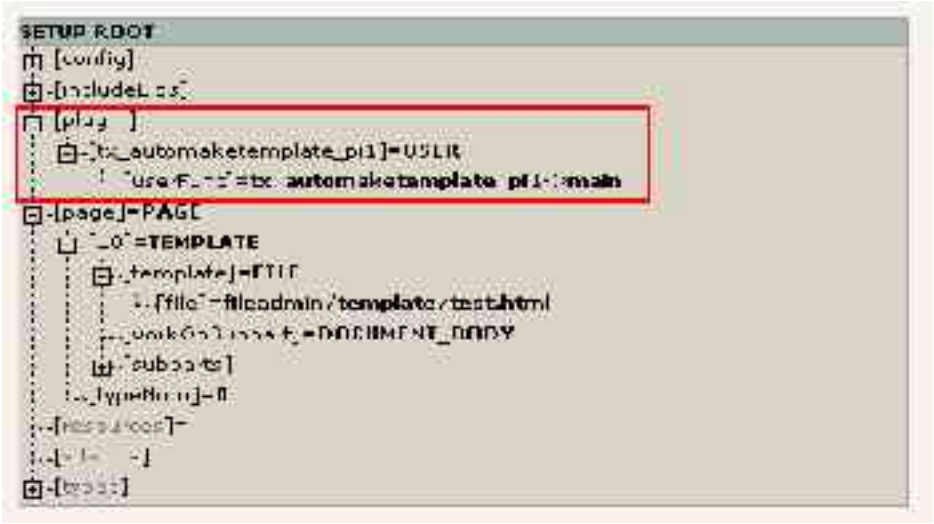
Trouvez l'extension "automaketemplate", cliquez sur l'icône d'import :

Un import réussi devrait vous donner ce résultat :



Utilisez "Go back", sélectionnez "Available Extensions to Install" et installez l'extension:

Après avoir pressé le bouton "Make updates" button, retournez dans le module "Template" et regardez "l'Object Browser" :



L'extension a rajouté un cObjet USER dans l'arborescence "plugin.tx_automaketemplate_pi1". Ce cObjet peut maintenant être utilisé plutôt que le cObject FILE afin de lire la maquette de Raphaël et d'y remplacer automatiquement les marqueurs et corriger les chemins relatifs.

Configurer le "Template Auto-parser"

Comme pour toutes les extensions, vous devriez consulter son manuel sur typo3.org afin de découvrir son fonctionnement et connaître les différentes possibilités de configuration. Cliquez ce lien afin d'accéder au tableau des propriétés de ce cObject.

Malheureusement nous n'y trouvons pas d'exemple de code -ce qui aurait été formidable (une simple suggestion à l'intention de l'auteur de cette extension). Nous allons donc proposer un tel exemple maintenant.

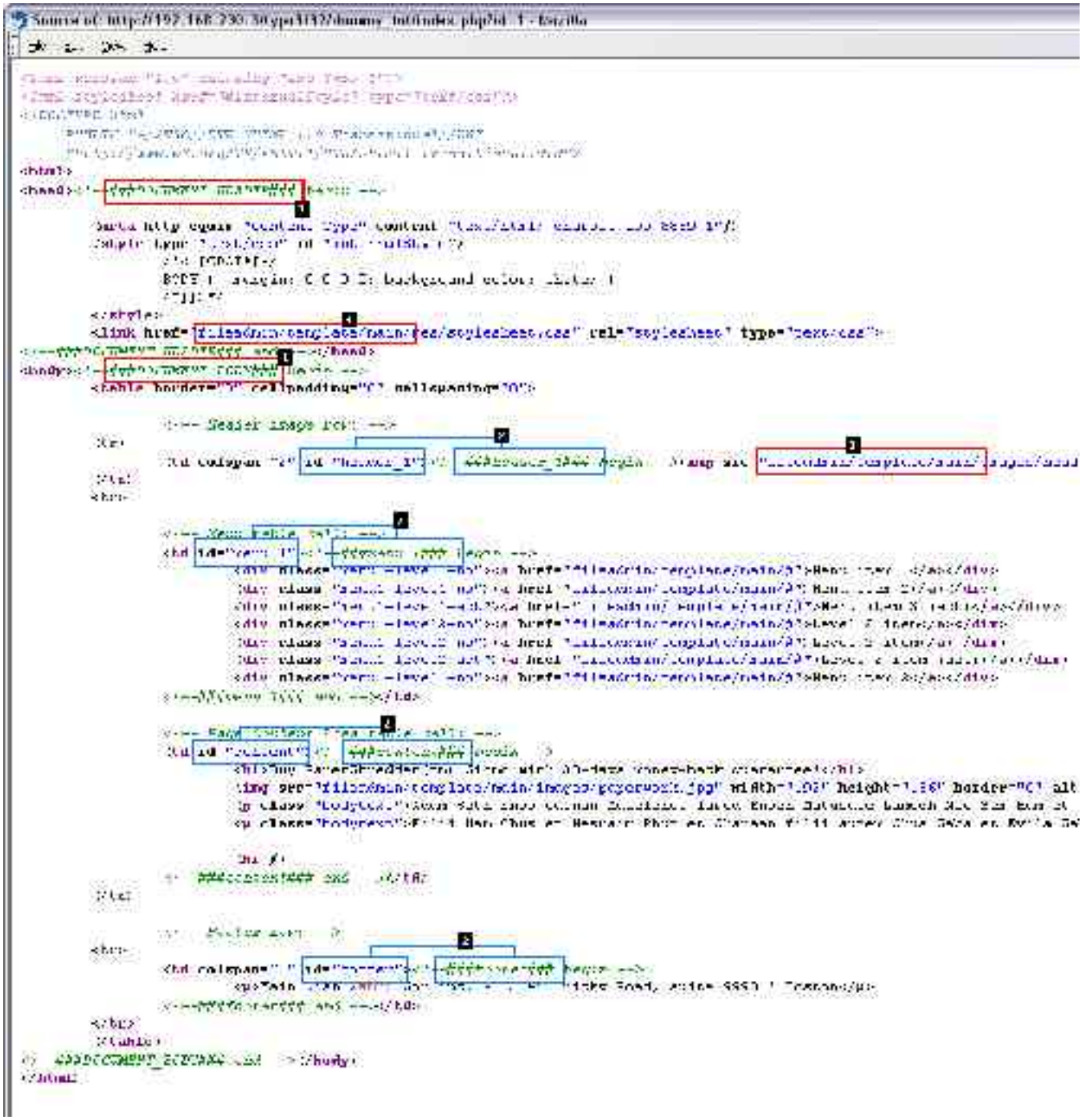
Afin de vous faire comprendre clairement ce que fait le "Template Auto-parser" je vais simplement insérer le contenu généré par le plugin comme seul contenu de la page puis je configurerais l'object PAGE "page" afin qu'il ne génère pas les headers et footers habituels.

Voici le champ Setup du gabarit :

```
# Configuring the Auto-Parser:
plugin.tx_automaketemplate_pi1 {
    # Read the template file:
  content = FILE
  content.file = fileadmin/template/main/template_1.html

    # Here we define which elements in the HTML that
    # should be wrapped in subpart-comments:
  elements {
    BODY.all = 1
    BODY.all.subpartMarker = DOCUMENT_BODY

    HEAD.all = 1
    HEAD.all.subpartMarker = DOCUMENT_HEADER
    HEAD.rmTagSections = title
```

```
    TD.all = 1
  }

    # Prefix all relative paths with this value:
  relPathPrefix = fileadmin/template/main/
}


# Default PAGE object:
page = PAGE
page.typeNum = 0
page.config.disableAllHeaderCode=1

page.10 =< plugin.tx_automaketemplate_pi1
```

Enregistrez le gabarit et visualisez la page en frontend. Vous devriez voir exactement ce que le fichier fileadmin/template/main/template_1.html donne :



Et alors, vous me direz ... ? C'est déjà pas mal, mais si vous regardez le code généré, vous comprendrez pourquoi :

Comme vous le constatez, deux choses se sont produites :

- Tous les blocs de la maquette ont été automatiquement inclus dans des sous-parties (1&2).
- Tous les liens / références relatifs ont été préfixés par le chemin "fileadmin/template/main/" (3)

Ceci est du au fait que l'analyseur de maquette "Template Auto-parser" a été configuré de la manière suivante pour traiter ces éléments :

```
...
  elements {
    BODY.all = 1
    BODY.all.subpartMarker = DOCUMENT_BODY

    HEAD.all = 1
    HEAD.all.subpartMarker = DOCUMENT_HEADER
    HEAD.rmTagSections = title

    TD.all = 1
  }
...
```

(Les "elements" sont tous les tags HTML ayant un tag de début ET un tag de fin comme par exemple le tag <td>.Les tags simples (sans tags de fin comme <img>)doivent être définis par la propriété "single" de l'analyseur de maquette)

Ainsi, la configuration des ces éléments indique que :

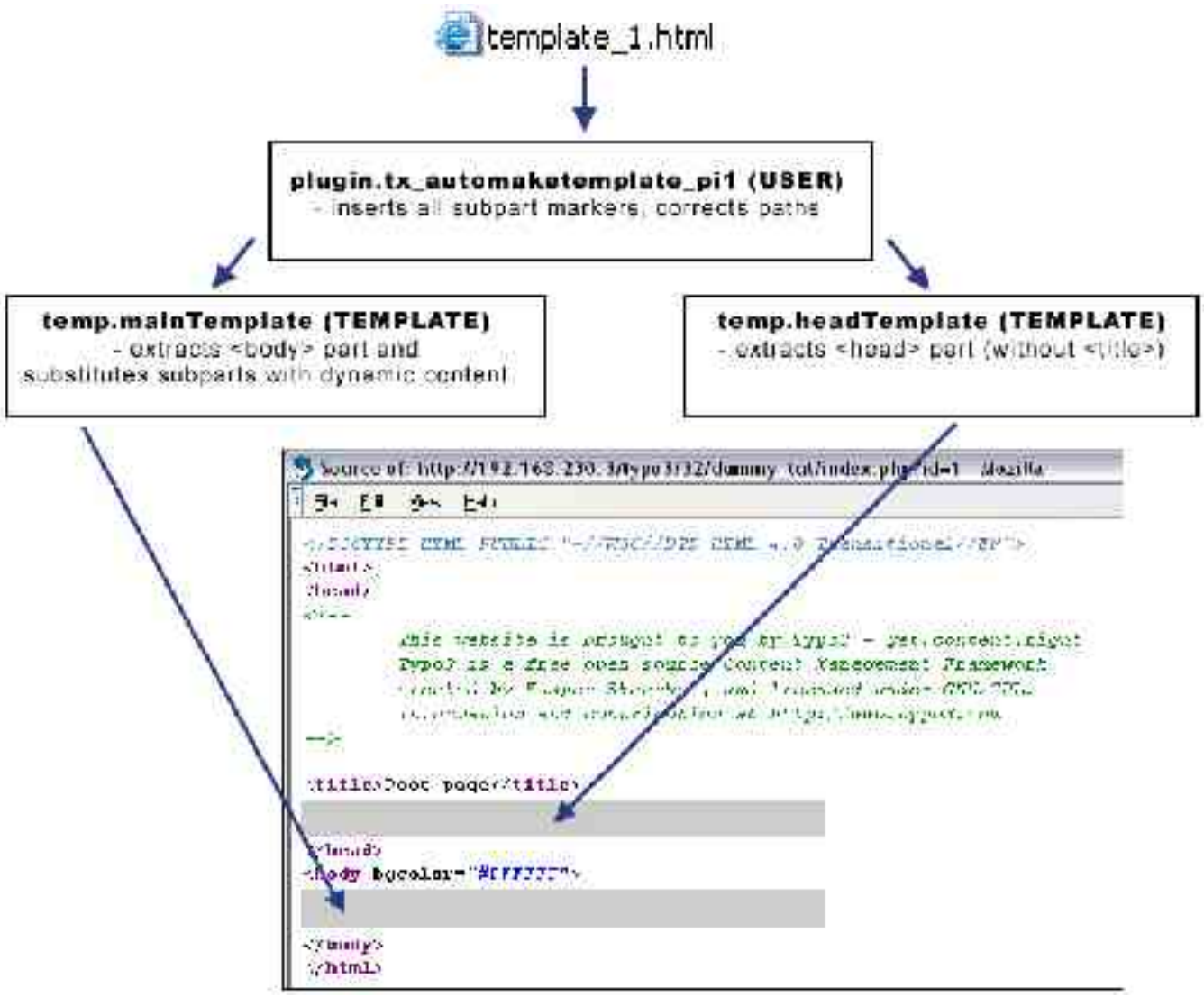
- a) l'élément <body>doit être encapsulé dans le marqueur de sous-partie "###DOCUMENT_BODY###"
- b) l'élément <head>doit être inclus dans le marqueur de sous-partie "###DOCUMENT_HEADER###".De plus, toute section <title>doit être retirée. (Nous ne voulons pas que le tag <title>de Raphaël fasse doublon avec celui généré par Typo3 ...)
- c) Tous les éléments<td>doivent entourés . Comme il n'y a pas de .subpartMarker de défini, seuls les tags avec un attribut "id" ou "class" seront wrappés dans une sous-partie dont le marqueur correspondra à l'id/class. Ainsi, le tag <td id="menu_1">verra le contenu *à l'interieur de ces balises* encapsulé entre des marqueurs de sous-parties "<!–###menu_1###->...<!–###menu_1###->".

Comment pouvons nous tirer partie de cette fonctionnalité ?

L'analyseur de maquettes permet à Raphaël de développer ses maquettes en utilisant les feuilles de styles et en utilisant avec discernement les attributs id / class. Dans le même temps, ces attributs constitueront des marqueurs permettant à Typo3 de remplacer des portions de la maquette par du contenu dynamique ! La technique est donc aisée pour Rapahël (lle designer), pose moins de difficultés à Benoît (le développeur) et rassure le comptable de Picouto puisque moins de main d'oeuvre sera nécessaire pour convertir à la main les maquettes statiques en maquettes dynamiques avec marqueurs de sous-parties.

Synthèse

Voici une illustration de ce que nous souhaitons obtenir :

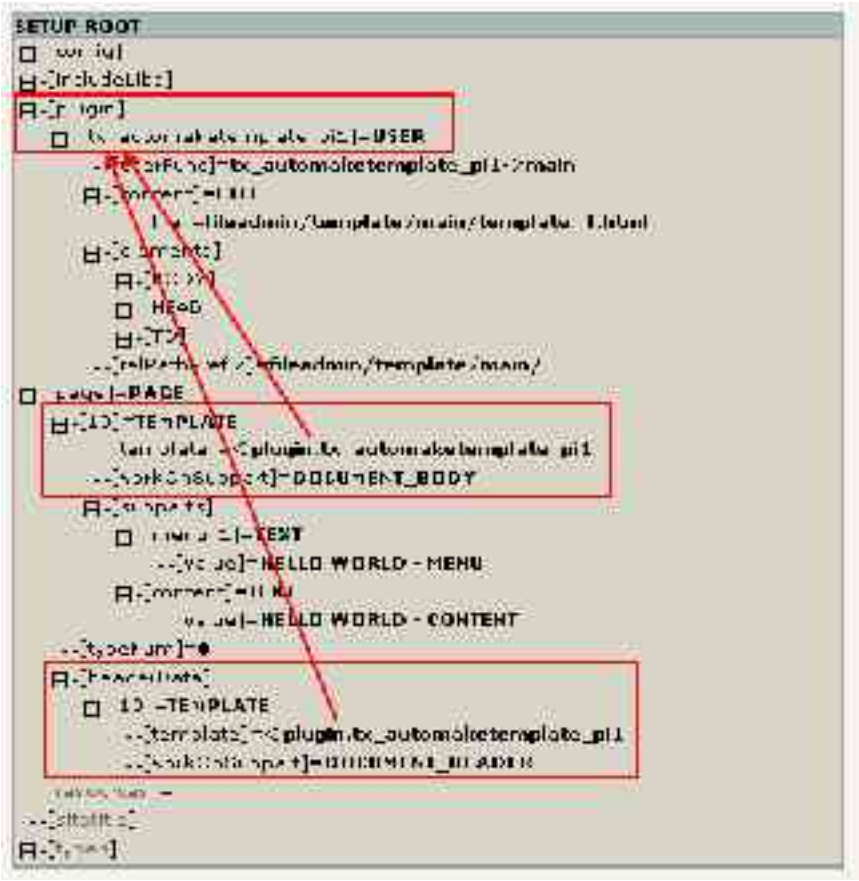


Le fichier de la maquette est lue par l'Auto-Parser, le résultat alors passé au cObject TEMPLATE qui substitut les sous-parties

et les insère au niveau du body et du header. Ceci sera obtenu grâce au TypoScript ci-dessous (champ Setup du gabarit). Ce code est assez long, mais nous prendrons le temps de l'analyser.

```
# Configuring the Auto-Parser for main template:
plugin.tx_automaketemplate_pi1 {
    # Read the template file:
  content = FILE
  content.file = fileadmin/template/main/template_1.html

    # Here we define which elements in the HTML that
    # should be wrapped in subpart-comments:
  elements {
    BODY.all = 1
    BODY.all.subpartMarker = DOCUMENT_BODY

    HEAD.all = 1
    HEAD.all.subpartMarker = DOCUMENT_HEADER
    HEAD.rmTagSections = title

    TD.all = 1
  }

    # Prefix all relative paths with this value:
  relPathPrefix = fileadmin/template/main/
}


# Main TEMPLATE cObject for the BODY
temp.mainTemplate = TEMPLATE
temp.mainTemplate {
    # Feeding the content from the Auto-parser to the TEMPLATE cObject:
  template =< plugin.tx_automaketemplate_pi1
    # Select only the content between the <body>-tags
  workOnSubpart = DOCUMENT_BODY

    # Substitute the ###menu_1### subpart with some example content:
  subparts.menu_1 = TEXT
  subparts.menu_1.value = HELLO WORLD - MENU

    # Substitute the ###content### subpart with some example content:
  subparts.content = TEXT
  subparts.content.value = HELLO WORLD - CONTENT
}


# Main TEMPLATE cObject for the HEAD
temp.headTemplate = TEMPLATE
temp.headTemplate {
    # Feeding the content from the Auto-parser to the TEMPLATE cObject:
  template =< plugin.tx_automaketemplate_pi1
    # Select only the content between the <head>-tags
  workOnSubpart = DOCUMENT_HEADER
}



# Default PAGE object:
page = PAGE
page.typeNum = 0

# Copying the content from TEMPLATE for <body>-section:
page.10 < temp.mainTemplate

# Copying the content from TEMPLATE for <head>-section:
page.headerData.10  < temp.headTemplate
```

On obtient la structure suivante :



Comme vous pouvez le constater, les cObjects "temp.mainTemplate" et "temp.headTemplate" ont été copiés à leurs positions respectives dans l'arborescence des objets. Chacun d'eux font référence au cObject USER du plugin d'analyse de Template (*plugin.tx_automaketemplate_pi1=USER*).

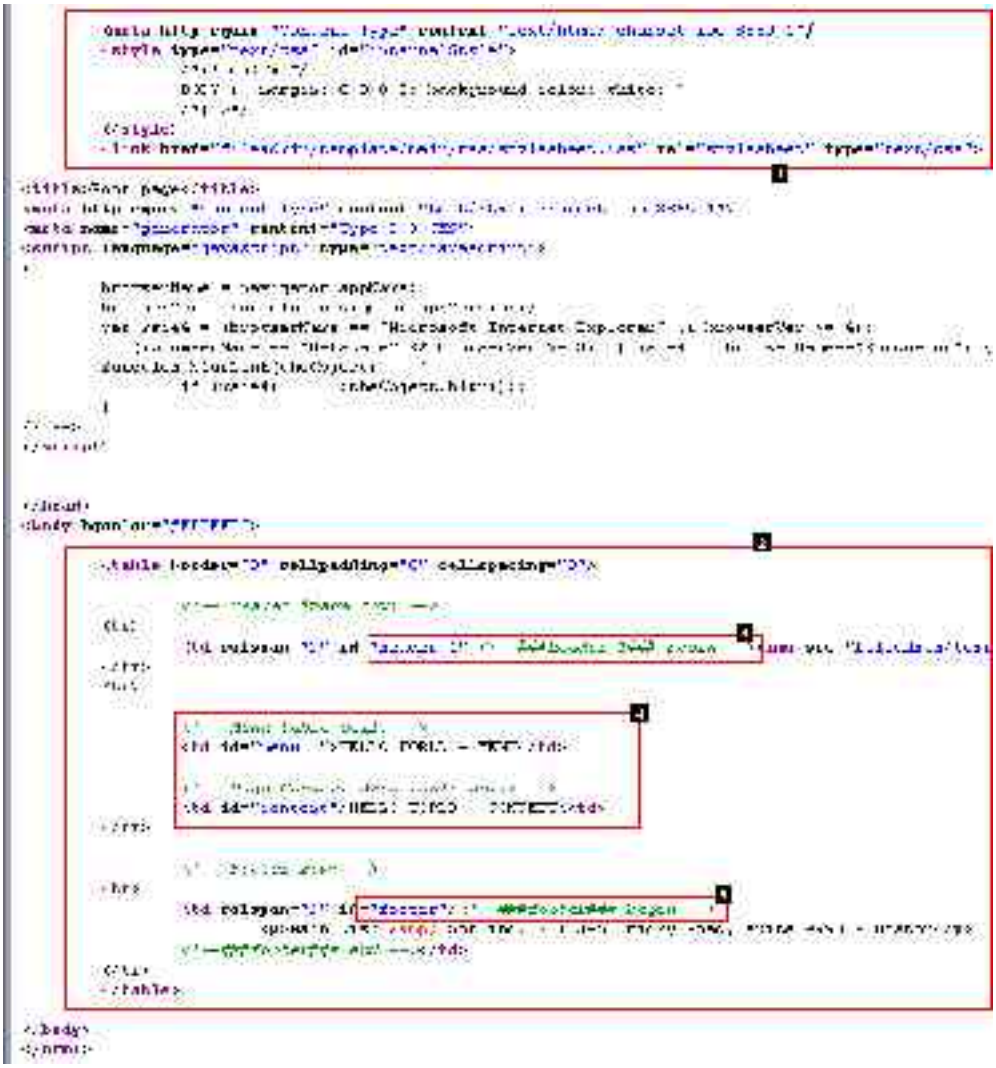
Notez également que l'objet "page.headerData" est un tableau d'objets définissant le contenu de la section "head" de la page. Ceci vous montre comment insérer du contenu dans la partie <head> des pages générées par Typo3.

Résultat

Le résultat est le suivant :

Le contenu des zones menu et contenu ont été remplacées par des cObject TEXT de test !

Voici le code source :



1. La partie <head> sans le tag <title> tag est inséré
2. La partie <body> est insérée et le contenu est substitué
3. Les sous-parties ###header_1### et ###footer### ne sont pas substituées car elles ne sont pas définies dans le cObject TEMPLATE "temp.mainTemplate" ! Leur présence est donc parfaitement normale.
4. Les sous-parties ###menu_1### et ###content### sont substituées conformément à ce que nous avions défini des le cObject TEMPLATE "temp.mainTemplate".

Résumé

Typo3 extrait automatiquement de la maquette de Raphaël les parties <head> et <body>, corrige les chemins relatifs des images, liens et feuilles de style, effectue les substitutions de contenu dynamique et insère enfin les sections <head> et <body> dans la page retournée par Typo3.
Le canevas est maintenant en place – nous devons simplement ajouter les contenus dynamiques que sont les menus et le contenu principal. La manière la pus efficiente d'effectuer ceci est d'utiliser les cObjects TypoScript permettant de construire des menus et des contenus.

Créer le menu

Le menu en 2 niveaux à gauche de la maquette doit être généré dynamiquement afin de refléter fidèlement l'arborescence des pages. Bien que Raphaël ait créé une superbe maquette pour les menus avec des marqueurs pour chaque item, il n'est pas aisé d'extraire ces marqueurs et de les remplacer en utilisant l'analyseur de maquette comme nous venons de le faire. Nous allons plutôt coder intégralement le menu en Typoscript. Ceci signifie que Benoît (le développeur) doit identifier et extraire ce qui correspond à une ligne du menu dans la maquette de Raphaël. Il pourra ensuite l'utiliser dans le gabarit pour construire le générateur de menus. Ceci implique que Raphaël ne peut pas changer le schéma de base du menu sans en notifier Benoît qui devra répercuter ces changements dans le gabarit. Cependant, si Raphaël construit sa maquette intelligemment -en utilisant des CSS par exemple, il est possible d'apporter de nombreuses modifications au style du menu sans toucher au gabarit. D'une manière générale, il est préférable d'utiliser les feuilles de style pour toute la gestion des effets visuels.

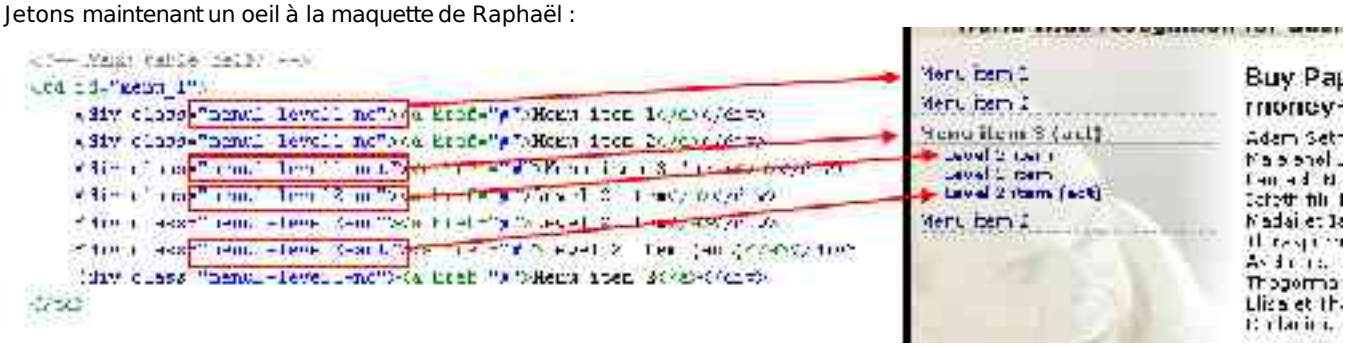
Des objets du menu et leurs états

Le premier des cObject à utiliser pour générer le menu est le HMENU. Cet objet appelle les autres cObjects de menu : TMENU ou GMENU ou GMENU_LAYERS etc. pour chaque niveau de menu que nous voulons afficher (le type d'affichage du niveau dépendant du type de cObject inséré : texte, graphique, menus déroulants ...).

Pour chaque objet du menu (par exemple un TMENU ou un GMENU), vous pouvez définir des propriétés générales affectant le niveau (profondeur) du menu qu'il représente et des propriétés spécifiques en fonction du type d'objet (par exemple la hauteur et la largeur d'un objet GMENU (graphique)). Ces propriétés spécifiques sont toujours définies pour un état donné de l'objet. L'état normal (NO) doit toujours être renseigné, mais il est également possible de renseigner l'état actif de l'objet (ACT), c'est à dire la présentation de cette ligne du menu si nous nous trouvons dans la page correspondante (ou une de ses sous-pages).

Cependant, l'objet de ce tutoriel n'est pas de détailler le cObject HMENU, reportez vous au manuelTypoScript by Example pour cela.

Jetons maintenant un oeil à la maquette de Raphaël :



Comme vous le voyez, il a utilisé un tag pour chaque ligne du menu, indépendamment de son niveau ou de son état. La différence entre ces lignes est faite par la valeur de l'attribut*class*. Ceci et particulièrement pertinent puisque cela va épargner beaucoup de travail à Benoît pour coder le menu. De plus, le design du menu est géré en dehors de Typo3, dans la feuille de style.

Notez également que Raphaël a fourni une présentation pour les menus "actifs".

L'implémentation des menus sera donc particulièrement facile. On procédera de la manière suivante :

Tout d'abord, définissez un objet temporaire au début du gabarit (avant "temp.mainTemplate") :

```
# Menu 1 cObject
temp.menu_1 = HMENU
  # First level menu-object, textual
temp.menu_1.1 = TMENU
temp.menu_1.1 {
    # Normal state properties
  NO.allWrap = <div class="menu1-level1-no"> | </div>
    # Enable active state and set properties:
  ACT = 1
  ACT.allWrap = <div class="menu1-level1-act"> | </div>
}
  # Second level menu-object, textual
temp.menu_1.2 = TMENU
temp.menu_1.2 {
    # Normal state properties
  NO.allWrap = <div class="menu1-level2-no"> | </div>
    # Enable active state and set properties:
  ACT = 1
  ACT.allWrap = <div class="menu1-level2-act"> | </div>
}
```

Notez les lignes en rouge qui contiennent les marqueurs HTML que Benoît a extrait de la maquette de Raphaël. Maintenant, ces marqueurs sont codés en dur dans le gabarit.

La seule chose que vous devez faire est de copier cet objet afin qu'il soit le cObjet affecté au "menu_1" de "temp.mainTemplate" :

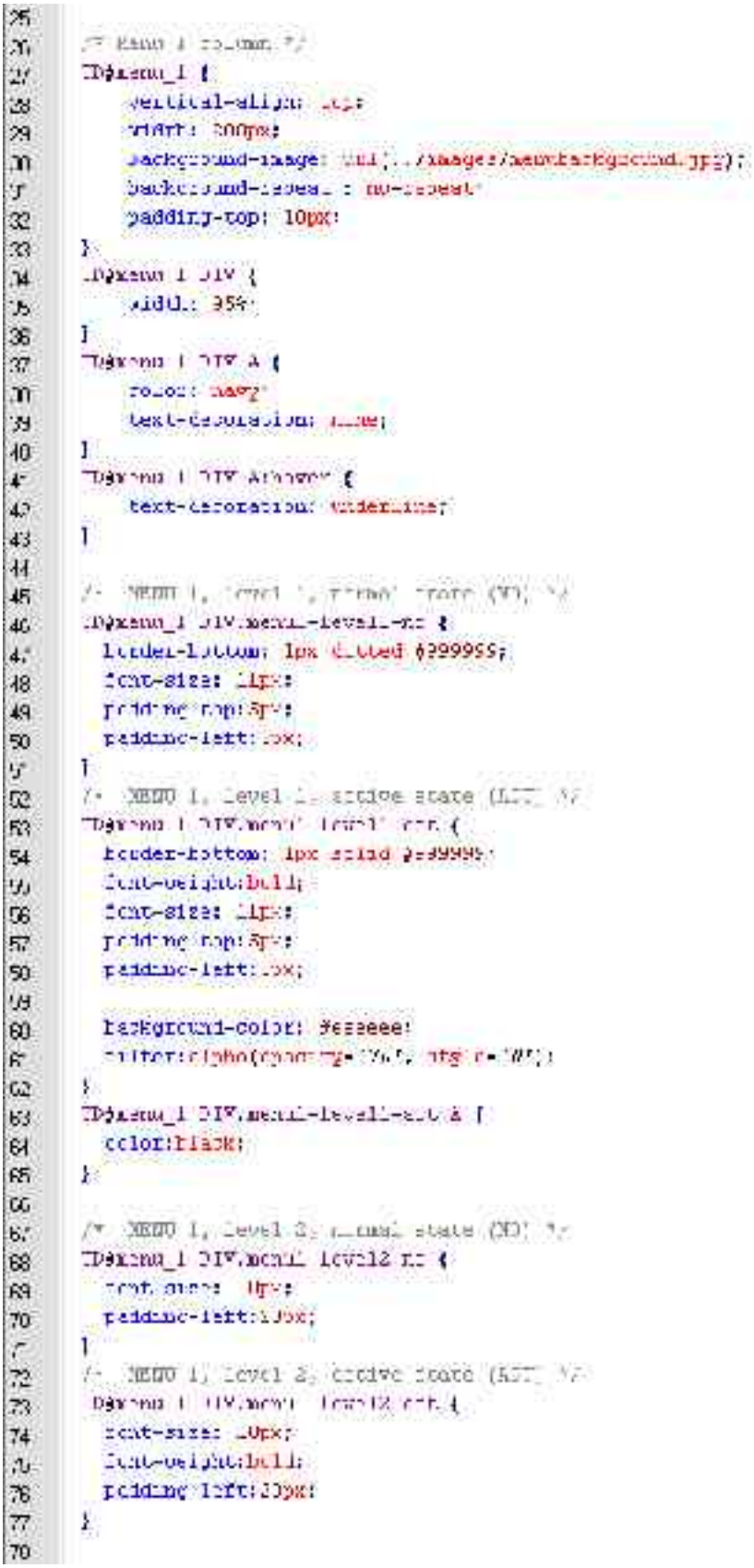
```
...
# Main TEMPLATE cObject for the BODY
temp.mainTemplate = TEMPLATE
temp.mainTemplate {
    # Feeding the content from the Auto-parser to the TEMPLATE cObject:
  template =< plugin.tx_automaketemplate_pi1
    # Select only the content between the <body>-tags
  workOnSubpart = DOCUMENT_BODY

    # Substitute the ###menu_1### subpart with dynamic menu:
  subparts.menu_1 < temp.menu_1

    # Substitute the ###content### subpart with some example content:
  subparts.content = TEXT
  subparts.content.value = HELLO WORLD - CONTENT
...
```

(La ligne en rouge vous montre le changement !)

... et le résultat qui se passe d'explications:

La structure des pages (voir ci-dessous) est clairement affichée dans le menu ci-dessus !

Mais sera-t-il facile de changer le look du menu me direz vous ? Éditez la feuille de style "fileadmin/template/main/res/stylesheet.css" :



Voyez ! Cette approche du problème vous permet de déporter les principaux facteurs contrôlant l'aspect visuel - maquette HTML et feuille de style – en dehors de Typo3 afin qu'ils soient directement accessibles et modifiables par Raphaël, le designer. Benoît pour sa part n'a eu qu'une charge de travail mineure. Il a seulement paramétré le moteur de front-end afin qu'il prenne en compte le design de Raphaël – juste un peu de TypoScript dans le gabarit.

Veuillez vous référer au manuel TypoScript by Example si vous voulez plus d'informations sur les HMENUs..

Insérer un contenu de page

La manière la plus simple de gérer les contenu de pages dans Typo3 est de créer des*éléments de contenu* dans la table *tt_content*. En fait je ne pense pas que quelqu'un procède autrement car c'est la manière la plus simple, la plus souple et la plus directe à mettre en œuvre. Cependant, cela peut être en contradiction avec vos *a priori* sur l'encapsulation de contenu structuré au sein des maquettes HTML.

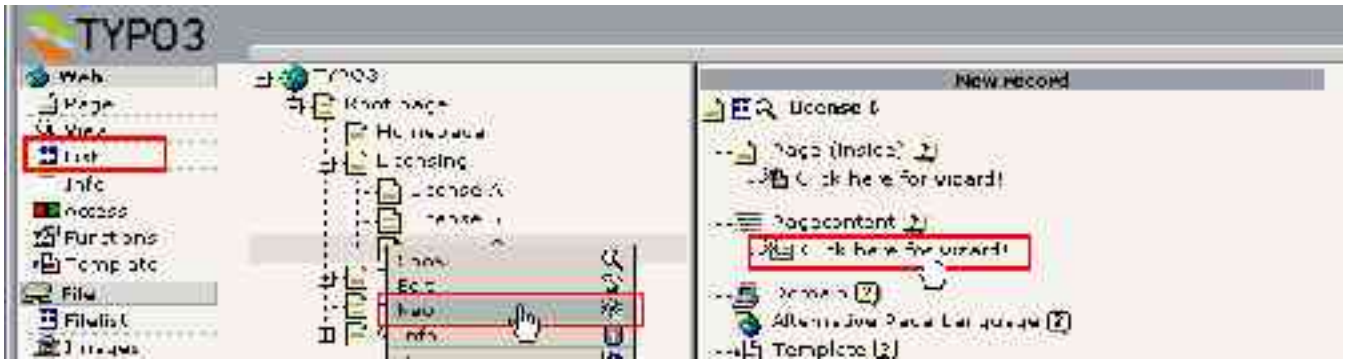
Les *éléments de contenu* ont plusieurs types prédéfinis (Text, Text w/image, Bulletlist, Table, Login box, etc) et généralement une maquette statique associée à du TypoScript permettant d'effectuer le rendu de ces éléments sans que vous ne vous occupiez de rien. Comme les éléments *de contenu* ont un type et que chaque type a de nombreuses options, il est impossible de générer ces *éléments de contenu* avec un systèmes de maquette fixe pour chaque type d'éléments. Cela ne fonctionnerait pas et vous pourrez vous en rendre compte lorsque vous aurez joué un peu avec ce concept. C'est pourquoi des cObjects combinés à des fonctions PHP sont invoquées pour générer le contenu en fonction de conditions complexes.

Alors, comment faire pour créer les contenus sans utiliser de templates HTML ? En fait, en utilisant la dernière technique Typo3 de génération de contenu - l'extension "css_styled_content" – tous vos contenus seront encapsulés dans des éléments HTML. Ceci déportera alors les décisions de design vers les feuilles de style externes !

Regardons comment cela fonctionne en commençant par créer un contenu.

Un élément de contenu

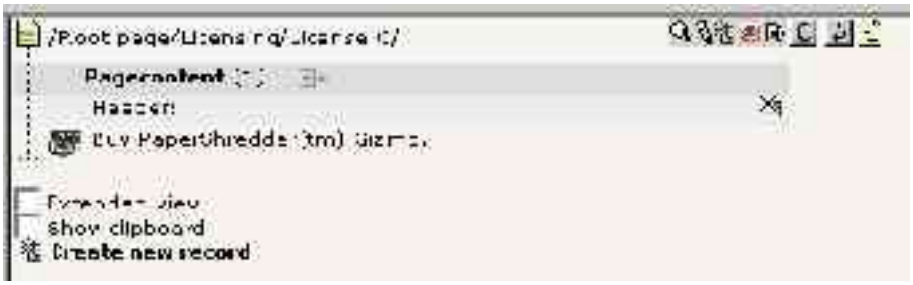
Dans le module Liste, cliquez sur l'icône de la page "licence C", puis "New" et enfin dans la frame de droite, cliquez sur le lien de l'assistant de contenu *(Click here for wizard).*



Sélectionnez à droite : "Text with image" et tout en bas de la page la colonne "Normal" :

Après avoir saisi votre contenu et l'avoir enregistré, vous devriez obtenir quelque chose comme ceci :

Nous avons maintenant un contenu dans la page "License C":



Inclure la maquette statique

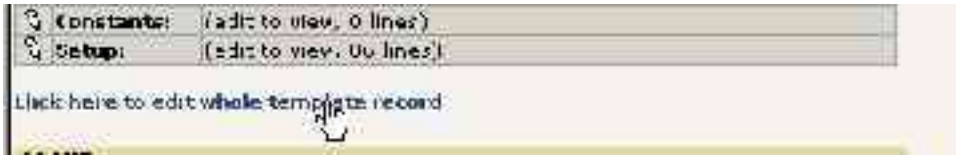
Afin d'afficher cet élément de contenu, il nous faut inclure une maquette statique qui fournira les centaines de lignes de TypoScript qui se chargeront du rendu. Ceci sera fait en éditant le gabarit.

Mais avant toute chose, il faut installer l'extension "css_styled_content":

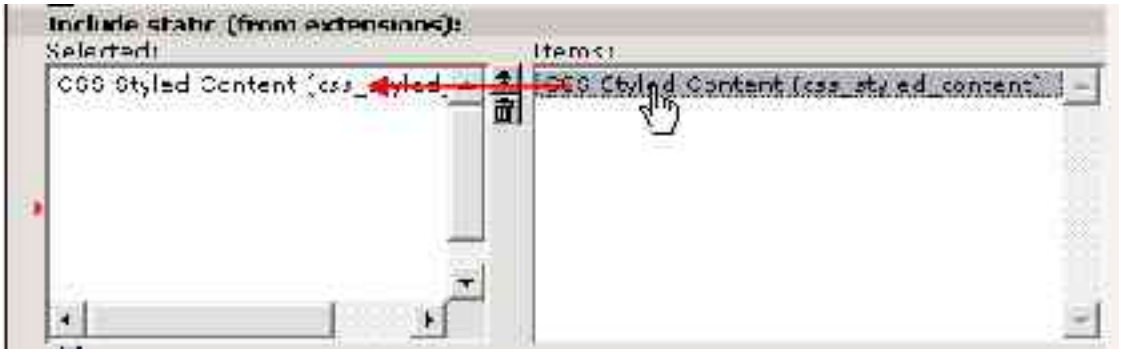
Dans le module "extension manager" / "Available extensions" , cliquez simplement sur le bouton "Install" de*CSS Styled Content* et sur le bouton*Accept* de la page suivante.

(Attention : au moment de l'écriture de ce manuel [mars 2003] le plugin "CSS Styled Content" n'est pas totalement terminé ! Il n'effectue le rendu que des éléments *Text, Text w/image, Bullet list et Table* ainsi que des "*Insert plugins*" et éventuellement quelques autres. Désolé ! Ce plugin sera à terme LE moyen de générer le contenu mais actuellement il vous montre simplement la direction que Typo3 prendra. Bien évidemment, le mécénat permettra d'accélérer le développement et / ou de changer les priorités ..)

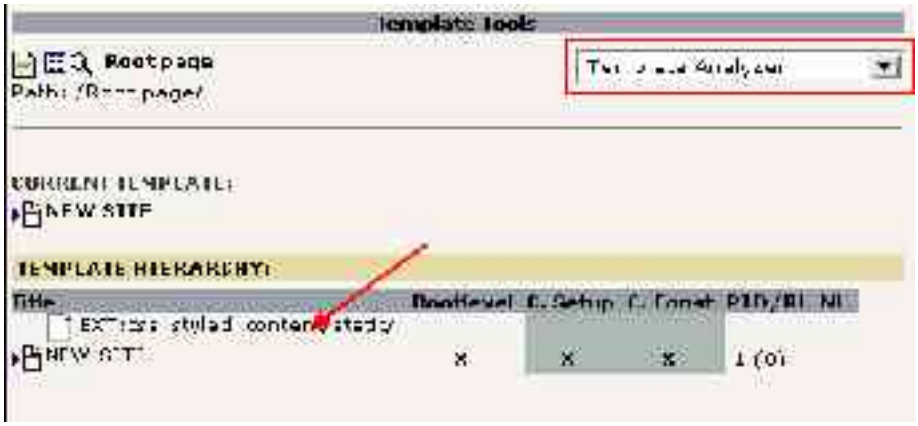
De retour dans le module Gabariat : cliquez sur ce lien :



Vous devriez voir apparaître dans la boîte "Include Static (from extensions)" le contenu "CSS Styled Content". Cliquez afin de l'ajouter



Cliquez dessus afin de l'ajouter, enregistrez la maquette et allez dans le *Template Analyzer*:



L'analyseur de gabarit (*Template Analyzer*) vous montre la hiérarchie des gabarit qui sont rattachés à la maquette principale (*main template record*). En incluant la maquette statique "*CSS Styled Content*", celle-ci a été insérée avant le TypoScript contenu dans le gabarit NEWSITE. Cela signifie que les objets définis dans *EXT:css_styled_content/.....* peuvent être copiés et utilisés dans le gabarit NEWSITE. C'est ce que nous allons faire maintenant car la maquette statique incluse contient l'objet "*style.content.get*" qui sélectionne tous les éléments de la colonne "*Normal*" pour la page courante.

Dans le gabarit on changera donc la définition de l'objet "temp.mainTemplate" de la manière suivante :

```
...
# Main TEMPLATE cObject for the BODY
temp.mainTemplate = TEMPLATE
temp.mainTemplate {
    # Feeding the content from the Auto-parser to the TEMPLATE cObject:
```

```
  template =< plugin.tx_automaketemplate_pi1
    # Select only the content between the <body>-tags
  workOnSubpart = DOCUMENT_BODY

    # Substitute the ###menu_1### subpart with dynamic menu:
  subparts.menu_1 < temp.menu_1

    # Substitute the ###content### subpart with some example content:
  subparts.content < styles.content.get
...
```

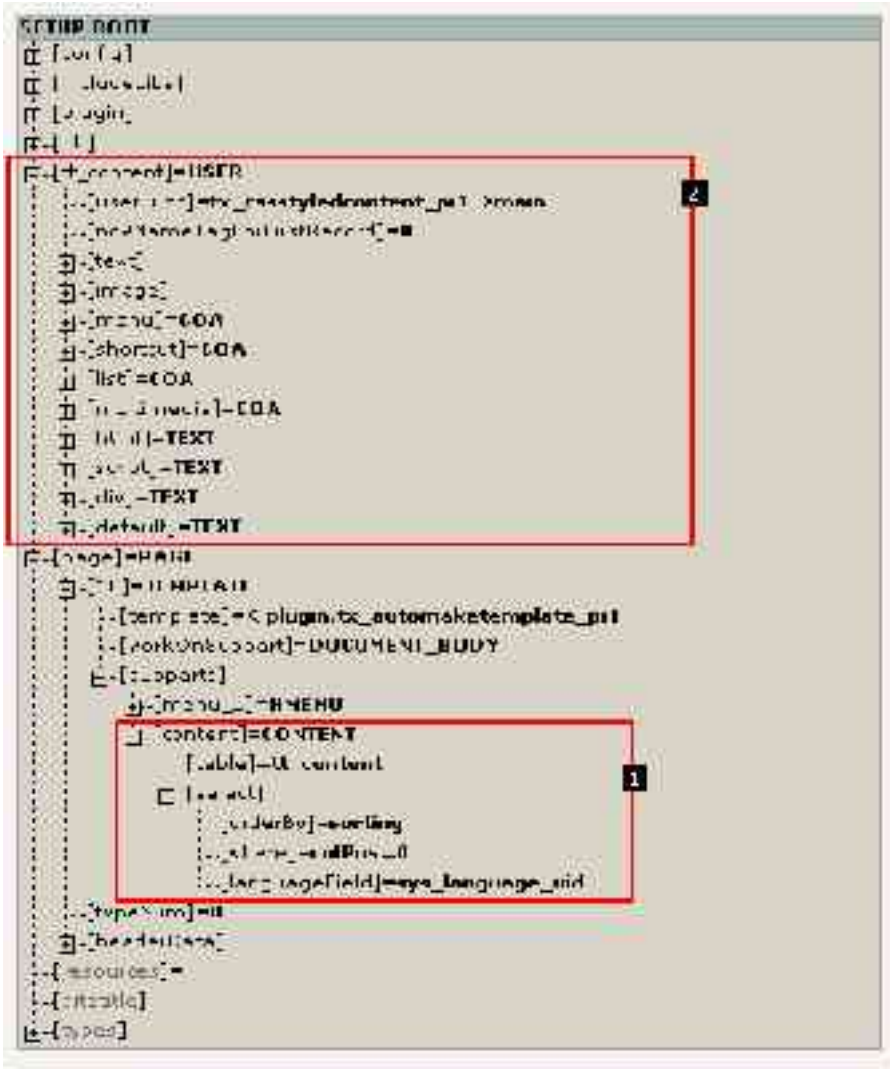
Si vous rafraîchissez le frontend pour la page "Licence C" vous devriez obtenir ceci (pensez à vider les caches si cela ne fonctionne pas (en bas du menu gauche dans l'interface d'administration)) :

Dans le source HTML on trouvera ceci :

1. Apparemment, le titre de l'élément de contenu a été rendu avec des tags <h1>. Si vous sélectionnez d'autres "Layout" pour ce titre, .vous changerez la manière dont il est rendu (par exemple <h2>pour un Layout 2).
2. Chaque ligne du corps de texte est encapsulée dans des tags <p>avec une classe css "bodytext". De cette manière, nous pouvons changer la présentation de nos contenus directement depuis la feuille de style ! Notons que Raphaël a déjà prévu la chose puisque le contenu d'exemple portait déjà des tags <pclass="bodytext">...
3. Chaque élément de contenu inséré portera un tag <a>dont le nom sera l'uid de l'élément. Ceci fournira des points d'ancrage qui pourront être utilisés dans les URLs pour se rendre directement à ce contenu au sein de la page
4. On trouve également le rendu pour les objets de menu de niveau 2 provenant du HMENU. Notez que le lien pointe automatiquement sur la bonne page et que la gestion d'événement onFocus est même prise en charge.

Et l'arbre des objets Typo ?

En copiant l'élément de contenu virtuel "styles.content.get", qu'avons nous réellement inséré dans l'arborescence des objets Typo ? Examinons cela avec l'*Object Browser* :



Nous remarquons deux choses intéressantes ;

1. Le chemin de l'objet "*styles.content.get*" contient un cObjet du type CONTENT. En examinant ses propriétés, on peut en déduire qu'il sélectionne des enregistrements de la table *tt_content* en les classant par le contenu du champ *sorting*. Pour plus d'informations consultez la documentation du cObject CONTENT..
2. Un nouveau TLO *(Top Level Object)* a été créé : "tt_content". Par défaut, le cObject CONTENT qui sélectionnera ses enregistrements dans la table *tt_content* utilisera ce TLO pour le rendu de chaque enregistrement trouvé ! Comme nous pouvons le constater, le TLO "tt_content" est défini comme une cObject de type USER invoquant une fonction PHP de l'extension "css_styled_content" – c qui a l'air correct puisque c'est justement l'extension que nous venons d'installer pour prendre en charge cela ! Pour plus de détails consultez la documentation de l'extention "css_styled_content".

Pour les insatiables curieux, nous pouvons examiner la classe "tx_cssstyledcontent_pi1" et décortiquer son fonctionnement (*la magie du libre !*). Voici un extrait de la fonction *main()* :

```
function main($content,$conf)    {
      $this->conf = $conf;

      // This value is the Content Element Type - determines WHAT kind of element to render...
      $CTypeValue = (string)$this->cObj->data["CType"];
      $content="";
      switch($CTypeValue)        {
            case "header":
                  $content = $this->getHeader().$this->render_subheader();
            break;
            case "bullets":
                  $content = $this->getHeader().$this->render_bullets();
            break;
            case "table":
                  $content = $this->getHeader().$this->render_table();
            break;
            case "text":
                  $content = $this->getHeader().$this->render_text();
            break;
            case "image":
                  $content = $this->getHeader().$this->render_image();
            break;
```

```
                case "textpic":
                    $content = $this->render_textpic();
                break;
...
```

Un peu de ménage !

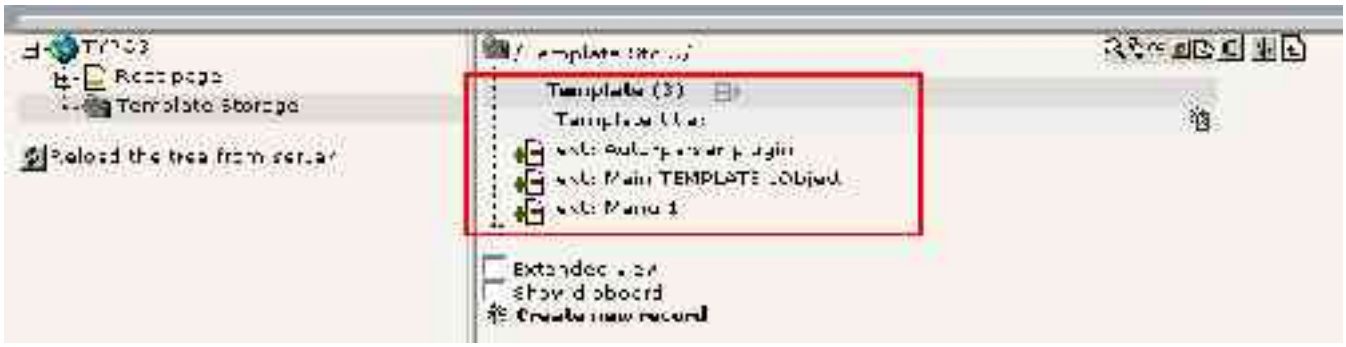
Nous avons maintenant terminé notre site Web. Benoît et Raphaël ont achevé leur travail et Mr Picouto en a profité pour saisir tous les contenus sur l'ensemble des pages du site. Le site web initial et donc en place. Il est basé sur des fichiers en pur HTML dans le répertoire "fileadmin/template/main/" avec un menu et des contenus remplaçant dynamiquement les zones nécessaires.

Une meilleure structure de maquette

La dernière petite chose à faire est de réorganiser le gabarit. Bien que le TypoScript utilisé soit réduit au minimum, nous pouvons constater que ce gabarit devient rapidement énorme et peu pratique. La solution est de créer une série de gabarit à inclure dans le gabarit principal.

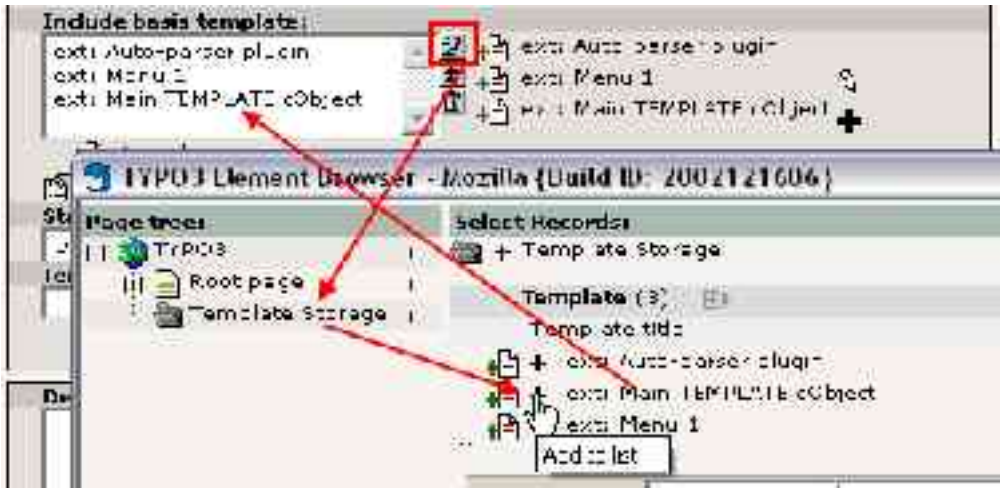
Tout d'abord, nous allons créer une nouvelle page à la racine de l'arborescence. Utilisez le type "*sysFolder*" (ce qui correspond à un espace de stockage) et appelez le *Template Storage.*

Puis créez trois gabarit comme dans la capture ci-dessous :



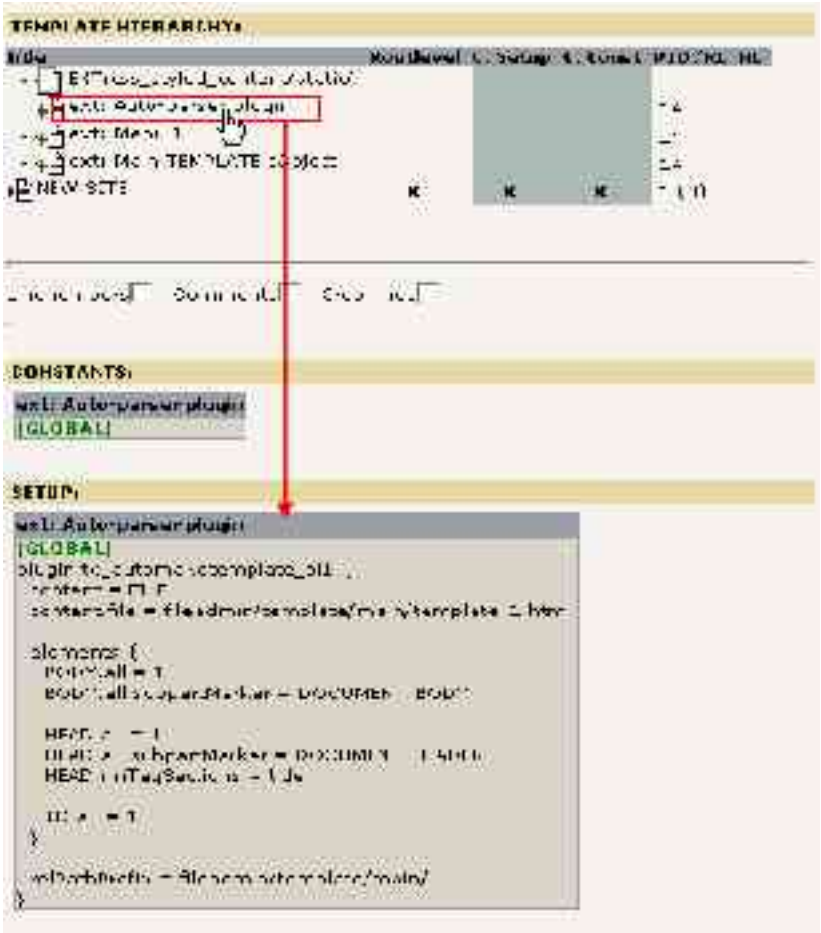
Ne cochez aucune case "Clear" ! Copiez simplement les définitions d'objets "*temp.xxxx*" depuis le gabarit "NEW SITE" dans ces nouveaux gabarit. L'ordre de ces gabarit n'a aucune importance.

L'étape suivante est d'inclure ces trois gabarit dans le gabarit principal ("NEW SITE"). Pour cela éditez le et ajoutez les trois gabarit dans la boîte *Include Basis template*:



Ici, l'ordre est important : la première des maquettes listée est insérée la première. Comme le gabarit "ext: Main TEMPLATE cObject" effectue des copies par exemple de l'objet "temp.menu_1" il doit être inclus en dernier.

Enregistrez le gabarit et visualisez la nouvelle structure dans l'analyseur de gabarit.



Vous voyez que le gabarit statique (*EXT:css_styled_content*) a été inclus en premier puis les trois gabarit de notre répertoire de stockage et enfin le gabarit principal. Pour chacun d'eux, vous pouvez cliquer sur le titre et consulter son contenu en bas de page.

Quelques considérations sur le design HTML

Cette section du tutoriel vous a montré qu'il est possible d''intégrer une maquette d'un designer Web d'une manière souple et confortable dans Typo3 en conservant une parfaite compatibilité avec son travail originel (aucun marqueur n'a été inséré manuellement dans la maquette). De plus, ceci ne met en œuvre que des technologies familières (HTML / CSS) - pas de choses complexes comme des feuilles de styles XSLT (bien que qu'un outil de rendu XSLT puisse être développé au niveau PHP par Benoît si tel était le besoin (par exemple pour publier des documents XML provenant de Openoffice.org)).

Cependant, bien que la liberté de Raphaël, le designer, est à peu près totale, il est nécessaire qu'il comprenne les bases des contenus dynamiques et statiques.

Parties dynamiques, parties statiques

Raphaël – et vraisemblablement toute l'équipe web – doit en premier lieu bien comprendre quelles parties des pages sont statiques et quelles parties sont dynamiques. Les parties statiques sont toutes les parties que Typo3 ne touche pas. Les parties dynamiques sont elles remplacée par Typo3 avec le contenu tiré de la base de données.

Lorsque les zones dynamiques sont clairement identifiées, Raphaël doit s'assurer de deux choses :

- Cette zone doit être encapsulée dans un seul élément HTML. Ce peut être un élément , ou <td>.
- Ce élément d'encapsulation doit avoir un attribut *id* ou *class* que le plugin *Template AutoParser* correctement configuré pourra retrouver et encadre de marqueurs de sous-parties. Par exemple, <td id="content"> pour une cellule qui hébergera le contenu.

Cela , n'a pas d'incidence de laisser le contenu d'exemple dans la maquette puisqu'il sera remplacé par le contenu dynamique.

Simplifiez la marquage, utiliser les feuilles de style

Nous sommes en 2003 (voire plus) et il serait peut-être temps d'abandonner les tags <font>, les attributs bgcolor et de tout mettre dans des feuilles de style externes. Pour un CMS, cette technique sert bien l'objectif de mettre le maximum de choses possibles en externe et n'utiliser que le strict nécessaire de ce qui est spécifique du CMS (comme les gabarit TypoScript).

Dans cette partie du tutoriel, le meilleur exemple est sûrement le menu de niveau 2 :

```
<!-- Menu table cell: -->
<td id="menu_1">
      <div class="menu1-level1-no"><a href="#">Menu item 1</a></div>
      <div class="menu1-level1-no"><a href="#">Menu item 2</a></div>
      <div class="menu1-level1-act"><a href="#">Menu item 3 (act)</a></div>
      <div class="menu1-level2-no"><a href="#">Level 2 item</a></div>
      <div class="menu1-level2-no"><a href="#">Level 2 item</a></div>
      <div class="menu1-level2-act"><a href="#">Level 2 item (act)</a></div>
      <div class="menu1-level1-no"><a href="#">Menu item 2</a></div>
</td>
```

Regardons juste un de ces éléments (celui en rouge). Il aurait également pu être implémenté de la manière suivante :

```
<!-- Menu table cell: -->
<td id="menu_1">
            <!-- Menu item level 1, begin -->
      <table border="0" cellpadding="0" cellspacing="0" width="95%"><tr>
            <td bgcolor="#eeeeee"><font face="verdana" size="2">
               <a href="#" class="menu1-items">Menu item 1</a>
            </font></td>
      </tr>
      <tr>
            <td><img src="gray_dotted_line.gif" width="180" height="1" alt=""/></td>
```

```
                </tr>
                </table>
                      <!-- Menu item level 1, end -->
                ...
          </td>
```

Rappelez vous cependant que le TMENU à l'intérieur du gabarit a été défini pour entourer l'élément au moyen d'un :

```
temp.menu_1.1 {
    # Normal state properties
  NO.allWrap = <div class="menu1-level1-no"> | </div>
...
```

donc en utilisant cette implémentation, il faudra encadrer les éléments de la manière suivante :

```
temp.menu_1.1 {
    # Normal state properties
  NO.allWrap ( <table border="0" cellpadding="0" cellspacing="0" width="95%"><tr>
                      <td bgcolor="#eeeeee"><font face="verdana" size="2">
                        |
                      </font></td>
                </tr>
                <tr>
                      <td><img src="gray_dotted_line.gif" width="180" height="1" alt=""/></td>
                </tr>
                </table>
)
...
```

Ces modifications ne seront cependant pas suffisantes. Il faudra également ajouter la classe correspondant au tag <a>avec une autre propriété de l'état NO et préfixer l'image "gray_dotted_line.gif" avec le chemin correct du fichier – quelque chose comme "*fileadmin/template/main*".

N'utilisez pas de classe pour les liens dynamiques !

Beaucoup de designers seraient tentés de marquer leurs éléments de menu et les autres liens en utilisant des attributs de classe pour le tag <a>, par exemple :

```
          <!-- Menu table cell: -->
          <td id="menu_1">
                <div><a href="#" class="menu1-level1-no">Menu item 1</a></div>
                ...
          </td>
```

Cependant cela ne sera pas le cas car le tag <a>est généré dynamiquement par Typo3. Afin de faire fonctionner le style pour le TMENUITEM, il faudra utiliser une propriété TypoScript supplémentaire :

```
temp.menu_1.1 {
    # Normal state properties
  NO.allWrap = <div> |</div>
  NO.ATagParams = class="menu1-level1-no"
...
```

Ceci complexifie les propriétés TypoScript et donne moins de flexibilité dans la feuille de style. Par exemple, si vous souhaitez contrôler l'apparence d'un lien par une feuille de style, vous devrez employer dans ce cas :

```
A.menu1-level1-no { color: navy; }
A.menu1-level1-no:hover { color: red; }
```

Mais vous ne pouvez pas contrôler l'encapsulation par l'élément car il n'est pas identifié par un attribut*class* ou *id*. Par contre, si vous considérez que le tag <a>ne doit pas avoir d'attribut*class* comme indiqué précédemment, le tag peut, lui porter un tel attribut. En combinant ces différentes possibilités, vous pouvez contrôler à la fois le bloc ainsi que les liens qu'il contient :

```
DIV.menu1-level1-no  { padding: 2px 2px 2px 2px; }
DIV.menu1-level1-no  A { color: navy; }
DIV.menu1-level1-no  A:hover { color: red; }
```

Gardez vous des row/colspans!

C'est sûrement la plus grande fonctionnalité pour le contenu dynamique – l'utilisation de colspan. C'est cependant hors de question. Considérez par exemple l'implémentation suivante du menu :

```
<tr>

      <!-- Menu table cell: -->
      <td class="menu1-level1-no"><a href="#">Menu item 1</a></td>

      <!-- Page Content Area table cell: -->
      <td id="content" rowspan="7">
            . . .
      </td>
</tr>
<tr>
```

```
      <td class="menu1-level1-no"><a href="#">Menu item 2</a></td>
</tr>
<tr>
      <td class="menu1-level1-act"><a href="#">Menu item 3 (act)</a></td>
</tr>
<tr>
      <td class="menu1-level2-no"><a href="#">Level 2 item</a></td>
</tr>
<tr>
      <td class="menu1-level2-no"><a href="#">Level 2 item</a></td>
</tr>
<tr>
      <td class="menu1-level2-act"><a href="#">Level 2 item (act)</a></td>
</tr>
<tr>
      <td class="menu1-level1-no"><a href="#">Menu item 2</a></td>
</tr>
```

Ceci est impossible à implémenter. Le premier objet est une ligne avec la cellule de contenu. Les 6 autres objets du menu sont dans des lignes séparées du tableau. De plus, la cellule de contenu nécessite un rowspan=*7* afin qu'elle s'étende sur toutes les lignes du menu !

C'est très nettement une mauvaise conception d'un point de vue technique :

1. les marqueurs du menu sont dispersés au sein du code HTML.
2. Il existe une dépendance entre l'attribut rowspan et le nombre de lignes du menu (qui est dynamique !).

Par conséquent, les designers devraient s'abstenir de telles constructions. Raphaël et ses confrères devraient plutôt concevoir ces marqueurs pour les éléments dynamiques afin qu'il soient facilement répétables. Voici quelques exemples :

Celui de notre maquette :

```
<!-- Menu table cell: -->
<td id="menu_1">
      <div class="menu1-level1-no"><a href="#">Menu item 1</a></div>
      <div class="menu1-level1-no"><a href="#">Menu item 2</a></div>
      <div class="menu1-level1-act"><a href="#">Menu item 3 (act)</a></div>
      ...
</td>
```

Une autre construction employant des tableaux :

```
<!-- Menu table cell: -->
<td id="menu_1">
      <table border="0" cellspacing="0" cellpadding="0">
            <tr><td class="menu1-level1-no"><a href="#">Menu item 1</a></td></tr>
            <tr><td class="menu1-level1-no"><a href="#">Menu item 2</a></td></tr>
            <tr><td class="menu1-level1-act"><a href="#">Menu item 3 (act)</a></td></tr>
               ...
      </table>
</td>
```

Ces exemples peuvent être encore plus complexes à l'image du précédent qui incluait chaque élément du menu dans un tableau. L'important n'est cependant pas la quantité de HTML impliqué, mais la nature répétitive du marqueur pour mettre en oeuvre les éléments dynamiques.

Il est important que Raphaël (et l'ensemble des designers) comprennent bien ces problèmes afin que Benoît ne passe pas des heures à tenter de faire fonctionner une maquette inadéquate – incidemment, Mr Picouto pourrait être tellement content de l'achèvement rapide du site grâce aux efforts de Raphaël et Benoît qu'il pourrait bien les inviter à déjeuner !

Note Finale.

Ce tutoriel a ses parties 2 et 3 dans un autre document. Ces parties – en particulier le 2 – sont des suites de celle-ci et sont basées sur le même cas "réel" impliquant l'équipe web (Raphaël, Benoît, Mr Picouto). Mais avant de continuer avec ceux-ci, veuillez noter que vous aurez peut être besoin de compétences supplémentaires pour lire les parties 2 et 3. Avant de commencer leur lecture, il serait raisonnable de vous entraîner avec Typo3, lire quelques autres tutoriels et éventuellement augmenter vos compétences techniques et de développement.

Appendice: Créer une template en TypoScript pur

Introduction

Certains préfèrent créer des sites web sans avoir recours à des maquettes externes en HTML. Ceci est possible bien que cela soit plus complexe et soit beaucoup moins répandu dans la communauté des développeurs Typo3. Je vais donc vous en donner un exemple. Si après la lecture, cela ne vous paraît pas une approche adéquate, oubliez le.

.... [Contenu à ajouter ici] ....

Changements :

26 novembre 2003 : Première version.

15 janvier 2004 : Corrections et ajout des notes de traductions.