

Современная разработка шаблонов, Часть 1 (МТВ/1)

Ключ расширения: **doc_tut_templselect**
Copyright 2003-2004, Kasper Skårhøj, <kasper@typo3.com>

Этот документ публикуется в соответствии с Open Content License
доступной на <http://www.opencontent.org/opl.shtml>

Содержимое этого документа относится к TYPO3
– GNU/GPL CMS/Framework доступной с www.typo3.com

Оглавление

Современная разработка шаблонов, Часть 1 (МТВ/1).....1

Введение	2
О чем это?.....	2
Основы	4
Введение.....	4
Начало работы – установка пакета "dummy".....	4
Создание структуры страницы.....	5
Пояснения к Дереву Страниц и Записям Шаблонов.....	7
Основные инструкции поля "Setup".....	10
Еще примеры с PAGE и cObjects.....	11

Повторное обращение к объекту PAGE.....	16
Часть 1: Интеграция HTML шаблона	20
Реализация CMS.....	20
Шаблон HTML.....	22
Основы ШАБЛОНА cObject.....	24
Расширение Анализатора Шаблона.....	26
Все вместе.....	29
Создание меню.....	33
Вставка страничного содержимого.....	36
Добавление совместимости с XHTML.....	41
Одновременная очистка.....	42
Некоторые предложения по дизайну HTML.....	44
Последнее замечание	47

Введение

О чем это?

Это расширение из чернового наброска руководства по созданию сайта с использованием CMS TYPO3 на базе шаблона HTML.

Для разработчиков начального уровня.

Сайт, созданию которого Вас будут обучать, будет выглядеть так:



Сайт состоит из динамической области для динамического меню(слева) и динамического страничного содержимого (справа) – остальное – статичный дизайн из шаблона HTML.

Цель

Цель этого руководства – вооружить Вас реальным умением создания сайтов под управлением TYPO3. Это позволит Вам быстро приобрести опыт работы с системой и даст Вам общее представление о подходах к разработке сайтов с помощью TYPO3.

Если Вам кажется, что это руководство слишком длинное и Вам нужно что-нибудь покороче, пожалуйста поищите другую CMS, потому что любое другое средство, такое же мощное и комплексное как TYPO3, нельзя описать короче. И даже это руководство – всего лишь снимки экрана с комментариями.

Несмотря на то, что TYPO3 бесплатна, все же она требует определенного времени для изучения разработчиком! Потребуется большое время для вживания и обучения – как и для *любой* другой коммерческой альтернативы такого же уровня. Итак, Вы предупреждены. Полет на самолете требует мастерства и, поэтому, времени на обучение. Надеемся, что это руководство станет для вас хорошим трамплином Вас...

Уровни мастерства

Это руководство разделено на четыре раздела:

Основы – введение для новичка по созданию сайтов на TYPO3, записей шаблонов, TypoScript и Объектов содержимого (Content Objects – cObjects). Любой, желающий использовать TYPO3 для разработки, должен ознакомиться с этой концепцией.

Часть 1: Интеграция шаблонов HTML – эта часть задумана специально для среднего уровня дизайнеров, с небольшим багажом технических знаний.

Часть 2: Создание селектора шаблонов – эта часть адресована для среднего уровня веб разработчиков с хорошим знанием PHP, SQL и программированием в целом.

Часть 3: Расширение встроенной схемы доступа – для продвинутых TYPO3/PHP-разработчиков.

Отметим: Часть 2 и Часть 3 находятся в [другом документе, в расширении "doc_tut_tmplselect2"](#)

Вы можете перейти к любой, интересной для Вас части. Тем не менее, если Вы начнете руководство с первой главы и дойдете до последней, то обнаружите, что главы последовательны, что даст Вам лучшее, связанное осознание. Отметим, что переход к следующему руководству может потребовать ряда экспериментов!

Расширение

Все файлы этого руководства содержатся в расширении TYPO3. Расширения, как правило, содержат программы и ресурсы, расширяющие возможности TYPO3. Отметим, что расширения этого руководства не взаимодействуют при установке с ядром TYPO3 – это лишь средство передачи файлов руководства на ваш сервер и интерактивного представления этого руководства на сайте typo3.org.

Таким образом, для продолжения следует сначала установить пакет dummy (смотри главу Основы) и затем импортировать расширение "doc_tut_tmplselect" из репозитория TER (TYPO3 Extension Repository) с помощью Менеджера Расширений (Extension Manager) и, тогда все файлы будут в Вас под рукой на вашем сервере.

Это руководство можно читать интерактивно или скачать для удобства SXW файл с сайта typo3.org.

Цена

Вам не надо платить за чтение этого документа. Тем не менее, от автора, Kasper Skårhøj, потребовалась целая неделя на подготовку, написание и редактирование. Эта работа не оплачивалась. Если Вы или ваша компания найдете, что это нужно и полезно для обслуживания мощных сайтов ваших клиентов, пожалуйста, рассмотрите возможность финансовой поддержки! Вы должны подставить мне плечо, чтобы я не пропал в одиночку. Иначе это руководство может стать последним...

ОСНОВЫ

Введение

Из этой главы новичок быстро вникнет в работу внешнего интерфейса TYPO3. Поймет что из себя представляют TypoScript шаблоны, объекты содержимого, HTML-шаблоны. Если Вы уже знакомы с этим, можете сразу перейти к следующей главе, "Интеграция HTML шаблона". Отметим, что Вам еще необходимо установить пакет dummy и создать ряд страниц. Об этом ниже.

Уровень мастерства

Для начинающих разработчиков TYPO3, независимо от опыта в других областях.

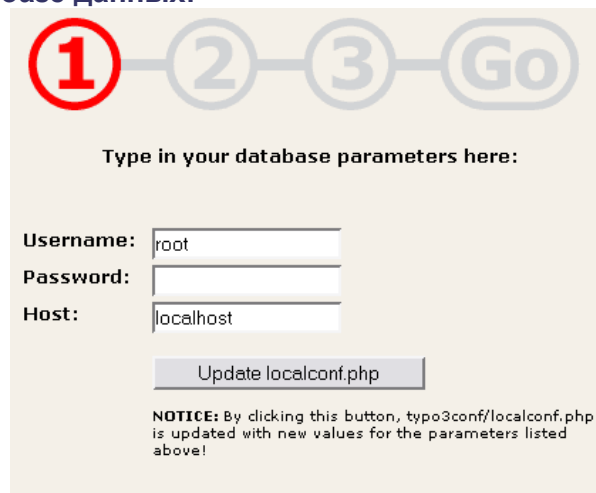
Некоторые понятия берутся из технологии вроде SQL, HTML, CSS и PHP. Содержание в основном технического плана, так как будет показана установка пакета dummy и объяснены основы работы движка шаблонов внешнего интерфейса TYPO3.

Начало работы – установка пакета "dummy"

Для того чтобы быстро начать работу, мы установим чистые движок и базу данных для TYPO3 и наилучшим выбором является использование пакета "dummy". "Пакет" состоит из основных исходных программ с локальными файлами сайта и, возможно, содержимого базы данных – вместе образующие полный сайт. В случае с пакетом dummy, это будет пустой сайт без содержимого. Но это крайне полезно для создания новых проектов.

Итак, возьмите пакет dummy и установите его на ваш сервер. После разархивирования пакета просто откройте в браузере файл index.php и Вы пройдете через следующие шаги.

Введите информацию о базе данных:



1 — 2 — 3 — Go

Type in your database parameters here:

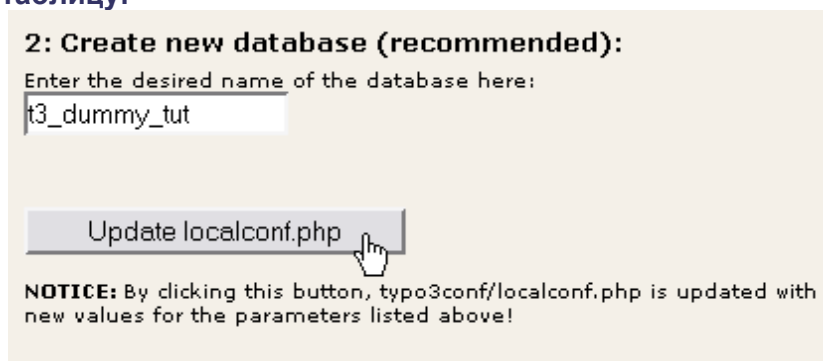
Username:

Password:

Host:

NOTICE: By clicking this button, typo3conf/localconf.php is updated with new values for the parameters listed above!

Создайте новую таблицу:

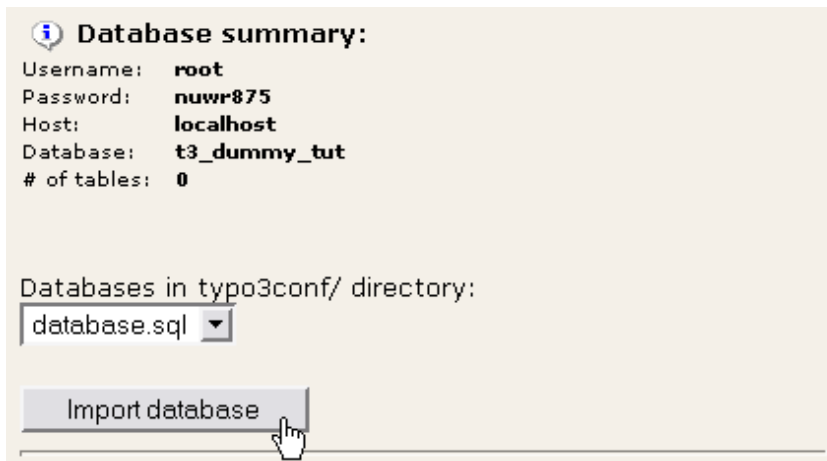


2: Create new database (recommended):

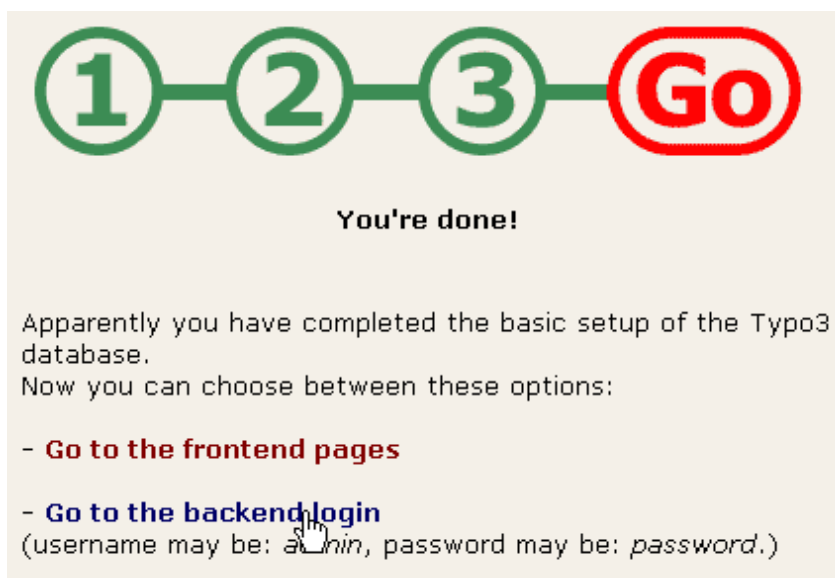
Enter the desired name of the database here:

NOTICE: By clicking this button, typo3conf/localconf.php is updated with new values for the parameters listed above!

Импортируйте базу данных dummy, "database.sql":



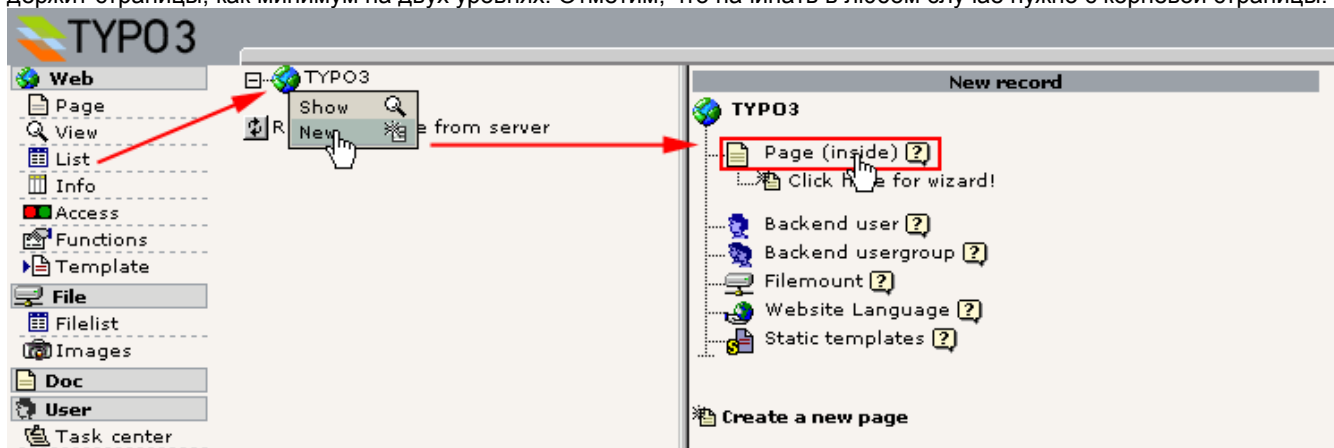
Переходите к внутреннему интерфейсу:



Для нормального выполнения этих шагов нужно сделать файл localconf.php доступным для записи (Вы об этом узнаете, если это не так!) и иметь под рукой имя и пароль пользователя базы данных.

Создание структуры страницы

Сейчас нужно создать структуру страниц, которую можно использовать в этом руководстве. Хорошая структура содержит страницы, как минимум на двух уровнях. Отметим, что начинать в любом случае нужно с корневой страницы:



Введите название страницы, сделайте ее видимой и сохраните:

Page NEW -

Hide page:

Type: Standard

Page title: Root page

Subtitle:

Можете продолжать таким же образом для создания структуры страниц. Отметим, что существует замечательный помощник для облегчения этого занятия. Просто перейдите к модулю “Функции” (Functions), выберите помощников (wizards) и из них “Создание множества страниц” (Create multiple pages), введите несколько названий страниц и нажмите кнопку “Создать страницы” (Create pages).

TYPO3

Web

- Page
- View
- List
- Info
- Access
- Functions
- Template
- File
- Filelist
- Images
- Doc
- User
- Task center
- Setup
- Tools
- User Admin
- Ext Manager
- DB check
- Configuration
- Install
- Log
- phpMyAdmin
- Help
- About modules

Advanced functions

Root page

Path: /Root page/

Select Wizard: Create multiple pages

CREATE MULTIPLE PAGES

Create new pages:

Page 1:	Homepage
Page 2:	Licensing
Page 3:	Terms & Conditions
Page 4:	Privacy Policy
Page 5:	Customer Login
Page 6:	
Page 7:	
Page 8:	
Page 9:	

Place new pages after the existing subpages
Hide new pages

Create pages! Clear fields

(Помощник “Создание множества страниц” требует установки расширения "wizard_cppages".)

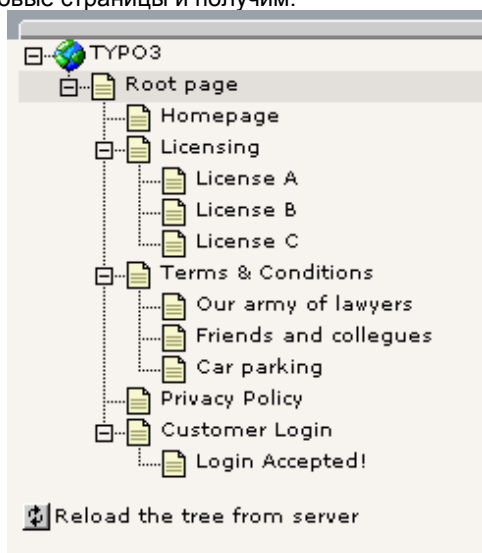
В результате получим прекрасное дерево страниц:

TYPO3

- Root page
 - Homepage
 - Licensing
 - Terms & Conditions
 - Privacy Policy
 - Customer Login

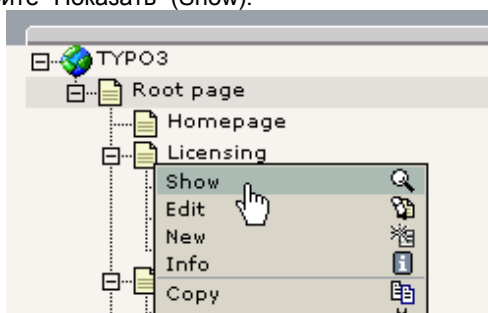
Reload the tree from server

Теперь, создадим дополнительные новые страницы и получим:

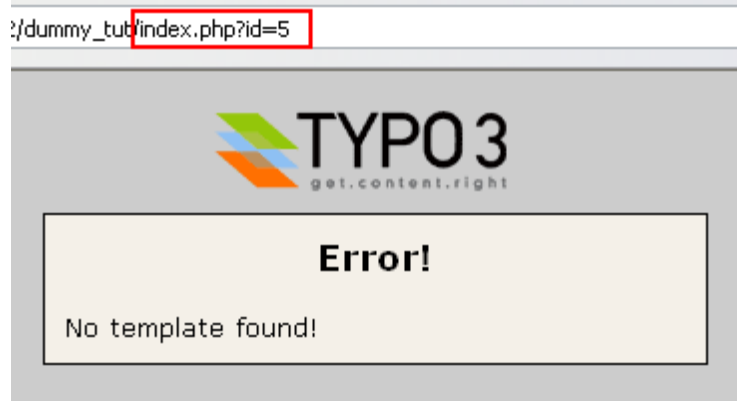


Пояснения к Дереву Страниц и Записям Шаблонов

Кликните по иконе страницы и выберите "Показать" (Show):



Результатом отображения страницы "Licensing" будет экран вроде следующего:



Страница "Licensing" имеет номер uid "5" и отображение содержимого этой страницы осуществляется посылкой параметра "id=5" программе index.php внешнего интерфейса.

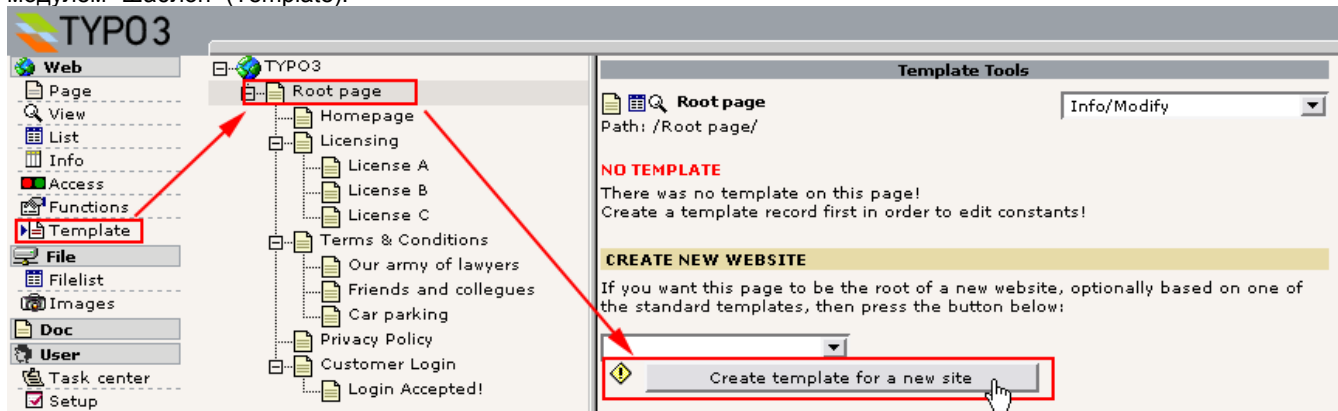
Теперь о сообщении "Не найден шаблон" (No template found). Это означает, что движок внешнего интерфейса не обнаружил *записи шаблона (template record)* в *корне* дерева страниц для страницы с номером id 5. Потребуется немного теоретических пояснений:

TYPO3 – многосайтовая CMS, что означает возможность создания любого количества различных сайтов в одной структуре дерева страниц. Каждый сайт в дереве страниц должен иметь *корневую страницу*, что отмечено наличием *записи шаблона* на ней. В этой записи шаблона должен быть установлен флаг "Корневая" (Rootlevel). Итак, как только запись шаблона с установленным флагом "Корневая" (Rootlevel) присоединена к странице, это означает, что "от этого места и далее начинается новый сайт в дереве страниц".

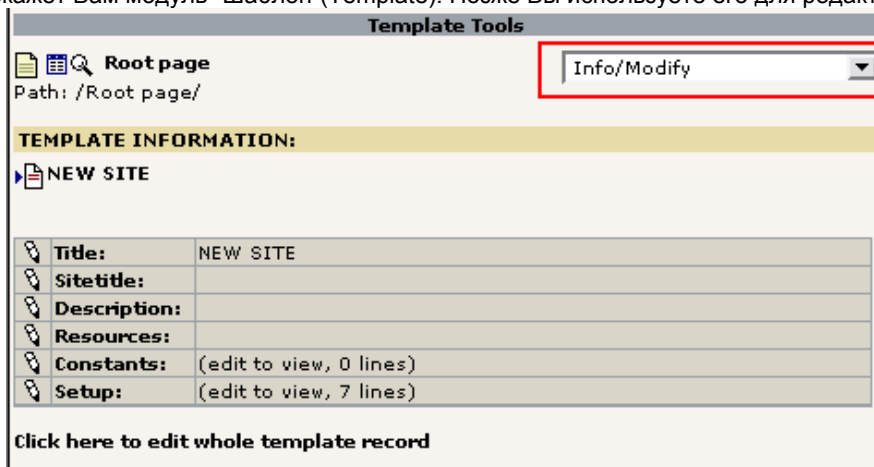
Корневая линия сродни слову "путь"(path). Например, Вы можете сказать, что страница "License A" в моем дереве страниц имеет путь "TYPO3 > Root page > Licensing > License A". И если Вы пошлете номер id страницы "License A" в программу index.php, то движок шаблонов внешнего интерфейса будет просматривать корневую линию от "License A", обратно к корню дерева страниц для определения записи шаблона, которая скажет, каким образом будет обработана эта ветка дерева.

Создание записи шаблона

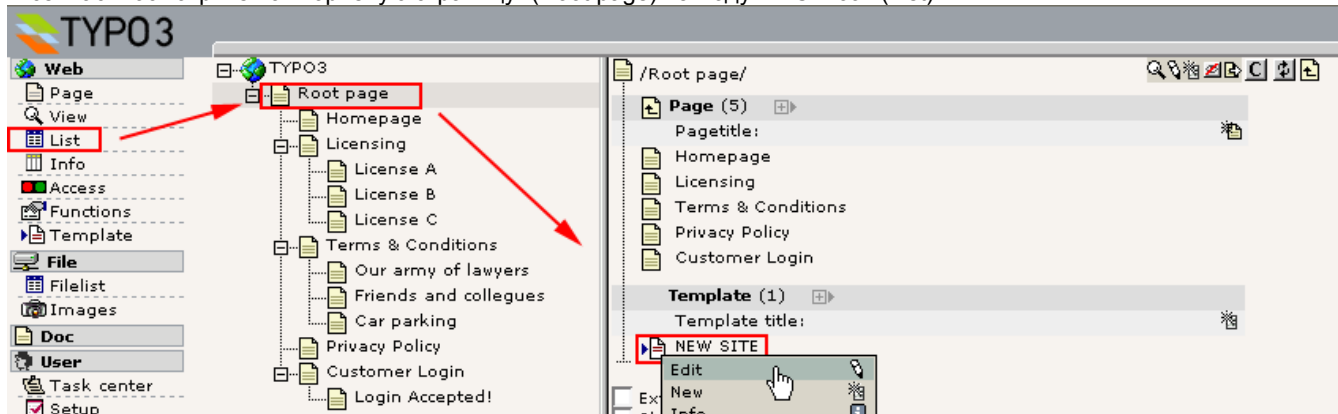
Итак, теперь мы создадим запись шаблона на странице “Корневая”(Root page). Самый простой путь, воспользоваться модулем “Шаблон” (Template):



Следующее окно покажет Вам модуль “Шаблон”(Template). Позже Вы используете его для редактирования шаблона:



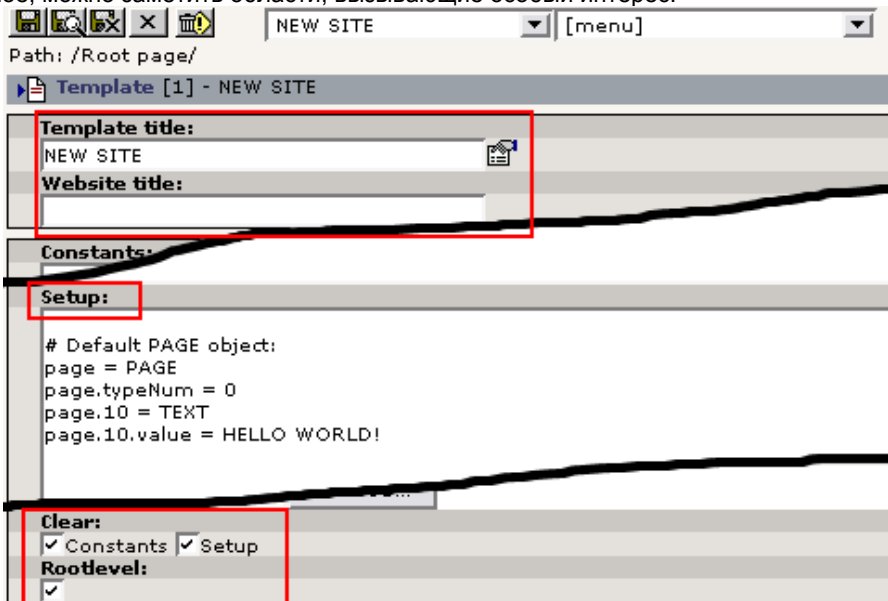
А сейчас посмотрите на “Корневую страницу” (Root page) из модуля “Список”(List):



Как видно, на странице “Корневая”(Root page) появилась запись “Шаблон”(Template).

Этот шаблон будет содержать *основные инструкции для движка внешнего интерфейса* о способах обработки сайта, начиная с этого места, особенности, настройки и т.д. Запись шаблона *всегда* необходима для любого нового сайта в TYPO3 (используя типовой движок внешнего интерфейса). Отметим, что Вы можете решить самостоятельно, какая часть дизайна сайта будет напрямую управляться записью шаблона; Вы можете запрограммировать весь сайт из записи шаблона. Также, Вы можете заставить запись шаблона сразу же вызвать функцию PHP, которую Вы создали для обработки и вывода всего страничного содержимого (на основе XML, XSLT или других баз данных – все что Вам угодно). Вы также можете предпочесть золотую середину обоих подходов. Вот этим мы и займемся в этом руководстве.

Глядя на содержимое, можно заметить области, вызывающие особый интерес:



“Название шаблона”(Template title) это просто название вашей записи шаблона – никуда не выводится.

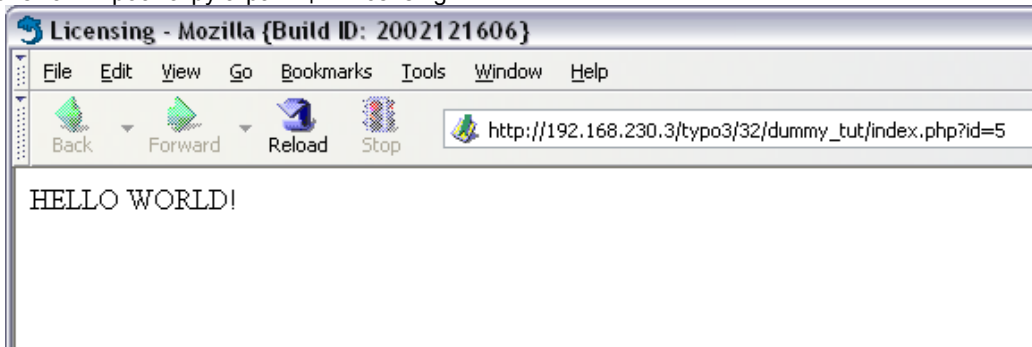
“Название сайта”(Website title) – префикс названия страницы, который находится между тегами <title> каждой страницы.

Поле "Setup" – здесь Вы введете *инструкции* на которые будет реагировать движок внешнего интерфейса. Эти инструкции содержатся в иерархической информационной структуре, определяемой синтаксисом TypoScript. Так как инструкции поля "Setup" определяются синтаксисом TypoScript, Вам часто придется сталкиваться со ссылками на записи шаблонов как на "TypoScript templates".

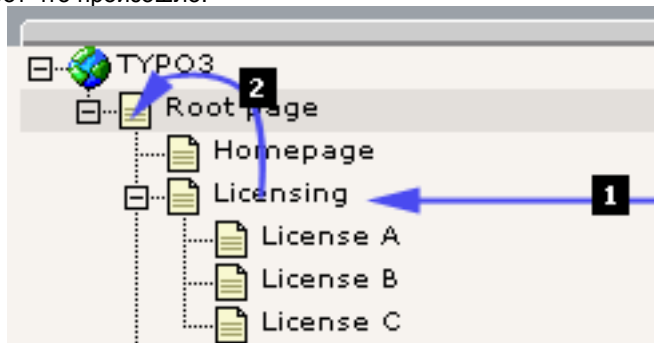
Флаги “Очистить константы” (Clear constants) и “Очистить setup” означают, что конфигурационный код TypoScript из записей шаблонов выше по дереву (ближе к корню) не наследуется, если и существует. Эти флаги всегда нужно устанавливать для основных (“Корневых” – “Rootlevel”) записей шаблонов.

Флаг “Корневой”(Rootlevel) просто означает: “Этот шаблон отмечает начало нового сайта”.

А теперь, вернемся к просмотру страницы "Licensing":



И вот, всплыло что-то иное. Вот что произошло:

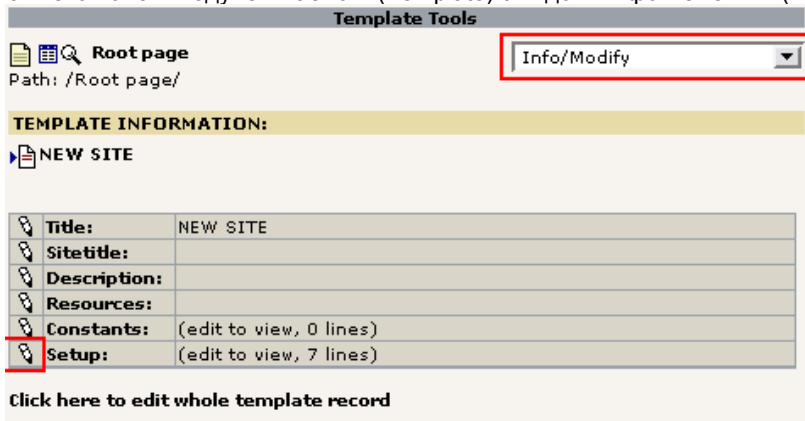


- Запрос страницы с номером uid "5" (1)
- Выбор первой записи шаблона на этой странице (pid=5); ничего не найдено!
- Так как не обнаружено записи шаблона на странице с номером uid "5", проверяется предыдущая страница в корневой линии.... (2)
- Выбирается первая запись шаблона на этой странице, "Корневая"(Root page), (pid=1); запись шаблона найдена!
- Так как запись шаблона была *обнаружена* на странице с номером uid "1", проверяется установка флага "Корневая" (Rootlevel)...
- Да, флаг "Корневая" (Rootlevel) установлен. Здорово, это начало сайта. Итак...
 - Разбор конфигурации TypoScript в поле Setup.
 - Начало выполнения инструкций из поля Setup.

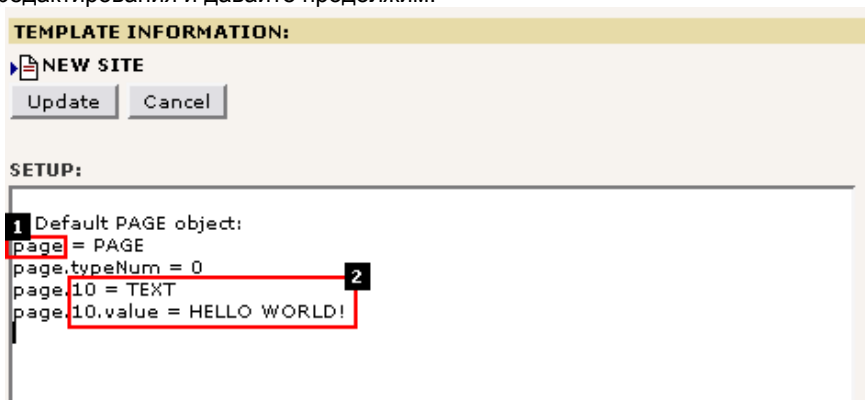
Основные инструкции поля "Setup"

Когда запись шаблона на месте, основной интерес представляет поле "Setup" этого шаблона. В этом поле Вы вводите набор инструкций, определяющих, что произойдет при отображении страницы во внешнем интерфейсе.

Легче всего редактировать это поле в модуле "Шаблон" (Template) с видом Инфо/Изменить (Info/Modify):



Кликните по иконке редактирования и давайте продолжим:



Приведем краткое объяснение:

Объект PAGE (1)

Сначала мы определяем новый объект "Toplevel Object" (TLO) называемый "page". Тип объекта – "PAGE" (Вы также можете определить FRAMESET, смотрите TSref). Вместо "page" можно использовать другие строки, кроме зарезервированного списка объектов TLO, смотрите [TSref about Toplevel Objects](#). В качестве свойства объекта "page" мы определим обязательное свойство, "typeNum", и зададим его значение равным 0 (нуль).

Все это означает, что этот Объект Верхнего Уровня (Toplevel Object) "page" теперь будет *диспетчером по-умолчанию* любых запросов, посылаемых к странице внутри ветви дерева страниц, для которой наша запись шаблона помечена как начало сайта.

А теперь, мы хотели бы сообщить объекту "page", что нужно сделать. Обычно, нам необходимо поискать [возможные свойства для объектов PAGE в TSref](#). Некоторые свойства являются meta свойствами, вроде свойств "config" или "includeLibs". Другие нужны для определения того, какого вида HTML код будет генерировать этот объект PAGE.

Наиболее важной установкой свойств для объектов PAGE является *числовой массив объектов содержимого (content objects – cObjects)*. Каждый из этих элементов содержимого может формировать строку HTML-содержимого, которые все вместе образуют содержимое между тегами <body> страницы.

Объект Контента (Content Object) (2)

В приведенном выше примере, имеется самый простой элемент содержимого – TEXT, определенный для индекса "10" числового массива объектов содержимого для объекта "page" типа PAGE. Этот cObject имеет свойство "value"(значение) установленное в "HELLO WORLD!". В результате эта строка берется из объекта cObject и становится содержимым страницы между тегами <body>:

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

Можно познакомиться с [набором свойств для TEXT cObject здесь](#).

ТуроScript не является языком программирования

Перед тем как двинуться дальше; пожалуйста отметьте, что ТуроScript не является процедурным языком программирования. В этом легко убедиться, особенно если у Вас есть опыт использования других языков программирования и знание того, как они работают, вероятно Вы сильно разочарованы. В действительности, ТуроScript просто *настраивает* систему, чтобы она работала в определенном ключе.

Для того, чтобы объяснить что же такое ТуроScript и указать на различие между ним и его использованием в ТУРОЗ, можете заглянуть в [этот документ объясняющий основы](#).

Еще примеры с PAGE и cObjects

В этом разделе просто приводится еще несколько примеров для PAGE и cObjects, чтобы Вы полностью смогли усвоить концепцию. Понимание этого позволит понять и "Шаблоны ТуроScript" более сложной структуры.

А если два cObjects?

Что произойдет, если Вы введете другой элемент содержимого:

```
# Объект PAGE по-умолчанию:
page = PAGE
page.typeNum = 0

# Первый объект содержимого:
page.10 = TEXT
page.10.value = HELLO WORLD!

# Второй объект содержимого:
page.20 = TEXT
page.20.value = HELLO UNIVERSE!
```

(BTW, строки начинающиеся с "#" или "/" – комментарии.)

Вывод:

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

Итак, как и ожидалось, несколько объектов содержимого просто суммируются!

А как изменить порядок?

Что если порядок определения изменен?

```
# Объект PAGE по-умолчанию:
page = PAGE
page.typeNum = 0

# Второй объект содержимого:
page.20 = TEXT
page.20.value = HELLO UNIVERSE!

# Первый объект содержимого:
page.10 = TEXT
page.10.value = HELLO WORLD!
```

Вывод:

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

Тоже самое – порядок отображения соответствует индексу в массиве.

Как сделать динамическое содержимое?

Что если мы захотим поместить в TEXT сObject заголовок страницы?

Тогда Вам придется [поискать в TSref есть ли у TEXT сObject](#) свойства, позволяющие сделать это. Как нам кажется, это возможно. Кроме свойства "значение" (value), на том же уровне объект TEXT также имеет свойства ["stdWrap"](#) ([кликните здесь](#)) которые позволяют выбирать и обрабатывать данные динамически.

Так как отображение сObject TEXT выполняется на основном уровне Объекта PAGE, *текущая запись* станет записью страницы. Таким образом, ссылка на имя поля в stdWrap со свойством "field" приведет к выборке содержимого из текущей записи страницы. Здесь поле названное "title" будет содержать название страницы (это, конечно, требует знания о том какие поля доступны в таблице страниц!):

```
# Объект PAGE по-умолчанию:
page = PAGE
page.typeNum = 0

# Объект содержимого выводящий название текущей страницы:
page.10 = TEXT
page.10.field = title
```

Вывод:

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

Так как текущей была страница "Licensing", этот заголовок и будет выведен!

Свойства stdWrap

Теперь развлечемся со свойствами "stdWrap", мы можем захватывать значения, обрабатывать их несколькими способами и т.п.

```
# Объект PAGE по-умолчанию:
page = PAGE
page.typeNum = 0

# Объект содержимого выводящий название текущей страницы:
page.10 = TEXT
page.10 {
  field = title
  crop = 8 | ...
  case = upper
  wrap = Это обрезанный заголовок страницы: <b> | </b>
}
```

(Отметим, каким образом я изменил форматирование в TurboScript – свойства были заключены в фигурные скобки. Такой синтаксис TurboScript делает легким включение множества свойств на одном уровне, но конечный результат не зависит от порядка всех этих свойств для объекта "page.10".)

Вывод:

```
<body bgcolor="#FFFFFF">
Это обрезанный заголовок страницы: <b>LICENSIN...</b>
</body>
```

Обратим внимание, как был организован захват строки (контейнер) для названия страницы, которое позже стало прописным, обрезано до 8 символов, и добавлено "...!"

Мы можем добиться такого же эффекта для вывода с помощью другого объекта сObject называемого "HTML", различие лишь в том, что все свойства stdWrap применяются для "value" (значения) объекта HTML сObject, а не для самого объекта сObject:

```
# Объект PAGE по-умолчанию:
page = PAGE
page.typeNum = 0

# Объект содержимого выводящий название текущей страницы:
page.10 = HTML
page.10.value {
    field = title
    crop = 8 | ...
    case = upper
    wrap = Это обрезанный заголовок страницы: <b> | </b>
}
```

Вообще, это вопрос предпочтения и стиля, какой из объектов использовать TEXT или HTML сObject.

Содержимое из программ PHP

Если хотите, Вы легко сможете вставить содержимое из программы на PHP. Это даже *рекомендуемая* практика, при необходимости вывода с условиями, похожими на приведенные выше, на основном уровне, так как объекты сObjects и их свойства не требуют "процедурного программирования", запомните, что это просто запрограммированные объекты, отвечающие набору установленных для них свойств, ничего более.

Итак, Вам нужен объект USER сObject.

Сначала давайте создадим файл PHP в `fileadmin/userfunctions.php`:

```
<?php

class user_functions {

    /**
     * Умножает номер ID текущей страницы на значение $conf["factor"]
     */
    function multiplyTest($content,$conf) {
        $currentPageUid = $GLOBALS['TSFE']->id;
        $factor = intval($conf['factor']);

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

Затем, давайте настроим [сObject типа USER](#) на вызов этой функции с единственным параметром "factor":

```
# Объект PAGE по-умолчанию:
page = PAGE
page.typeNum = 0
page.includeLibs.some_random_id_string = fileadmin/userfunctions.php
page.config.admPanel = 1

# Объект содержимого выводящий название текущей страницы:
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
```

И вот что будет выведено на экран:



Как Вы видите, мы определили несколько "meta"-свойств прямо не относящихся к выводу содержимого:

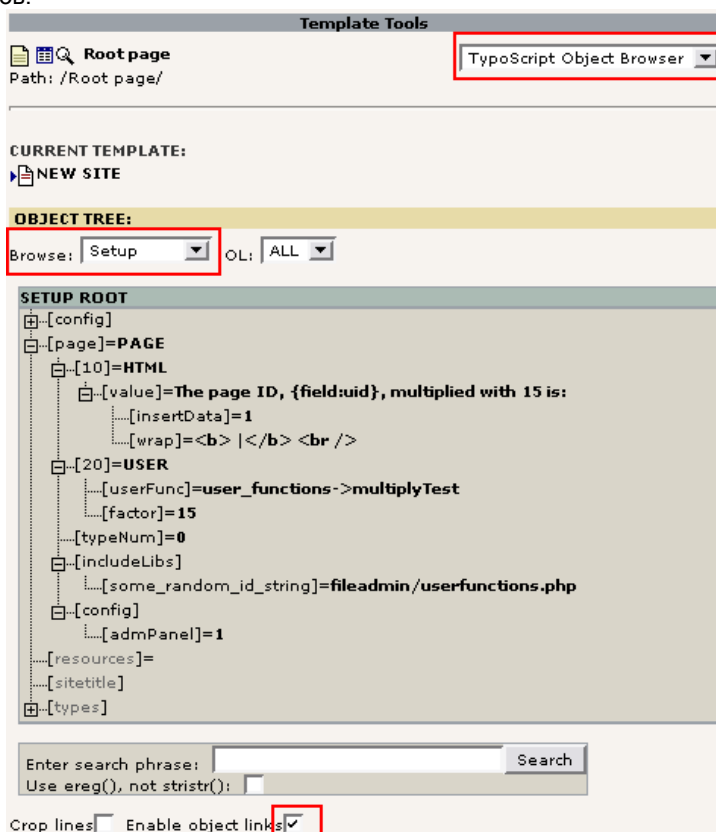
```
page.includeLibs.some_random_id_string = fileadmin/userfunctions.php  
page.config.admPanel = 1
```

"includeLibs" – свойство, которое позволяющее подключить список файлов PHP (классы и библиотеки функций!) перед отображением страницы. "config" разрешает использование всего множества параметров настроек, относящихся к внешнему виду объекта PAGE. В нашем случае добавлена административная панель (Admin Panel) внизу страницы.

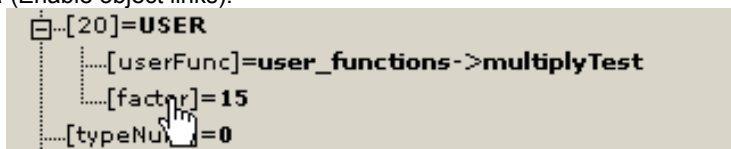
Вам на самом деле необходимо затратить несколько минут для изучения этого примера. Заметьте, как свойство "factor" объекта USER стало доступно для функции PHP. Как функция PHP получает доступ к ID номеру страницы. И как была подключена "административная панель"(admin panel). Вы сможете многое почерпнуть, немного поигравшись с этим примером. Дайте полет вашей фантазии, хоть ненадолго, это создаст хорошее настроение от возможностей управления.

Броузер Объектов (Object Browser)

Броузер объектов в модуле Шаблона – обязательный инструмент проверки иерархической информационной структуры Ваших записей шаблонов:



Как видим, значения TypoScript организованы в дерево. Это позволяет Вам проверить, на своем ли месте находится каждое установленное свойство, и даже возможно отредактировать отдельные значения, при включенном флаге "Разрешить связи объекта"(Enable object links):



Вернувшись к "Инфо/Изменение"(Info/Modify)", Вы увидите, что поле Setup шаблона изменилось:

```
SETUP:
# 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

page.20.factor = 30
```

Броузер объектов неудобен для поиска и изменения строки "page.20.factor = 15" – он просто добавляет в конце новую строку, переопределяющую любое установленное ранее значение. Вы можете убрать его вручную. Но Броузер объектов выполняет наилучшую установку этих значений, поскольку всегда используется правильный путь к объектам.

Попробуем условия?

Конечно, мы можем, использовать этот пример для демонстрации ["условий"\(conditions\)](#). Одно из основных условий – проверка браузера:

```
# Объект PAGE по-умолчанию:
page = PAGE
page.typeNum = 0
page.includeLibs.some_random_id_string = fileadmin/userfunctions.php
page.config.admPanel = 1

# Объект содержимого выводящий название текущей страницы:
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 />

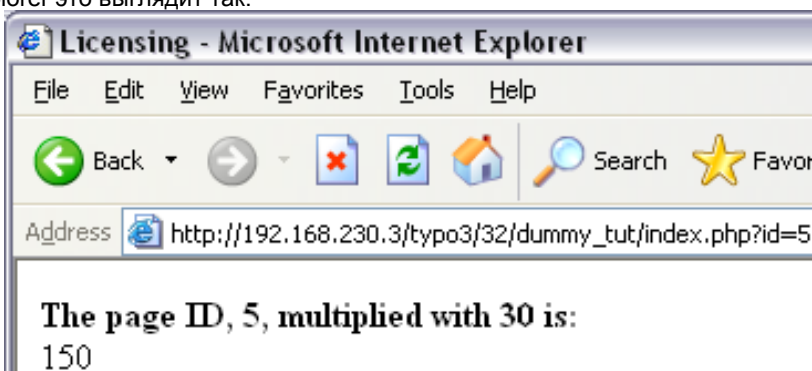
page.20 = USER
page.20.userFunc = user_functions->multiplyTest
page.20.factor = 15

[browser = msie]

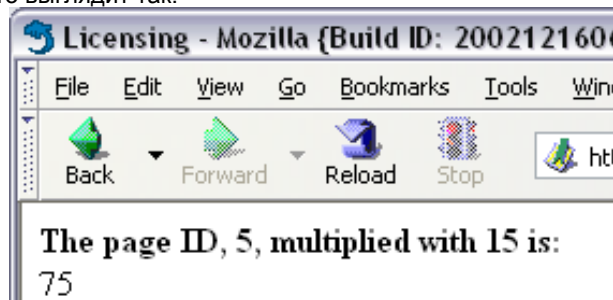
page.20.factor = 30
page.10.value = The page ID, {field:uid}, multiplied with 30 is:

[global]
```

В Microsoft Internet Explorer это выглядит так:



Во всех остальных браузерах это выглядит так:



Повторное обращение к объекту PAGE

Хорошо, здесь не будет полного обязательного курса по созданию шаблонов исключительно с помощью объектов сObjects – в этом руководстве мы будем в основном использовать объекты TEMPLATE, USER и HMENU сObjects и комбинировать их с предопределенным кодом TypoScript из статических шаблонов для реализации наших целей. Самым полным по [объектам и свойствам является справочник TSref](#), а уж если Вам потребуются примеры использования каждого типа объекта, смотрите [TypoScript by Example](#).

Тем не менее, я хотел бы завершить с парой вещей:

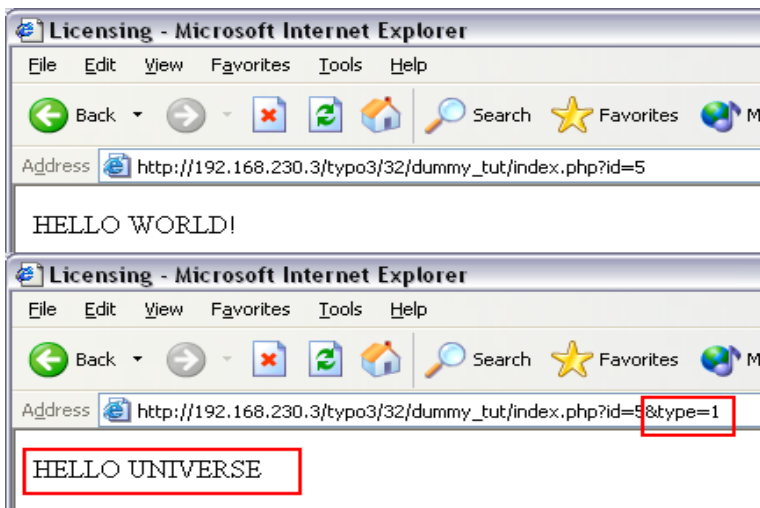
Важность "&type="

В примере, приведенном ниже, определено несколько объектов PAGE. Объект TLO "another_page" также определен как объект PAGE, но со свойством "typeNum" равным "1". Это означает, что объект "page" типа PAGE является диспетчером запросов к странице по умолчанию, в то время как объект "another_page" типа PAGE становится диспетчером запросов к странице, для которой кроме параметра id, установлен параметр "&type=1":

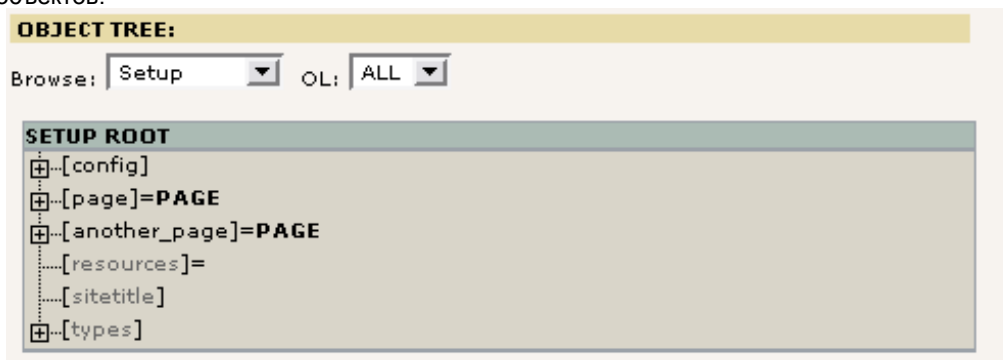
```
page = PAGE
page.typeNum = 0
page.10 = TEXT
page.10.value = HELLO WORLD!

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

Вот что получится:



А вот дерево объектов:



Копирование объектов

Для удобства, лучше разделить конфигурационный код TYPOScript вашего шаблона на связанные единицы и затем собрать их в конце шаблона копируя в нужное место. Возможно даже разделять элементы в иерархии подключаемых записей шаблона или статических шаблонов, с целью повторного использования TYPOScript. Вы изучите это позднее.

Основной особенностью TYPOScript, позволяющей это делать, является синтаксис для копирования одной ветви объекта в другую. Разберем этот пример:

```
# Создать временную версию первого объекта cObject:  
temp.world = TEXT  
temp.world.value = HELLO WORLD!
```

```
# Создать временную версию второго объекта cObject:  
temp.universe = TEXT  
temp.universe.value = HELLO UNIVERSE!
```

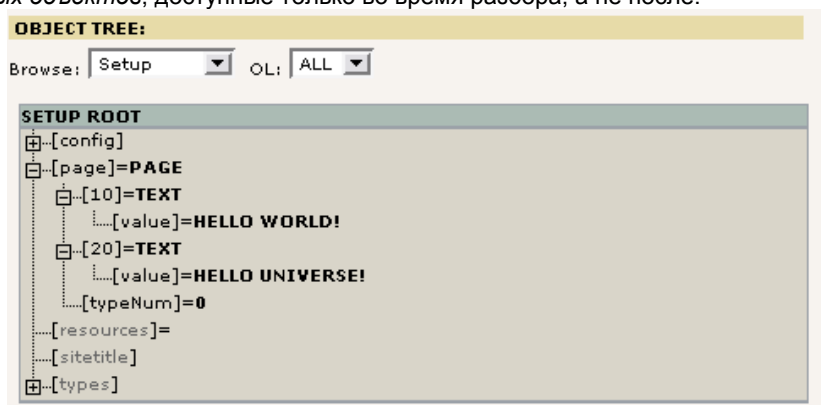
```
# Объект PAGE по-умолчанию:  
page = PAGE  
page.typeNum = 0
```

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

Результатом будет – конечно – это:

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

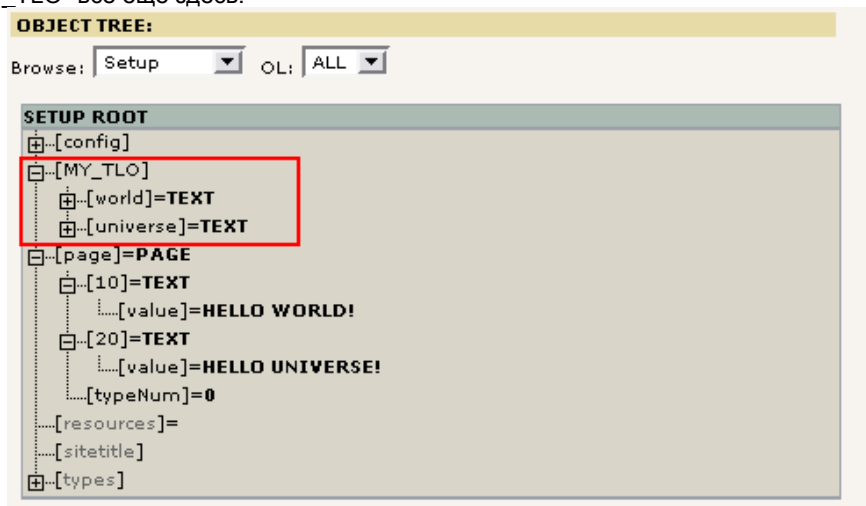
Если посмотреть в окно браузера, то видно, что объекты из "temp. ..." были скопированы в части "page.10" и "page.20". Тем не менее, мы ожидали увидеть TLO "temp." в дереве объектов, но не видим. Причина в том, что объекты TLO с именами "temp." и "styles." стали снова *неопределенными* после разбора TYPOScript, потому что они резервируются как *области временных объектов*, доступные только во время разбора, а не после.



Если мы хотим, чтобы объекты оставались в дереве, можно использовать другой объект TLO:

```
# Создать временную версию первого объекта cObject:  
MY_TLO.world = TEXT  
MY_TLO.world.value = HELLO WORLD!  
  
# Создать временную версию второго объекта cObject:  
MY_TLO.universe = TEXT  
MY_TLO.universe.value = HELLO UNIVERSE!  
  
# Объект PAGE по-умолчанию:  
page = PAGE  
page.typeNum = 0  
  
page.10 < MY_TLO.world  
page.20 < MY_TLO.universe
```

... и объект TLO "MY_TLO" все еще здесь:

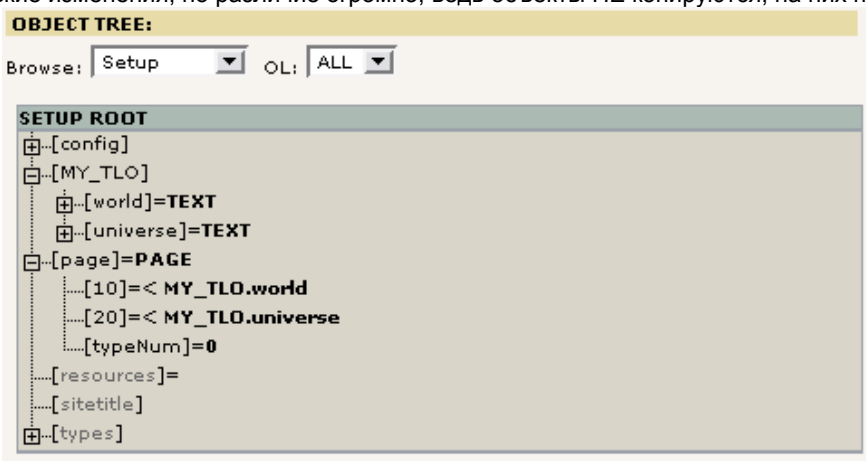


Ссылки – не копии

Это также означает, что можно создать ссылку на объекты MY_TLO.*, так как ссылка на объекты cObjects предполагает их существование в структуре в том числе и после разбора (в противоположность "temp" и "styles" TLO). Поэтому сценарий можно переписать вот так:

```
# Создать временную версию первого объекта cObject:  
MY_TLO.world = TEXT  
MY_TLO.world.value = HELLO WORLD!  
  
# Создать временную версию второго объекта cObject:  
MY_TLO.universe = TEXT  
MY_TLO.universe.value = HELLO UNIVERSE!  
  
# Объект PAGE по-умолчанию:  
page = PAGE  
page.typeNum = 0  
  
page.10 =< MY_TLO.world  
page.20 =< MY_TLO.universe
```

Казалось бы маленькие изменения, но различие огромно, ведь объекты НЕ копируются, на них просто ссылаются:



Существует несколько практических следствий и препятствий этому методу. Тем не менее, очевидно что ссылки на объекты `sObject`, определенные в дереве объектов, можно использовать внутри структуры много раз не расходуя память на дубликаты – что тоже непростая задачей, если Вы захотите переопределить свойства всех дубликатов в дереве. Пример этой проблемы можно найти в конце Части 2 этого руководства.

Отметим: Ссылки, подобные этим, не являются частью синтаксиса `TypoScript`, как таковые. Это внутренняя особенность `sObjects` и, поэтому, ссылки можно использовать *только* для объектов `sObjects`, а *не* для любых типов объектов или свойств (кроме упомянутых).

Очистка буфера(cache)

Наконец, нужно побеспокоиться о том, что при внесении изменений в запись шаблона в модуле Шаблон (Template) будут очищены все кэши (Cache-all). Это надо делать потому, что при разборе в ТУР03, окончательная структура шаблона `TypoScript` сохраняется в кэш, таким образом, она снова может быть загружена при очередной выдаче страницы. Тем не менее, это означает, что изменения внесенные *прямо* в модуле Список (List) например, *не* очищают кэш, и тогда Вы должны это сделать *вручную* в меню слева.

Обычно, перед тем как рассматривать странное поведение как ошибку, нужно убедиться, что это случилось не из-за шаблона или файлов расширения, оставшихся в кеше!

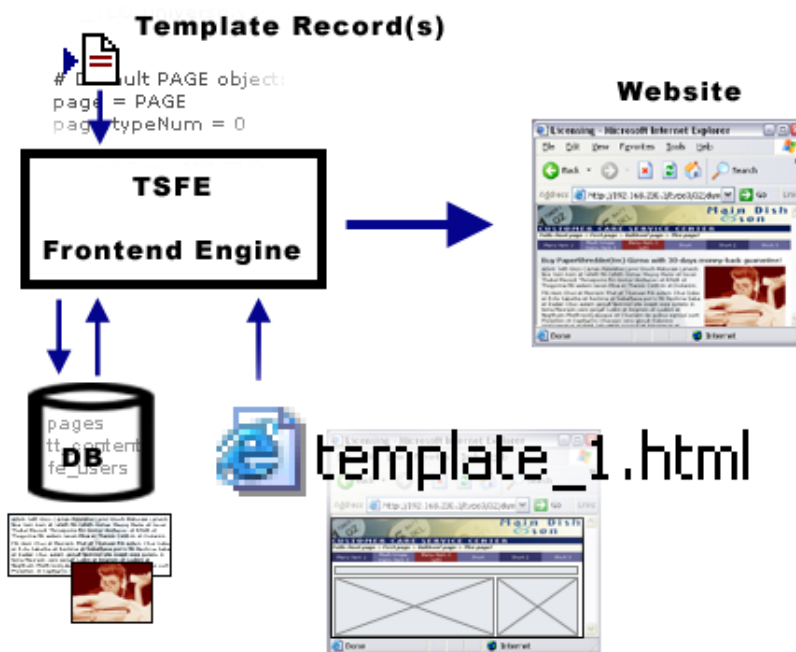
Часть 1: Интеграция HTML шаблона

Реализация CMS

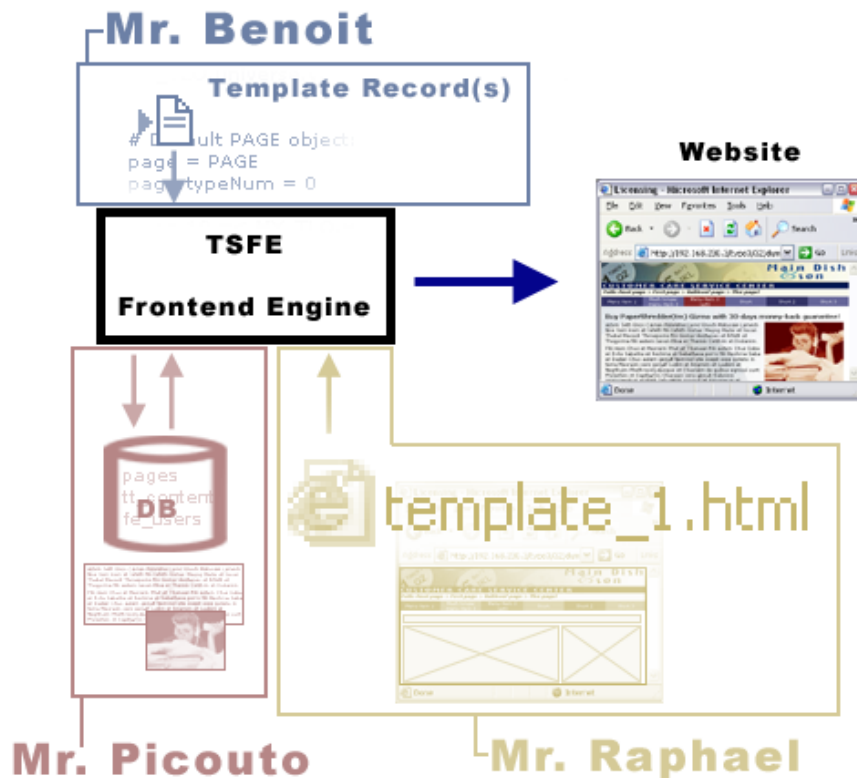
Создание сайтов с использованием CMS требует включения большего числа компонентов, чем для сайтов на чистом, статичном HTML. Главная идея CMS – хранение содержимого отдельно от его представления. Некоторые CMS сохраняют содержимое в файлах XML, другие используют реляционные базы данных. Различие, чисто технически, имеет свои за и против, но идея та же: "содержимое отдельно от его представления".

Когда TYPO3 формирует веб-страницы, движок внешнего интерфейса комбинирует неформатированное содержимое из источника данных (базы данных) с шаблоном HTML, определяющим все форматирование. В этом процессе запись шаблона является управляющим элементом, говорящим движку внешнего интерфейса о том, как выполнить это комбинирование.

На этом изображении Вы можете видеть каким образом запись шаблона управляет, "программирует" работу движка внешнего интерфейса, который последовательно находит содержимое в базе данных, читает шаблон, вставляет содержимое в назначенные области шаблона и выводит отличную веб-страницу!



В большинстве веб-агентств созданием сайтов занимается команда. В этой команде мы найдем графического дизайнера, разработчика и создателя содержимого (по крайней мере простого текста). Каждый член команды является профессионалом в своей области и, таким образом, создание сайта суммирует результаты работы в различных областях:



На иллюстрации выше, различные компоненты CMS управления сайтом разделены для каждого члена команды:

- **Рафаэль** – художник. Рафаэль – отличный графический дизайнер, он владеет Photoshop, он знает свой Dreamweaver, CSS таблицы стилей, HTML и т.д.. Рафаэль фокусируется с клипами. Но Рафаэль не компетентен в PHP, TurboScript, SQL и другими техническими дисциплинами. Итак, Рафаэль делает для нас шаблоны HTML!
- **Бенуа** – разработчик. Он обожает биты и байты, он без ума от директив компилятора, регулярных выражений, логики, PHP, SQL – и, вскоре, он полюбит и TurboScript. Тем не менее, Бенуа мечтает о прекрасной половине, носит голубые джинсы, как и каждый программист, поскольку он немного хиппи, а типографику, сочетания цвета и практичность нельзя найти в его гороскопе. Поэтому, Бенуа привлечен к конфигурированию Записей Шаблонов в TurboScript.
- **Писака** – занимается содержимым. Он маркетолог и изумлен работой этих двоих, Рафаэля и Бенуа, которые творят чудеса у него на глазах, перенося идеи с бумаги на экран. Писака не является ни программистом, ни дизайнером, но он очень коммуникабелен. И во внутреннем интерфейсе TYPO3 он может создавать содержимое, без всякого предварительного знакомства с текстовым редактором.

Итак, у нас есть три персонажа с уникальными способностями, из этого можно заключить, что создание сайта на базе CMS – *не прогулка в парке*, даже если Вы мастер в трех вышеупомянутых областях: дизайн, техника и маркетинг. Это обычное дело для веб команды, но очень необычно для окружающих. Итак, Вы заранее предупреждены. (Если Вы менее квалифицированы, чем нужно, Вам потребуется либо много учиться, либо воспользоваться Стандартными Шаблонами, с готовым дизайном, которые можно использовать с минимальными вариациями. Что не рассматривается в этом руководстве.)

В этом руководстве, я покажу Вам каким образом Рафаэль, Бенуа и Писака объединяются с целью создания современных сайтов, когда каждый член команды может свободно бродить по своей территории, используя только любимые ими средства разработки, обеспечив простоту верстки сайтов и любой дизайна о котором только можно мечтать. В TYPO3.

Уровень мастерства

Среднему веб-дизайнеру HTML/CSS профиля, не требуется веб-программирование.

Чтобы завершить этот раздел руководства, Вы должны хорошо знать HTML и CSS. Для продолжения, у Вас должна быть рабочая база данных TYPO3 с деревом страниц, созданным в предыдущих разделах этого руководства. Наконец, Вам может потребоваться усвоение программной концепции, для полного понимания всего. Но не беспокойтесь – Вы точно знаете, что Вам нужно сделать, наберитесь терпения и найдите время разобраться с примерами.

Шаблон HTML

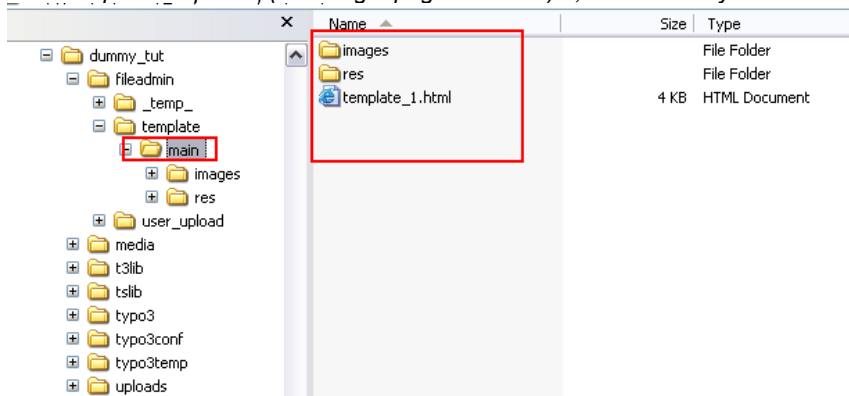
У веб команды появился новый заказчик – Main Dish & Son, и Рафаэль, художник команды, сделал шаблон сайта как обычный файл HTML:



Этот HTML файл находится в каталог "fileadmin/template/main/", относительно инсталляции TYPO3 (пакет dummy).

Следуя этому руководству, скопируйте содержимое каталога "part1/" из расширения к этому руководству в каталог "fileadmin/template/main/". Если Вы не импортировали расширение к руководству "doc_tut_templateselect" с сайта TER, сделайте это сейчас!

Далее, если Вы пропустили предыдущую главу о TypoScript, удостоверьтесь, что создано дерево страниц, как описано в главе "Создание дерева страниц"(Creating a page structure) и, возможно пустая запись шаблона.



А теперь, вернемся к работе Рафаэля; файл шаблона HTML – на самом деле, обычная страница HTML. Но в TYPO3 этот файл импортируется как шаблон с единственной целью – сделать *определенные части* динамическими. Имеется в виду меню слева и, также, секция с пока пустующим содержимым в центре/справа.

Рассматривая исходный код шаблона HTML, обнаруживаем документ, совместимый с простым XHTML, ссылающийся на таблицу стилей и использующий одну таблицу для позиционирования различных элементов на странице:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet href="#internalStyle" type="text/css"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Base template, header, menu, content and footer.</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<style type="text/css" id="internalStyle">
/**/
BODY { margin: 0 0 0 0; background-color: white; }
/*]]&gt;*/
&lt;/style&gt;
&lt;link href="res/stylesheet.css" rel="stylesheet" type="text/css" /&gt;
&lt;/head&gt;
&lt;body&gt;
&lt;table border="0" cellpadding="0" cellspacing="0"&gt;
&lt;tr&gt;
&lt;td colspan="2" id="header_1"&gt;&lt;img src="images/headerimage.jpg" width="600" height="100" alt="" border="0" /&gt;&lt;/td&gt;
&lt;/tr&gt;
&lt;tr&gt;
&lt;td id="menu_1"&gt;
&lt;div class="menu1-level1-no"&gt;&lt;a href="#"&gt;Menu item 1&lt;/a&gt;&lt;/div&gt;
&lt;div class="menu1-level1-no"&gt;&lt;a href="#"&gt;Menu item 2&lt;/a&gt;&lt;/div&gt;
&lt;div class="menu1-level1-act"&gt;&lt;a href="#"&gt;Menu item 3 (act)&lt;/a&gt;&lt;/div&gt;
&lt;div class="menu1-level2-no"&gt;&lt;a href="#"&gt;Level 2 item&lt;/a&gt;&lt;/div&gt;
&lt;div class="menu1-level2-no"&gt;&lt;a href="#"&gt;Level 2 item&lt;/a&gt;&lt;/div&gt;
&lt;div class="menu1-level2-act"&gt;&lt;a href="#"&gt;Level 2 item (act)&lt;/a&gt;&lt;/div&gt;
&lt;div class="menu1-level1-no"&gt;&lt;a href="#"&gt;Menu item 2&lt;/a&gt;&lt;/div&gt;
&lt;/td&gt;
&lt;td id="content"&gt;
&lt;h1&gt;Buy PaperShredder(tm) Gizmo with 30-days money-back guarantee!&lt;/h1&gt;
&lt;img src="images/paperwork.jpg" width="192" height="156" border="0" alt="" align="right"/&gt;
&lt;p class="bodytext"&gt;Adam Seth Enos Cainan Malelehel Jared Enoch Matusale Lamech Noe Sem Ham et Iafeth filii
&lt;p class="bodytext"&gt;Filii Ham Chus et Mesraim Phut et Chanaan filii autem Chus Saba et Evila Sabatha et Rechm
&lt;br /&gt;
&lt;/td&gt;
&lt;/tr&gt;
&lt;tr&gt;
&lt;td colspan="2" id="footer"&gt;
&lt;p&gt;Main Dish &amp;amp; Son inc. * 12345 Tricky Road, suite 9998 * Boston&lt;/p&gt;
&lt;/td&gt;
&lt;/tr&gt;
&lt;/table&gt;
&lt;/body&gt;
&lt;/html&gt;</pre></div><div data-bbox="91 646 519 661" data-label="Text"><p>Прокомментируем этот шаблон HTML и выделенные части:</p></div><div data-bbox="91 666 949 893" data-label="List-Group"><ol><li>1. Эта секция из заголовка документа должна попасть на нашу веб страницу, так как она ссылается на используемую таблицу стилей.<br/>Выделение: нужно удостовериться, что движок внешнего интерфейса вставит эту часть документа в секцию заголовка!</li><li>2. Меню слева создается секцией &lt;div&gt; на каждый элемент меню. Каждый из этих элементов &lt;div&gt; имеет класс. Дизайн элемента определяется CSS таблицей стилей по имени класса.<br/>Это очень разумный подход – создать меню, так что каждому элементу требуется из минимальное количество кода HTML (удобно для представления в TurboScript), элементы легко повторяемые (необходимо для динамического меню).<br/>Выделение: здесь, вместо фиктивного меню, нужно подставить динамически генерируемое!</li><li>3. Этот фиктивный содержимое Рафаэль поместил в файл шаблона просто для получения верного визуального представления. Отметим, что применено форматирование с помощью тегов &lt;h1&gt; и &lt;p&gt; (используется класс "bodytext") – это правильно, так как содержимое вставляемое TYPO3 позже будет их использовать для форматирования! (Рафаэль должно быть предварительно экспериментировал с TYPO3, не так ли?)<br/>Выделение: нужно заменить фиктивное содержимое на динамически генерируемое содержимое страницы.</li><li>4. Тег обрыва строки нужен просто для создания дополнительного места после содержимого, чтобы нижний колонтитул отодвинуть от основного текста на странице.</li></ol></div><div data-bbox="91 897 949 936" data-label="Text"><p>Наконец, Вы должны были обратить внимание, что табличные секции для меню и содержимого <i>помечены</i> с помощью <i>атрибутов id</i>. Это используется не только таблицей стилей! Использование атрибутов id – разумный подход! Но сначала немного теории о шаблонах HTML.</p></div><div data-bbox="97 942 222 970" data-label="Page-Footer"><img alt="TYPO3 logo"/></div><div data-bbox="550 949 949 964" data-label="Page-Footer"><p>Современная разработка шаблонов, Часть 1 (МТВ/1) – 23</p></div>
```

Основы ШАБЛОНА сObject

Для начала создадим новый файл, fileadmin/template/test.html, со следующим содержимым:

```
<!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>
```

(Можно найти в каталоге расширения руководства "misc/test.html")

Затем в поле Setup вашей записи шаблона поместим следующие строки:

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

# Объект PAGE по-умолчанию:
page = PAGE
page.typeNum = 0

page.10 < temp.mainTemplate
```

Таким образом мы помещаем объект сObject типа "TEMPLATE" в положение "page.10". Свойство "template" объекта TEMPLATE сObject определено как другой объект сObject типа "FILE", который читается из файла, "fileadmin/template/test.html". [Свойства объекта TEMPLATE сObject можно просмотреть здесь.](#)

Теперь, если Вы сохраните изменения в поле Setup и посмотрите во внешнем интерфейсе, то увидите:



А просмотр исходного кода говорит нам, что в основном ,объект TEMPLATE сObject просто читает файл и возвращает его необработанным:

```
<body bgcolor="#FFFFFF">
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

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

<body>

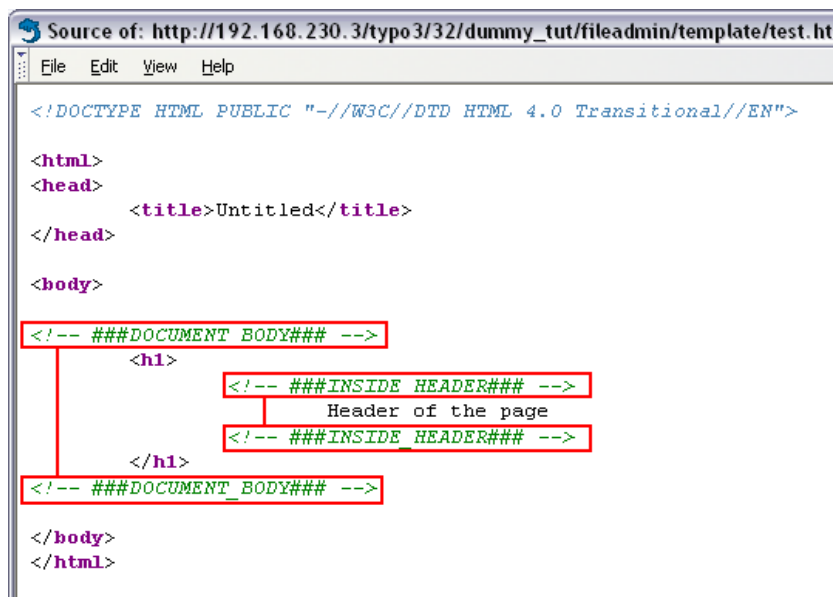
<!-- ###DOCUMENT_BODY### -->
  <h1>
    <!-- ###INSIDE_HEADER### -->
      Header of the page
    <!-- ###INSIDE_HEADER### -->
  </h1>
<!-- ###DOCUMENT_BODY### -->

</body>
</html>

</body>
</html>
```

Теперь подходит черед объекта TEMPLATE сObject; он не только читает файл HTML – он также позволяет нам извлекать подчасти внутри и заменять их динамическим содержимым!

Подчасть определяется как содержимое между парными строками схожих граничных маркеров ###, вставленных в парные комментарии HTML (хотя и не обязательно). Файл "test.html" имеет две подчасти, "DOCUMENT_BODY" и "INSIDE_HEADER". Как Вы видите, *маркеры подчастей* находятся внутри тегов комментариев HTML, что делает их невидимыми.



```
Source of: http://192.168.230.3/typo3/32/dummy_tut/fileadmin/template/test.ht
File Edit View Help

<!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>
```

Теперь, попытаемся изменить поле Setup записи шаблона на следующее:

```
# Объект содержимого шаблона (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!
}
```

```
# Объект PAGE по-умолчанию:
page = PAGE
page.typeNum = 0

page.10 < temp.mainTemplate
```

Перейдите во внешний интерфейс и Вы увидите следующий код HTML:

```
<body bgcolor="#FFFFFF">
    <h1>
        HELLO WORLD!
    </h1>
</body>
</html>
```

Изменения в свойствах объекта TEMPLATE сObject привели к следующему:

- Во-первых и в дальнейшем, объект TEMPLATE сObject вынужден работать только с подчастью, помеченной как "###DOCUMENT_BODY###" – таким образом, фиктивный заголовок и теги из тела были вырезаны!
- Во-вторых, подчасть помеченная как "###INSIDE_HEADER###" была *заменена* на отображаемое содержимое объекта TEXT сObject, определенное для свойства "subparts.INSIDE_HEADER" объекта TEMPLATE сObject.

Это просто чудесно усваивается. С этого момента, все что нам необходимо – указать на исходный файл шаблона Рафаэля, fileadmin/template/main/template_1.html, вставить подобные маркеры подчастей и заменить макет меню и макет содержимого на динамически генерируемые меню и настоящие элементы содержимого страницы!

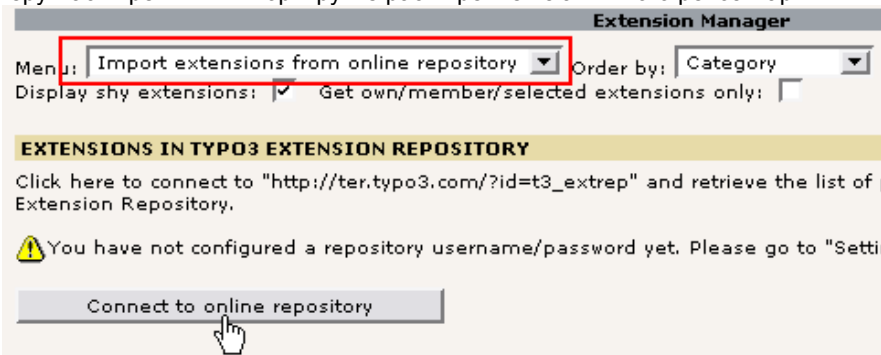
Расширение Анализатора Шаблона

Очевидно, что все действия кажутся простым редактированием шаблона Рафаэля. Хотя, судя по моему опыту, Вы не можете быть уверенными, что широко используемые HTML-редакторы не удалят или не переопределят комментарии HTML внутри документа, если вдруг позднее Рафаэль изменит файл шаблона. Просто представьте, что Рафаэль немного изменил шаблон в DreamWeaver и маркеры подчастей незначительно переместились, а ведь шаблон будет использоваться постоянно! Кроме того, пути, указывающие на таблицы стилей и картинки, в шаблоне Рафаэля *относятся* к каталогу [domain]/fileadmin/template/main/, а не к [domain]/ откуда будет отображаться содержимое HTML движком внешнего интерфейса TYPO3! Итак, все пути должны быть абсолютными!

Это делается из лучших побуждений. А теперь, пожалуйста, импортируйте и установите расширение анализатора шаблона(Template Auto-parser) – это предел ваших желаний...

Установка расширения

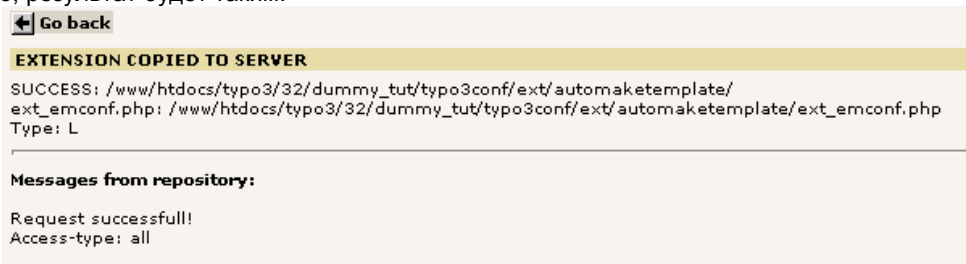
Перейдите к Менеджеру Расширений и импортируйте расширение из активного репозитория:



Найдите расширение "automaketemplate", кликните по иконке загрузки:

	Template Auto-parser	automaketemplate	0.0.9
	Tip-A-Friend	tipafriend	1.0.4

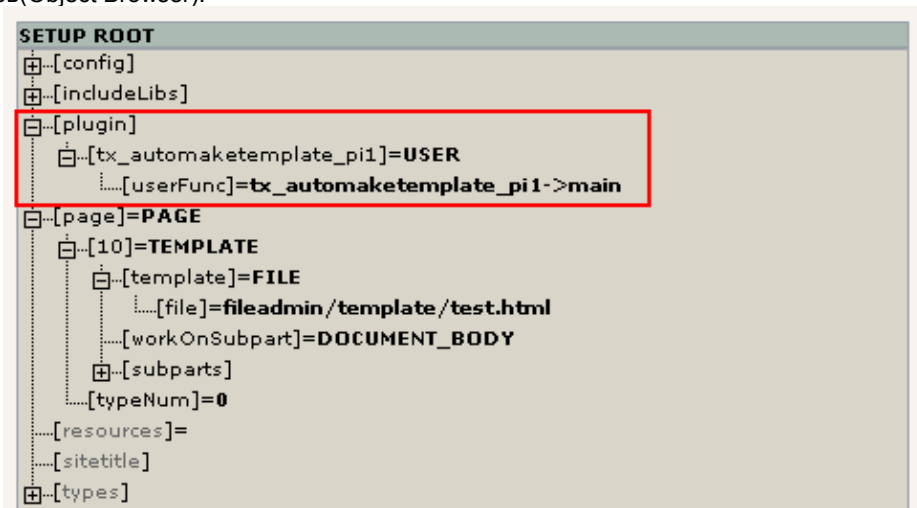
Если все хорошо, результат будет таким:



Вернитесь, выберите "Расширения доступные для установки"(Available Extensions to Install) и установите его:

	Template Auto-parser	automaketemplate	0.0.9
	Tip-A-Friend	tipafriend	1.0.4

После нажатия кнопки "Сделать обновления"(Make updates), вернитесь в модуль Шаблон(Template) и проверьте в Бrowsers Объектов(Object Browser):



Расширение добавило объект USER сObject в ветку "plugin.tx_automaketemplate_pi1" как Вы уже заметили. Этот объект сObject можно использовать вместо объекта FILE сObject для чтения файла шаблона Рафаэля, при этом появится возможность автоматически сделать маркеры и исправить относительные пути!

Настройка Анализатора Шаблона(Template Auto-parser)

Как и для любого расширения, нужно получить справку по нему в руководстве на сайте typos.org. [Если кликнуть по этой ссылке, то можно увидеть таблицу свойств объекта сObject.](#)

Для того чтобы Вам стало ясно, что такое Анализатор Шаблонов, я просто вставляю результат работы программы(plugin) как содержимое, а затем настрою объект типа PAGE в объекте "page" не выводить обычный заголовков и нижний колонтитул.

Это поле Setup содержимого записи шаблона:

```
# Конфигурирование Анализатора (Configuring the Auto-Parser):
plugin.tx_automaketemplate_pi1 {
    # Прочсть файл шаблона (Read the template file):
    content = FILE
    content.file = fileadmin/template/main/template_1.html

    # Здесь мы определим какие элементы HTML
    # будут вставлены в комментарии-подчасти (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
    }

    # Вставить этот префикс перед всеми относительными путями:
    relPathPrefix = fileadmin/template/main/
}

# Объект PAGE по-умолчанию:
page = PAGE
page.typeNum = 0
page.config.disableAllHeaderCode=1

page.10 =< plugin.tx_automaketemplate_pi1
```

Сохраним шаблон и перейдем во внешний интерфейс. Вы увидите в *точности* то, как выглядит fileadmin/template/main/template_1.html:

```

File Edit View Help

<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet href="#internalStyle" type="text/css"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><!--###DOCUMENT_HEADER### begin -->
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
    <style type="text/css" id="internalStyle">
        /*![CDATA[*
        BODY { margin: 0 0 0 0; background-color: white; }
        /*]]>*/
    </style>
    <link href="fileadmin/template/main/res/stylesheet.css" rel="stylesheet" type="text/css">
<!--###DOCUMENT_HEADER### end --></head>
<body><!--###DOCUMENT_BODY### begin -->
    <table border="0" cellpadding="0" cellspacing="0">

        <!-- Header image row: -->
        <tr>
            <td colspan="2" id="header_1"><!--###header_1### begin --><!--###menu_1### begin -->
                <div class="menu1-level1-no"><a href="fileadmin/template/main/#">Menu item 1</a></div>
                <div class="menu1-level1-no"><a href="fileadmin/template/main/#">Menu item 2</a></div>
                <div class="menu1-level1-act"><a href="fileadmin/template/main/#">Menu item 3 (act)</a></div>
                <div class="menu1-level2-no"><a href="fileadmin/template/main/#">Level 2 item</a></div>
                <div class="menu1-level2-no"><a href="fileadmin/template/main/#">Level 2 item</a></div>
                <div class="menu1-level2-act"><a href="fileadmin/template/main/#">Level 2 item (act)</a></div>
                <div class="menu1-level1-no"><a href="fileadmin/template/main/#">Menu item 2</a></div>
            <!--###menu_1### end --></td>

            <!-- Page Content Area table cell: -->
            <td id="content"><!--###content### begin -->
                <h1>Buy PaperShredder(tm) Gizmo with 30-days money-back guarantee!</h1>
                Adam Seth Enos Cainan Malelehel Iared Enoch Matusale Lamech Noe Sem Ham et
                <p class="bodytext">Filii Ham Chus et Mesraim Phut et Chanaan filii autem Chus Saba et Evila Sa
                <br />
            <!--###content### end --></td>

        </tr>

        <!-- Footer row: -->
        <tr>
            <td colspan="2" id="footer"><!--###footer### begin -->
                <p>Main Dish & Son inc. * 12345 Tricky Road, suite 9998 * Boston</p>
            <!--###footer### end --></td>
        </tr>
    </table>
<!--###DOCUMENT_BODY### end --></body>
</html>

```

Ну и что..? – думаете Вы. Но ведь это просто прекрасно, когда Вы взглянете на исходный код этой страницы, Вы поймете почему.

А пока, как Вы видите, произошли две важные вещи:

- Множество блоков в файле шаблона автоматически размещены внутри подчастей! (1+2)
- Для всех относительных ссылок используется префикс "fileadmin/template/main/" (3)

Это произошло потому, что Анализатор шаблона(Template Auto-parser) был настроен для управления “элементами” вроде этих:

```

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

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

    TD.all = 1
}
...

```

("Элементы" – это парные теги, например теги <td> . Теги без закрывающей части, например , должны быть определены со свойством "single" в Анализаторе шаблонов(Template Auto-parser))

Итак, конфигурация элементов говорит, что

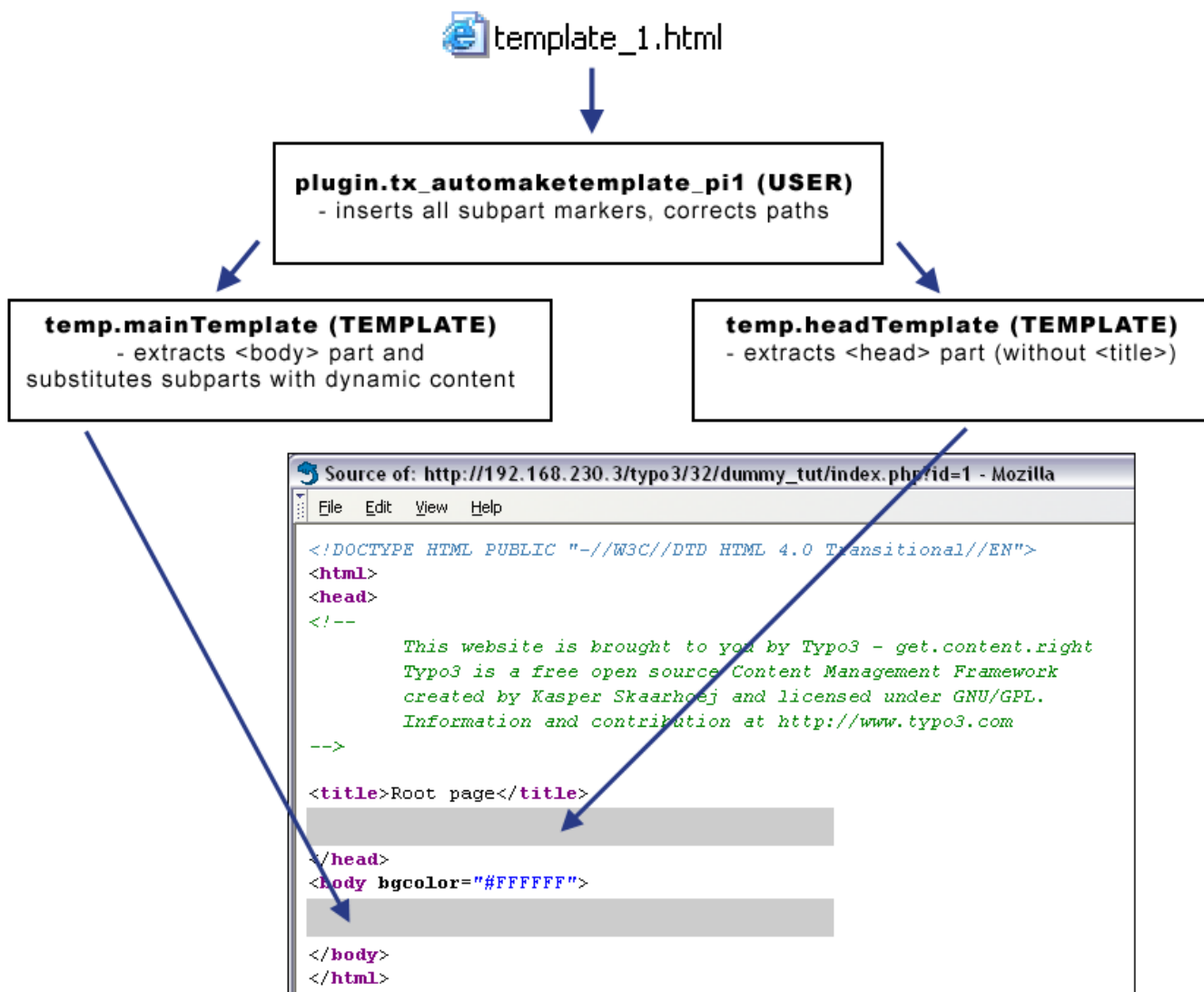
- а) элемент <body> должен быть заключен в маркер подчасти "###DOCUMENT_BODY###"
- б) элемент <head> должен быть заключен в маркер подчасти "###DOCUMENT_HEADER###". В дальнейшем, любая секция <title> должна быть удалена. (Нам не нужно воспроизводить тег <title> шаблона Рафаэля на наших страницах TYPO3...)
- в) Все найденные элементы <td> должны быть заключены в маркеры подчастей, но, так как величина (.subpartMarker) не была определена , будут заключены и затем, заменены на значения соответствующих id/class маркеров подчастей только теги <td> с атрибутами "id" или "class". Таким образом, тег <td id="menu_1"> будет содержать *внутри* содержимое помещенное в маркеры подчастей "<!--###menu_1###-->...<!--###menu_1###-->".

А теперь, как это можно использовать?

Хорошо, ответ заключается в том, что Анализатор(Auto-parser) позволяет Рафаэлю создавать шаблоны с использованием современной технологии таблиц стилей с *осознанным использованием идентификатора id и класса атрибутов* в элементах HTML, и, *в то же время*, эти атрибуты служат "маркерами", которые TYPO3 использует для замены части шаблона на динамический содержимое! Легко для Рафаэля (дизайнера), не так трудно для Бенуа (разработчика) и хорошо для финансов Писаки, так как требуется меньше времени для преобразования и ручной разметки шаблона маркерами подчастей!

Все вместе

Разберем иллюстрацию ниже. Вот что мы хотели сделать:



Файл шаблона читается Анализатором(Auto-parser), результат передается объекту TEMPLATE сObject, который заменяет подчасти и окончательно вставляет в секции заголовка и тела страницы содержимое из TYPO3.

Это выполняет TypoScript, как показано ниже, после ввода в поле Setup вашей записи шаблона. Это длинный листинг, но попытайтесь время чтобы разобраться в нем:

```
# Настройка Анализатора(Configuring the Auto-Parser) для основного шаблона:
plugin.tx_automaketemplate_pil {
    # Прочитать файл шаблона:
    content = FILE
    content.file = fileadmin/template/main/template_1.html

    # Здесь мы определим какие элементы HTML
    # будут вставлены в комментарии-подчасти (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
    }

    # Вставить этот префикс перед всеми относительными путями:
    relPathPrefix = fileadmin/template/main/
}

# Основной объект TEMPLATE сObject для BODY
temp.mainTemplate = TEMPLATE
temp.mainTemplate {
    # Посылка результата из Анализатора (Auto-parser) объекту TEMPLATE сObject:
    template =< plugin.tx_automaketemplate_pil
    # Выбрать только содержимое между тегами <body>
    workOnSubpart = DOCUMENT_BODY

    # Заменить подчасть ###menu_1### некоторым примерным контентом:
    subparts.menu_1 = TEXT
    subparts.menu_1.value = HELLO WORLD - MENU

    # Заменить подчасть ###content### некоторым примерным контентом:
    subparts.content = TEXT
    subparts.content.value = HELLO WORLD - CONTENT
}

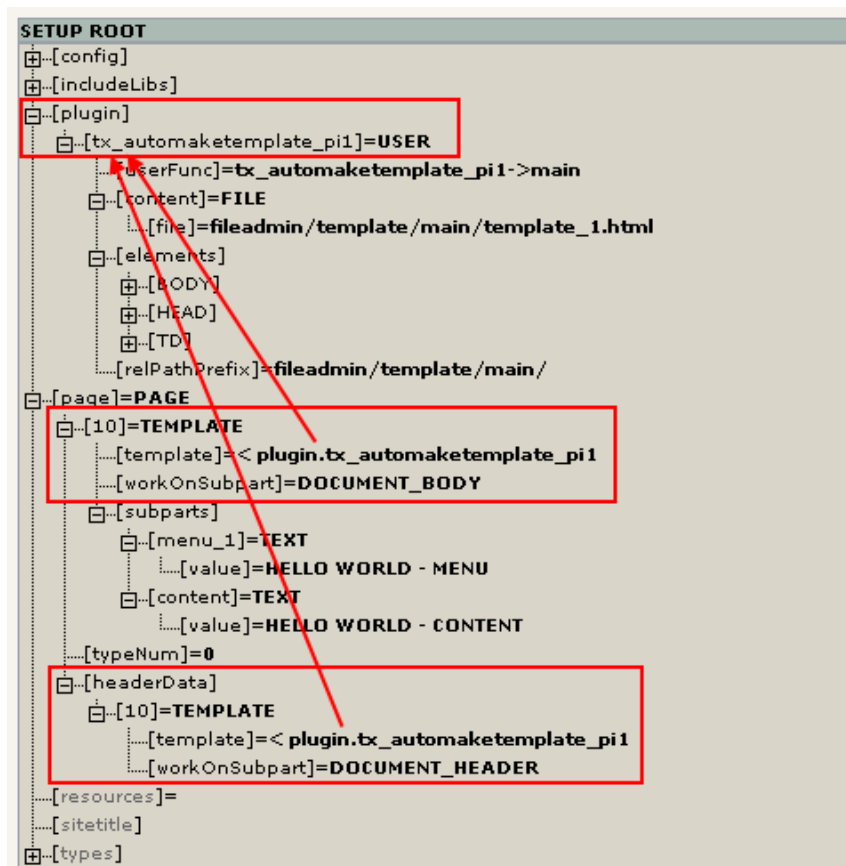
# Основной объект TEMPLATE сObject для HEAD
temp.headTemplate = TEMPLATE
temp.headTemplate {
    # Посылка результата из Анализатора (Auto-parser) объекту TEMPLATE сObject:
    template =< plugin.tx_automaketemplate_pil
    # Выбрать только содержимое между тегами <head>
    workOnSubpart = DOCUMENT_HEADER
}

# Объект PAGE по-умолчанию:
page = PAGE
page.typeNum = 0

# Копирование содержимого из TEMPLATE для секции <body>:
page.10 < temp.mainTemplate

# Копирование содержимого из TEMPLATE для секции <head>:
page.headerData.10 < temp.headTemplate
```

Все это отражается следующей структурой дерева:



Как Вы можете видеть, объекты "temp.mainTemplate" и "temp.headTemplate" сObject были скопированы в соответствующие позиции дерева объектов. Для каждого из них, сохраняются ссылки на объект USER сObject из программы Анализатора Шаблона(Template Auto-parser plugin).

Отметим, что часть объекта "page.headerData" является числовым массивом объектов содержимого, определяющих содержимое секции <head> на странице. Вот так Вы добавляете содержимое между тегами <head> для страниц генерируемых TYPO3.

Результат

Результат выглядит вот так:



Отметим, что области меню и содержимого были заменены контентом объекта TEXT сObject, которые здесь используются в качестве тестовых!

Исходный текст на HTML выглядит вот так:

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<style type="text/css" id="internalStyle">
  /**/
  BODY { margin: 0 0 0 0; background-color: white; }
  /*]]&gt;*/
&lt;/style&gt;
&lt;link href="fileadmin/template/main/res/stylesheets.css" rel="stylesheet" type="text/css"&gt;
&lt;title&gt;Root page&lt;/title&gt;
&lt;meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"&gt;
&lt;meta name="generator" content="Typo 3.3 CMS"&gt;
&lt;script language="javascript" type="text/javascript"&gt;
&lt;!--
  browserName = navigator.appName;
  browserVer = parseInt(navigator.appVersion);
  var msie4 = (browserName == "Microsoft Internet Explorer" &amp;&amp; browserVer &gt;= 4);
  if ((browserName == "Netscape" &amp;&amp; browserVer &gt;= 3) || msie4 || browserName=="Konqueror") {
  function blurLink(theObject) {
    if (msie4) (theObject.blur());
  }
  // --&gt;
&lt;/script&gt;
&lt;/head&gt;
&lt;body bgcolor="#FFFFFF"&gt;
  &lt;table border="0" cellpadding="0" cellspacing="0"&gt;
    &lt;!-- Header image row: --&gt;
    &lt;tr&gt;
      &lt;td colspan="2" id="header_1"&gt;&lt;!--###header_1### begin --&gt; &lt;img src="fileadmin/temp
    &lt;/tr&gt;
    &lt;tr&gt;
      &lt;td id="menu_1"&gt;HELLO WORLD - MENU&lt;/td&gt;
      &lt;td id="content"&gt;HELLO WORLD - CONTENT&lt;/td&gt;
    &lt;/tr&gt;
    &lt;!-- Footer row: --&gt;
    &lt;tr&gt;
      &lt;td colspan="2" id="footer"&gt;&lt;!--###footer### begin --&gt;
        &lt;p&gt;Main Dish &amp;amp; Son inc. * 12345 Tricky Road, suite 9998 * Boston&lt;/p&gt;
      &lt;!--###footer### end --&gt;&lt;/td&gt;
    &lt;/tr&gt;
  &lt;/table&gt;
&lt;/body&gt;
&lt;/html&gt;</pre></div><div data-bbox="58 560 907 656" data-label="List-Group"><ol><li>1. Секция &lt;head&gt; вставлена без тега &lt;title&gt;</li><li>2. Вставлена секция &lt;body&gt;, замена содержимого выполнена</li><li>3. Подчасти <b>###header_1###</b> и <b>###footer###</b> не были подставлены, так как они все еще не определены в "temp.mainTemplate" TEMPLATE cObject! Итак, ошибок нет.</li><li>4. Подчасти <b>###menu_1###</b> и <b>###content###</b> были корректно подставлены, как и ожидалось, поскольку объект "temp.mainTemplate" TEMPLATE cObject был настроен для этого.</li></ol></div><div data-bbox="58 671 134 687" data-label="Section-Header"><h2>Резюме</h2></div><div data-bbox="58 688 907 727" data-label="Text"><p>Итак, из шаблона Рафаэля, TYPO3 автоматически извлекает содержимое &lt;head&gt; и &lt;body&gt;, корректирует относительные пути для рисунков, форм и таблиц стилей, выполняет подстановки динамического содержимого, и вставляет каждую часть в секцию &lt;head&gt; и &lt;body&gt; в страницу вывода из TYPO3.</p></div><div data-bbox="58 731 907 769" data-label="Text"><p>Основная структура запущена и работает – нам только нужно добавить некоторое динамическое содержимое в области меню и содержимого! Это весьма эффективно выполняет TypoScript cObjects для отображения элементов содержимого и меню.</p></div><div data-bbox="62 942 188 970" data-label="Page-Footer"><img alt="TYPO3 logo" data-bbox="62 942 188 970"/></div><div data-bbox="519 950 907 964" data-label="Page-Footer"><p>Современная разработка шаблонов, Часть 1 (МТВ/1) – 32</p></div>
```

Создание меню

Двухуровневое меню, в левой части шаблона, должно генерироваться динамически для отображения структуры страниц из внутреннего интерфейса. Хотя Рафаэль сделал все красиво, чистый шаблон с очень простой разметкой пунктов меню *не* является лучшим решением для проверки и извлечения этих частей с помощью Анализатора (Auto-parser); попробуем вместо этого их запрограммировать! Это означает, что Бенуа должен сам просмотреть шаблон Рафаэля, аккуратно определить из чего состоят *пункты простого меню* и использовать их внутри записи шаблона для создания объекта для генерации меню. Это также означает, что Рафаэль не может изменить основную конфигурацию меню без согласования изменений для записи шаблона с Бенуа! Но если Рафаэль сделает эту работу с умом, то этого не потребуется; для всех визуальных эффектов будут использоваться стили CSS.

Объекты меню и состояние пунктов

Во-первых, объектом сObject используемым для генерации меню является объект HMENU. Этот объект вызывает объекты "меню"-типов: TMENU или GMENU, или GMENU_LAYERS и т.п.; для каждого требуемого уровня меню (какой из них выбрать, зависит от того, какой тип меню Вы хотите отобразить – текстовый, графический, каскадные и т.д.).

Для каждого "меню-объекта" (например, TMENU или GMENU) Вы определяете основные свойства для каждого уровня представления и, дополнительно, свойства специфичные для пунктов (например ширину и высоту графического пункта GMENU). Специфичные свойства для пунктов всегда определяются для определенных *состояний* этих пунктов. Нормальное состояние (NO) должно быть определено всегда, но в дополнение к этому можно определить, например, активное состояние пункта (ACT) – что может означать "как будет выглядеть пункт меню, если мы на этой странице или на странице ниже этой".

Хорошо, целью этого руководства не является обучение особенностям объекта HMENU сObject и всех связанных с этим вопросов. Вам следует [обратиться к TypoScript by Example](#) для этого.

А теперь, взглянем на исходный текст HTML шаблона Рафаэля:

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



Как Вы можете видеть, он использовал теги <div> для каждого пункта меню, независимо от уровня и состояния. Существенная разница видна в значениях атрибутов класса! Это мудрый дизайн, так как разметка становится такой простой, что у Бенуа будет сэкономлено время на реализацию! :-) А полный, визуальный контроль находится вне TYPO3 – в таблице стилей CSS.

Отметим также, каким образом Рафаэль осуществил дизайн "активных" пунктов меню.

Реализация такого меню крайне легкая. Это делается так:

Сначала, определяем временный объект, где-то ближе к вершине записи шаблона (перед определением "temp.mainTemplate"):

```
# Menu 1 сObject
temp.menu_1 = HMENU
# Первый уровень объекта-меню, текстовый
temp.menu_1.1 = TMENU
temp.menu_1.1 {
  # Нормальное состояние свойств
  NO.allWrap = <div class="menul-level1-no"> | </div>
  # Включить активное состояние и установить свойства:
  ACT = 1
  ACT.allWrap = <div class="menul-level1-act"> | </div>
}
# Второй уровень объекта-меню, текстовый
temp.menu_1.2 = TMENU
temp.menu_1.2 {
  # Нормальное состояние свойств
  NO.allWrap = <div class="menul-level2-no"> | </div>
  # Включить активное состояние и установить свойства:
  ACT = 1
  ACT.allWrap = <div class="menul-level2-act"> | </div>
}
```

Отметим строки выделенные красным цветом, содержащие разметку HTML, которые Бенуа вырезал из шаблона Рафаэля. Теперь они запрограммированы в записи шаблона.

Единственная вещь, которую осталось сделать – скопировать этот объект cObject в подчасть "menu_1" в "temp.mainTemplate" cObject:

```
...
# Основной объект TEMPLATE cObject для BODY
temp.mainTemplate = TEMPLATE
temp.mainTemplate {
  # Псылка содержимого из Анализатора (Auto-parser) объекту TEMPLATE cObject:
  template =< plugin.tx_automaketemplate_pi1
  # Выбор только содержимого между тегами <body>
  workOnSubpart = DOCUMENT_BODY

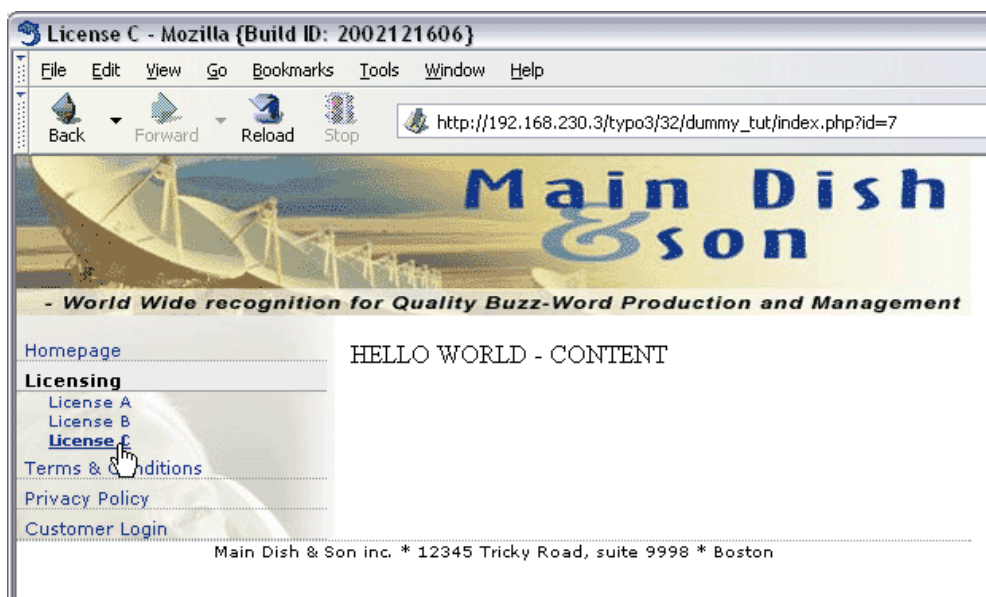
  # Подставить в подчасть ###menu_1### динамическое меню:
  subparts.menu_1 < temp.menu_1

  # Подставить в подчасть ###content### некоторый содержимое для примера:
  subparts.content = TEXT
  subparts.content.value = HELLO WORLD - CONTENT
...

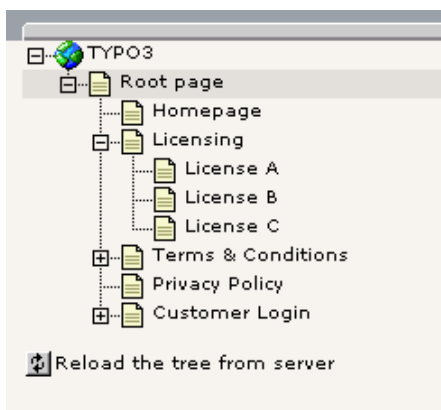
```

(Красные строчки отражают изменения!)

... и результат говорит сам за себя:



В меню, выше, отображено именно это дерево страниц!



Однако, можно ли легко изменить дизайн меню? Хорошо, попробуйте и отредактируйте таблицу стилей в "fileadmin/template/main/res/styleSheet.css":

```
25
26 /* Menu 1 column */
27 TD#menu_1 {
28     vertical-align: top;
29     width: 200px;
30     background-image: url(../images/menubackground.jpg);
31     background-repeat : no-repeat;
32     padding-top: 10px;
33 }
34 TD#menu_1 DIV {
35     width: 95%;
36 }
37 TD#menu_1 DIV A {
38     color: navy;
39     text-decoration: none;
40 }
41 TD#menu_1 DIV A:hover {
42     text-decoration: underline;
43 }
44
45 /* MENU 1, level 1, normal state (NO) */
46 TD#menu_1 DIV.menul-level1-no {
47     border-bottom: 1px dotted #999999;
48     font-size: 11px;
49     padding-top: 5px;
50     padding-left: 5px;
51 }
52 /* MENU 1, level 1, active state (ACT) */
53 TD#menu_1 DIV.menul-level1-act {
54     border-bottom: 1px solid #999999;
55     font-weight: bold;
56     font-size: 11px;
57     padding-top: 5px;
58     padding-left: 5px;
59
60     background-color: #eeeeee;
61     filter: alpha(opacity='70', style='0');
62 }
63 TD#menu_1 DIV.menul-level1-act A {
64     color: black;
65 }
66
67 /* MENU 1, level 2, normal state (NO) */
68 TD#menu_1 DIV.menul-level2-no {
69     font-size: 10px;
70     padding-left: 20px;
71 }
72 /* MENU 1, level 2, active state (ACT) */
73 TD#menu_1 DIV.menul-level2-act {
74     font-size: 10px;
75     font-weight: bold;
76     padding-left: 20px;
77 }
78
```

Посмотрите! Такой цельный подход позволяет Вам поместить важнейшие факторы управления визуальным дизайном – шаблон HTML и таблицу стилей, *вне* TYPO3 для удобного доступа непосредственно для Рафаэля, дизайнера HTML/CSS. И от Бенуа потребовалось выполнить минимальную работу по программированию на TurboScript внутри записи шаблона – он просто настроил движок внешнего интерфейса, чтобы дизайн Рафаэля работал с учетом предпочтений в средствах разработки, и наилучшим образом!

Пожалуйста ознакомьтесь с TurboScript by Example, если Вы хотите [потренироваться с объектами HMENU и связанными с ними](#).

Вставка страничного содержимого

Самым известным способом управления содержимым страниц в TYPO3 является создание *элементов содержимого* на странице в таблице `tt_content`. Действительно, мне не известны люди *не* использующие эту концепцию, так как она легко доступна, мощная и дает быстрый результат. Тем не менее, она может немного отличаться от вашего предубеждения о внедрении структурированного содержимого в шаблоны HTML.

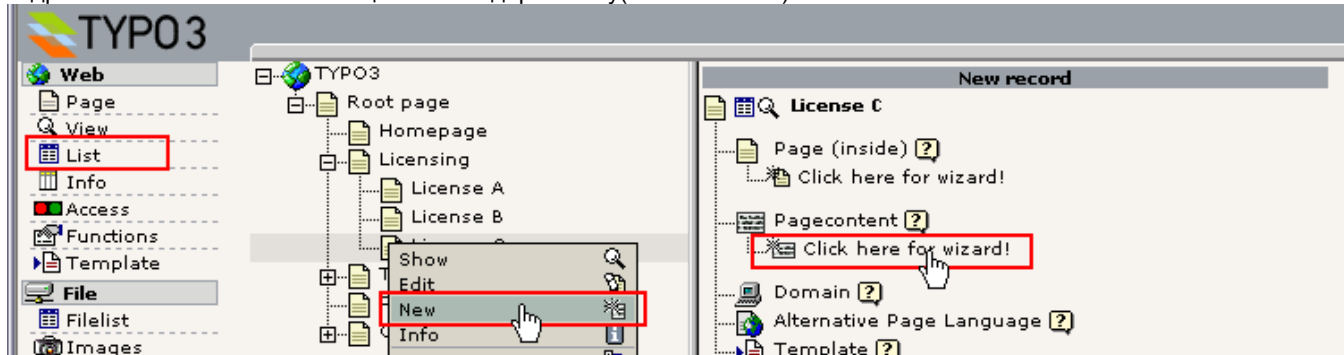
Элементы содержимого имеют многочисленные predefined типы (Text, Text w/image, Bullelist, Table, Login box, и т.д.) и *статические шаблоны* с predefined TypoScript, которые обычно идеально отображают ваши элементы. Это настоящее удовольствие! Так как *элементы содержимого* имеют тип и для каждого типа много дополнительных параметров, то невозможно отображать элементы содержимого в жесткой концепции, вроде по шаблону HTML на каждый тип элемента. Это просто не проходит и Вы поймете это немного попрактиковавшись. Поэтому, отображение элементов содержимого со сложными условиями осуществляют объекты сObject скомбинированные с PHP-функциями.

Итак, если мы не можем создавать элементы содержимого с помощью шаблонов HTML, что же нам тогда делать? Хорошо, если Вы используете самый современный подход к отображению элементов содержимого – то есть расширение "css_styled_content", тогда все элементы содержимого будут заключены в примитивы элементов HTML, что удачно переносит решения по дизайну на вашу внешнюю таблицу стилей CSS!

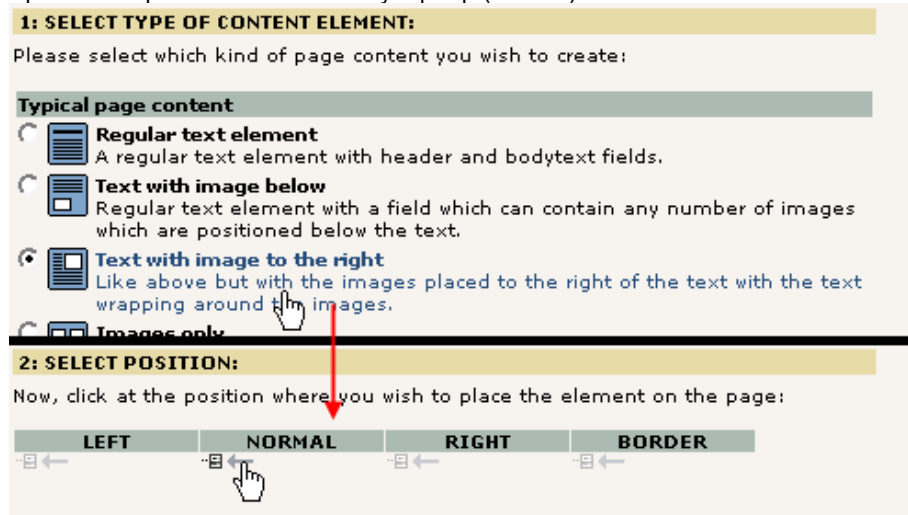
Давайте посмотрим как это работает. Сначала, создадим элемент содержимого:

Элемент содержимого

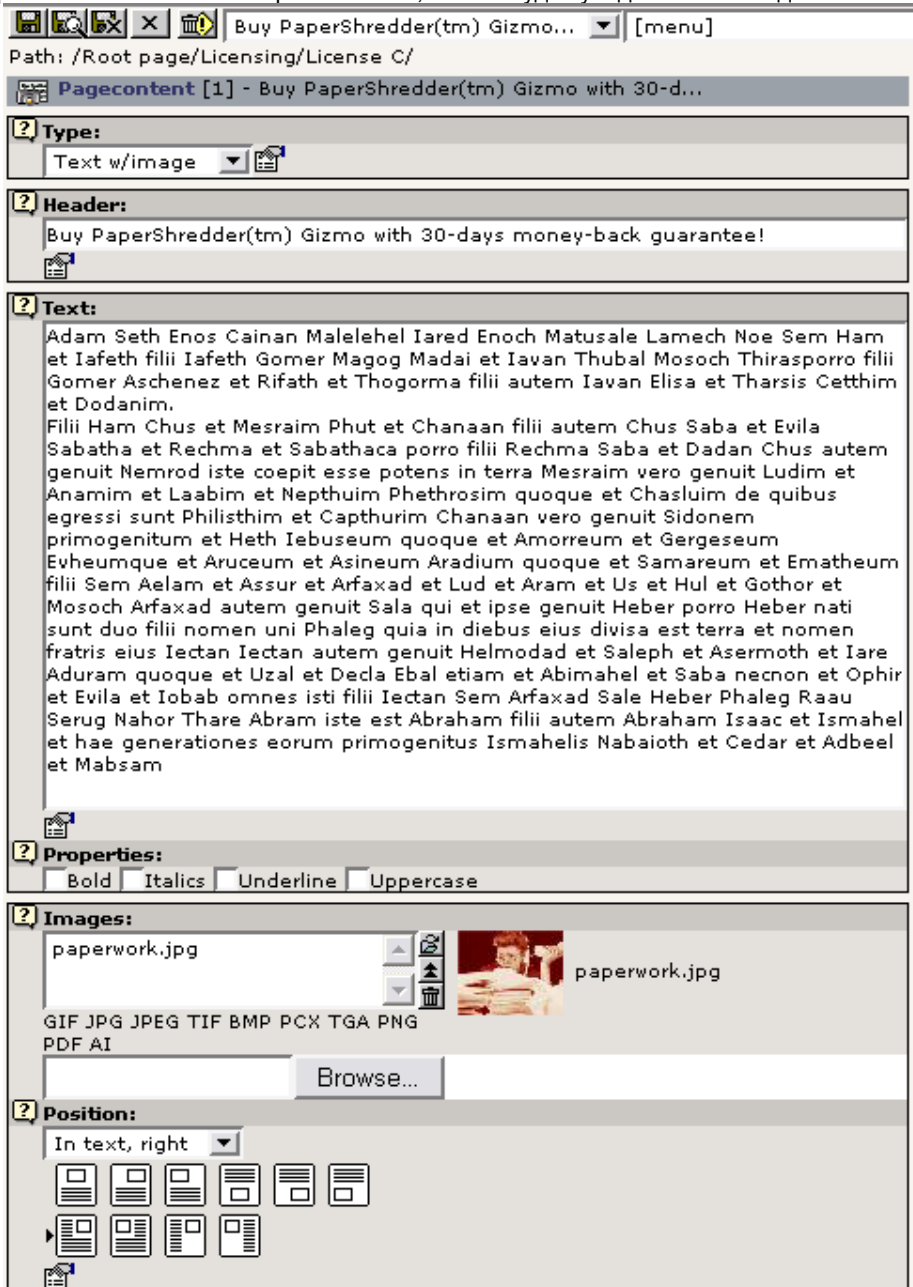
В модуле Список(List), кликнем по иконке страницы "License C", затем выберем "Новый"(New) и, после этого, в правом кадре кликнем по ссылке Помощника по содержимому(content wizard):



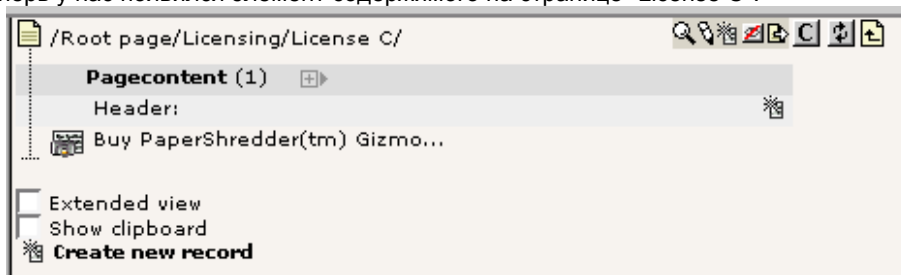
Выберем "Текст с картинкой справа" и затем колонку "Центр"(Normal):



После ввода содержимого в элемент и сохранения его , можно будет увидеть что-то подобное:



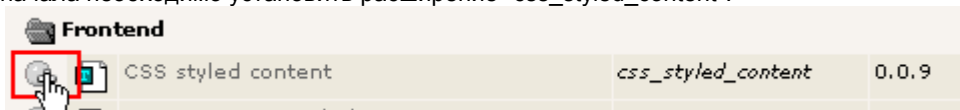
Таким образом, теперь у нас появился элемент содержимого на странице "License C":



Подключение статических шаблонов

Для вывода элемента содержимого нам нужно подключить *статический шаблон*, который обеспечит нас многими сотнями строк кода TurboScript для отображения. Это делается с помощью редактирования целой записи шаблона.

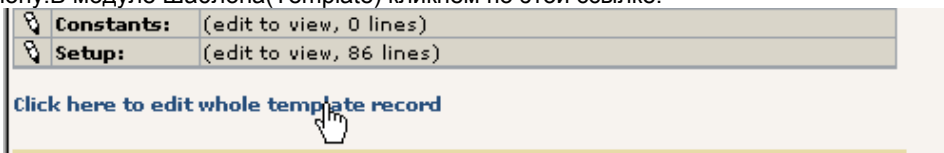
Тем не менее, сначала необходимо установить расширение "css_styled_content":



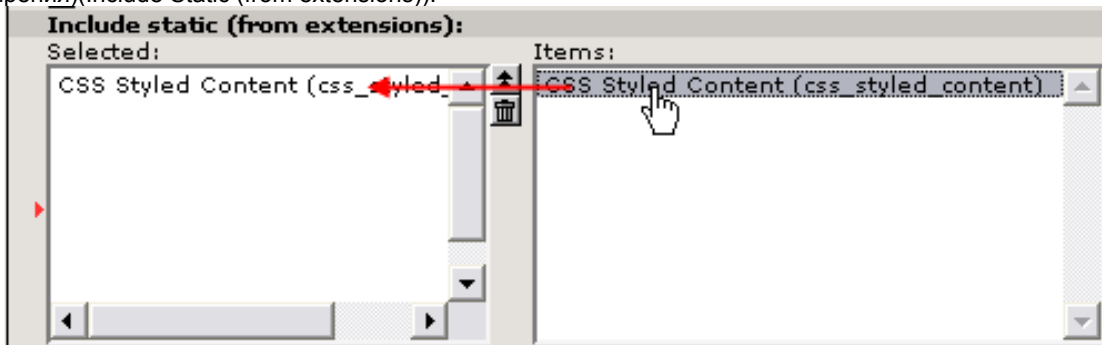
Просто кликните по кнопке Установки и нажмите кнопку подтверждения на следующей странице.

(Замечание: Во время написания этого руководства (январь 2004) модуль "CSS styled content" еще НЕ был завершен! Он первый среди трех важнейших дел Каспера (Kasper) и предназначен для немедленной разработки после катания на сноуборде (snowboard tour)).

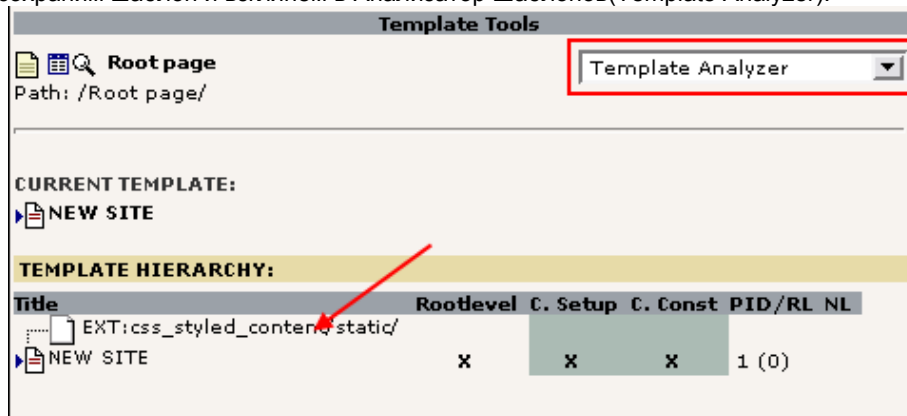
Вернемся к шаблону: В модуле Шаблона(Template) кликнем по этой ссылке:



После чего Вы увидите статический файл "CSS Styled Content" доступный в блоке выбора "Подключить статический (из расширения)(Include Static (from extensions))):



Кликнем по нему, сохраним шаблон и взглянем в Анализатор Шаблонов(Template Analyzer):



Анализатор Шаблонов показывает иерархию записей шаблонов и статических шаблонов, которые могут подключаться к текущей "основной записи шаблона"(main template record). Выбирая статический шаблон для вставки, мы просто включаем его перед кодом отображения TypoScript внутри записи шаблона "NEW SITE". Это значит, что любой объект определенный внутри статического шаблона "EXT:css_styled_content/...." может быть скопирован и использован в записи шаблона NEW SITE. Вот это мы сейчас и сделаем, так как подключенный статический шаблон содержит объект "styles.content.get", который является элементом содержимого для выбора всех элементов содержимого в колонке "Центр"(Normal) текущей страницы:

Итак, в записи шаблона, измените часть определения объекта "temp.mainTemplate":

```
...
# Основной объект TEMPLATE cObject для BODY
temp.mainTemplate = TEMPLATE
temp.mainTemplate {
    # Посылка содержимого из Анализатора (Auto-parser) объекту TEMPLATE cObject:
    template =< plugin.tx_automaketemplate_pi1
    # Выбор только содержимого между тегами <body>
    workOnSubpart = DOCUMENT_BODY

    # Подставить в подчасть ###menu_1### динамическое меню:
    subparts.menu_1 < temp.menu_1

    # Подставить в подчасть ###content### некоторый содержимое для примера:
    subparts.content < styles.content.get
...

```

Если Вы обновите во внешнем интерфейсе страницу "License C" Вы теперь увидите это:



В исходном коде HTML мы видим это:

```

<table border="0" cellpadding="0" cellspacing="0">
    <!-- Header image row: -->
    <tr>
        <td colspan="2" id="header_1"><!--###header_1### begin --><div class="menu1-level1-no"><a href="index.php?id=6" onFocus="blurLink(this);">Homepage</a></div
    </td>
    </tr>
    <!-- Page Content Area table cell: -->
    <td id="content"><a name="1"></a>
    <h1>Buy PaperShredder(tm) Gizmo with 30-days money-back guarantee!</h1><table width="203" border=0 cellspacing=0 cellpadding=0 al
    </p>
    <p class="bodytext">Fili Ham Chus et Mesraim Phut et Chanaan filii autem Chus Saba et Evila Sabatha et Rechma et Sabathaca porro
    </p>
    <!-- Footer row: -->
    <tr>
        <td colspan="2" id="footer"><!--###Footer### begin -->
        <p>Main Dish & Son inc. * 12345 Tricky Road, suite 9998 * Boston</p>
        <!--###Footer### end --></td>
    </tr>
</table>

```

1. Вероятно, элемент содержимого заголовка был заключен в теги <h1>. Если Вы выберете другой тип "Layout" для заголовка, то получите отображение с меткой, например, <h2> (для Layout 2).
2. Каждая строка основного текста(bodytext) заключена в теги <p> с установкой класса "bodytext". Таким образом, возможно указание определенного стиля содержимого страницы из элемента содержимого в вашей таблице стилей! Отметим, что Рафаэль уже создал дизайн макета содержимого в файлах шаблона, используя теги <p class="bodytext">!
3. Каждый вставленный элемент содержимого имеет тег <a> с именем, установленным в uid элемента содержимого. Этот якорь предназначен для прямого доступа к определенному месту на странице с использованием URL.
4. Это отображение двухуровневого меню из объектов HMENU. Просто отметьте, каким образом ссылка автоматически настроена на верную страницу и даже установлен дополнительный указатель onFocus.

А что в дереве Объектов (Object Tree)?

А теперь, копируя предлагаемый элемент содержимого, "styles.content.get", что в действительности мы вставляем в дерево объектов? Давайте посмотрим в Бrowsersе Объектов(Object Browser):



Мы обнаружим две интересные вещи:

1. Часть объекта "styles.content.get" явно содержала объект cObject типа CONTENT. Глядя на атрибуты, есть смысл сделать вывод о том, что выбираются записи из таблицы "tt_content", упорядоченные по полю, "sorting". Подробности [об объекте CONTENT cObject смотрите здесь](#).
2. Во-вторых, определен новый Объект Вернего Уровня – TLO (Toplevel Object) – "tt_content". По-умолчанию, объект CONTENT cObject выбирая записи из "tt_content" (1) будет использовать этот TLO (также определенный как cObject) для отображения каждой найденной записи!
Как видно, таблица "tt_content" TLO определена как объект USER cObject содержащий функцию PHP из расширения "css_styled_content" – как мне кажется, все верно, так как устанавливая расширение мы этого и добивались! Более детально с этим можно ознакомиться [в документации по расширению "css_styled_content"](#).

Для большей достоверности, мы можем даже заглянуть внутрь класса в "tx_cssstyledcontent_pi1" и разобраться с логикой. Приведем часть кода функции main():

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

    // This value is the Content Element Type - determines WHAT kind of element to render...
    $ContentType = (string)$this->cObj->data["ContentType"];
    $content="";
    switch ($ContentType)        {
        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;
    }
}
```

...

Добавление совместимости с XHTML

Если Вы хотите создавать сайты совместимые с XHTML, то в TYPO3 это возможно. Версия 3.6.0 TYPO3 поддерживает совместимость с XHTML, насколько это сегодня возможно (уровень: XHTML transitional).

Для этого руководства необходимо, чтобы Вы удостоверились, что используемые шаблоны совместимы с XHTML. Это, конечно, первый шаг и именно то, в чем *Вы* должны быть уверены.

Следующий шаг – заставить TYPO3 генерировать тип документа XHTML. Это легко достижимо с помощью простой строчки TurboScript, помещенной в поле Setup основной записи шаблона:

```
page.config.doctype = xhtml_trans
```

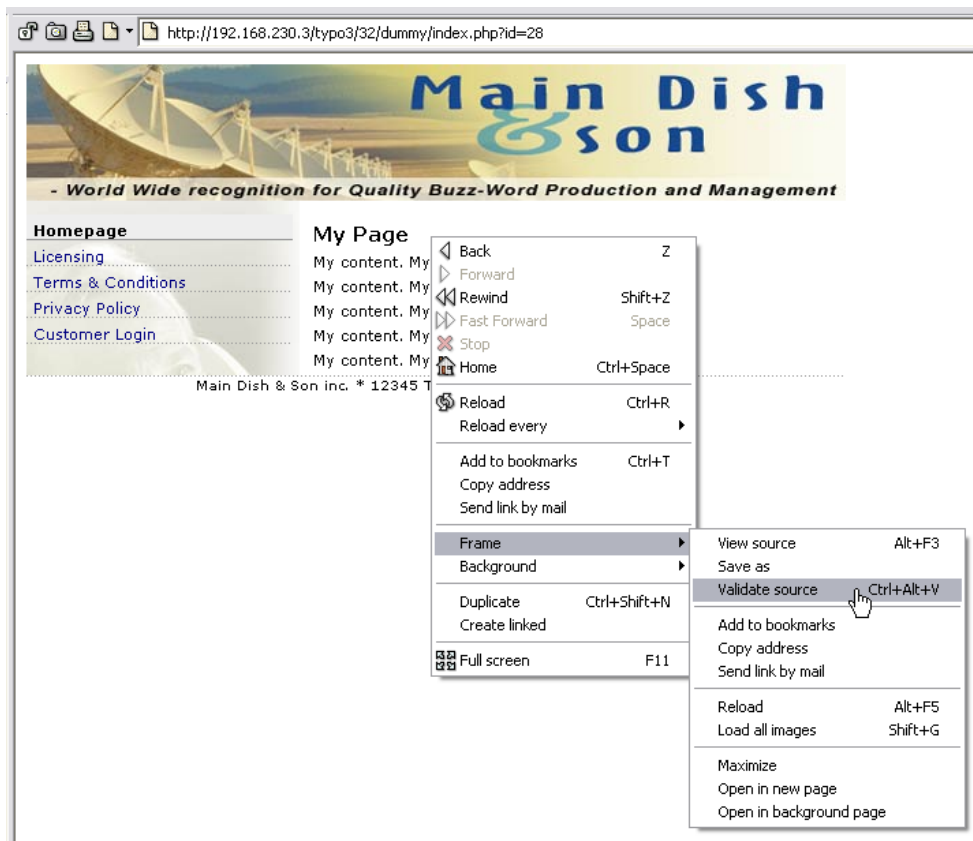
В дальнейшем, меню также станут совместимы. А пока, специальные символы, вроде “&”, выводятся как есть. Этого следует избегать. Итак, код меню TurboScript следует дополнить четырьмя строками (здесь выделены красным):

```
# Menu 1 cObject
temp.menu_1 = HMENU
# Объект меню первого уровня, текстовый
temp.menu_1.1 = TMENU
temp.menu_1.1 {
# Нормальное состояние свойств
NO.allWrap = <div class="menu1-level1-no"> | </div>
NO.stdWrap.htmlSpecialChars = 1
# Включить состояние активности и задать свойства:
ACT = 1
ACT.stdWrap.htmlSpecialChars = 1
ACT.allWrap = <div class="menu1-level1-act"> | </div>
}
# Объект меню второго уровня, текстовый
temp.menu_1.2 = TMENU
temp.menu_1.2 {
# Нормальное состояние свойств
NO.allWrap = <div class="menu1-level2-no"> | </div>
NO.stdWrap.htmlSpecialChars = 1
# Включить состояние активности и задать свойства:
ACT = 1
ACT.stdWrap.htmlSpecialChars = 1
ACT.allWrap = <div class="menu1-level2-act"> | </div>
}
```

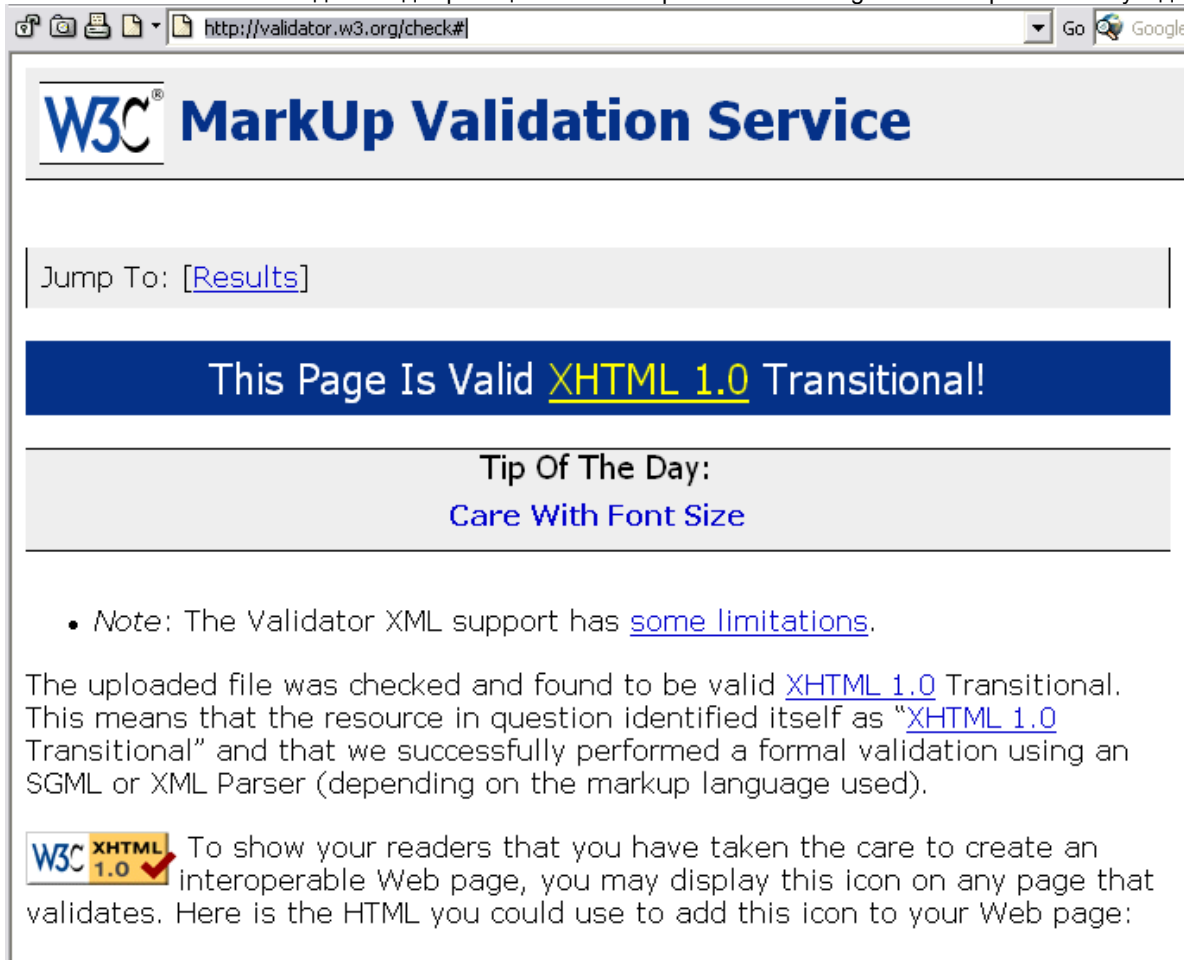
Тогда можно быть уверенным в том, что метки меню пройдут через функцию PHP htmlspecialchars() и специальные символы будут преобразованы, например “&” в “&”

После того как Вы сделаете это, можно протестировать сайт браузером Opera 7, который очень удобен для проверки совместимости.

На странице HTML кликните по правой кнопке мыши, выберите “Frame”, затем “Проверить исходный код”:



Opera автоматически пошлет исходный код страницы на сайт "http://validator.w3.org/" и по завершении Вы увидите:



Одновременная очистка

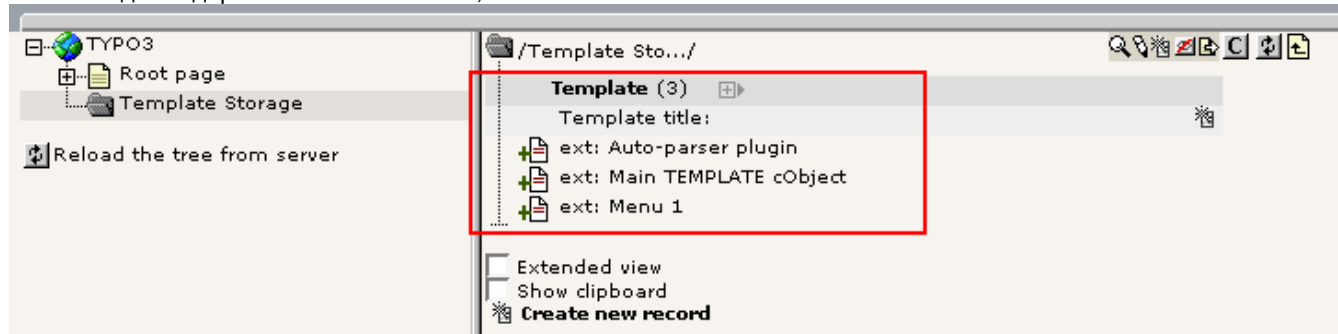
Мы завершили создание нашего сайта. Бенуа и Рафаэль довели свою работу до конца за приемлемое время, Писака возможно даже наполнил страницы сайта содержимым? Мы не знаем, но основа сайта готова, она сделана на основе обычного файла HTML, помещенного в каталог "fileadmin/template/main/", только с меню и областями для содержимого, динамически заменяемого при необходимости.

Лучшая структура шаблона

Единственное, что мы упустили – запись шаблона можно сделать чуть лучше. Несмотря на минимальное количество используемого кода TurboScript, мы легко можем убедиться, что такая запись шаблона очень быстро становится большой и неуправляемый. Выход – создание серии записей шаблонов для включения:

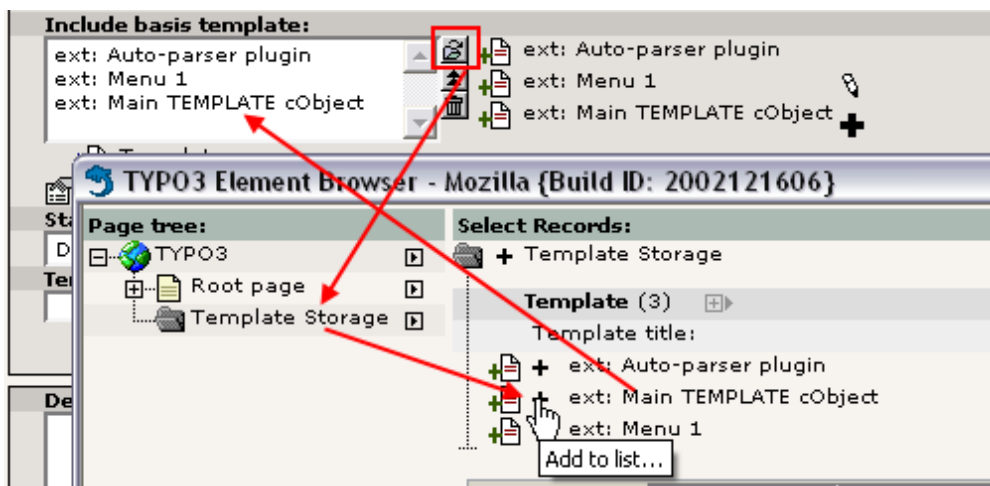
Сначала можно создать целую новую страницу в корне дерева страниц. Выбрать тип "системный каталог" (sysFolder) (что может означать "место для хранения любого желаемого элемента базы данных") и назовите его Хранилище Шаблона (Template Storage).

Затем создайте дерево записей шаблона, как показано ниже:



Не устанавливайте для них флаги "Очистить"(Clear)! Просто скопируйте соответствующие описания объекта "temp.xxxx" из шаблона "NEW SITE" в эти шаблоны. Порядок в этом списке также несущественен.

Следующим шагом будет включение этих трех шаблонов в основной шаблон "NEW SITE". Итак, редактируем шаблон "NEW SITE":



Порядок очень важен: первый показанный шаблон будет включен первым и, так как запись шаблона "ext: Main TEMPLATE cObject" создается копированием объекта "temp.menu_1", она должна включаться последней.

Сохраните запись шаблона и посмотрите новую структуру в Анализаторе Шаблона (Template Analyzer):

TEMPLATE HIERARCHY:

Title	Rootlevel	C. Setup	C. Const	PID/RL	NL
EXT:css_styled_content/static/					
ext: Auto-parser plugin					14
ext: Menu 1					14
ext: Main TEMPLATE cObject					14
NEW SITE	x	x	x		1 (0)

Linenumbers Comments Crop lines

CONSTANTS:

ext: Auto-parser plugin
[GLOBAL]

SETUP:

```

ext: Auto-parser plugin
[GLOBAL]
plugin.tx_automaketemplate_pi1 {
    content = FILE
    content.file = fileadmin/template/main/template_1.html

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

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

        TD.all = 1
    }

    relPathPrefix = fileadmin/template/main/
    
```

Как видите, первым включен статический шаблон, затем три "базовых шаблона" из нашего каталога хранения и, наконец, запись основного шаблона. Можно, кликнув по названию шаблона, ознакомиться с его содержанием.

Некоторые предложения по дизайну HTML

Как Вы могли увидеть в этом разделе руководства, получение шаблона HTML от графического дизайнера возможно в самой удобной форме, скрывающей полную *обратную совместимость* с его основной работой (так как нет необходимости вставлять вручную маркеры в его шаблон HTML). В дальнейшем будут включены только привычные технологии, вроде HTML и CSS, без создания сбивающих с толку таблиц стилей XLST (хотя XSLT, в качестве средства отображения, может подключиться на уровне PHP Бенуа, если захочет).

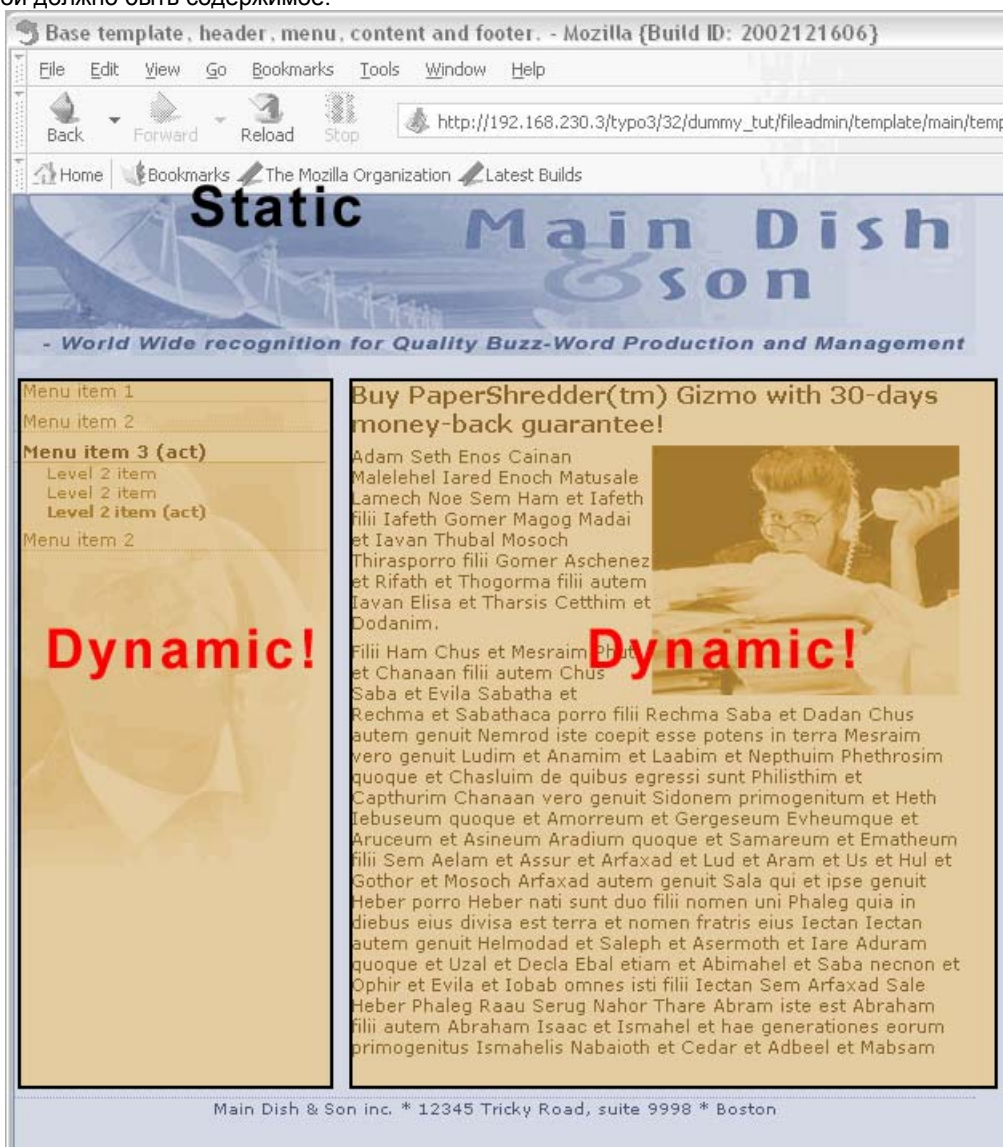
Но несмотря на свободу в дизайне, Рафаэль все еще нуждается в уяснении основ динамического и статического содержания.

Разберитесь, какие части динамические, а какие – статические

Рафаэлю, а возможно и всей веб-команде, нужно понять, какие части страницы останутся статическими, а какие динамическими. Статические части это все то, что НЕ обрабатываются ТУРОЗ. А вот динамические части, предположительно будут заменяться на содержимое данных в ТУРОЗ.

После определения динамических областей, Рафаэлю нужно удостовериться в двух вещах:

- Каждая часть должна быть размещена внутри простого элемента HTML. Это может быть элемент <div>, элемент <td>, элемент .
- Эти элементы должны содержать либо id, либо атрибут класса, на поиск и заключение в маркеры под частей которых настроен модуль Анализатора Шаблона (Template Auto-parser). Например, <td id="content"> для ячейки таблицы, в которой должно быть содержимое.



Безусловно ПРАВИЛЬНО, чтобы макет содержимого оставался в файле шаблона – по завершению он просто будет заменен на динамические эквиваленты.

Упрощайте разметку, пользуйтесь CSS таблицами стилей

Мы живем уже в 21 веке и самое время попрощаться с тегом , атрибутами bgcolor и поместить их все во внешнюю таблицу стилей. Такой подход реализации CMS максимально использует внешние технологии, и употребляется только необходимое количество специфичных технологий CMS (вроде записей шаблонов TypoScript).

Здесь лучшим примером является двухуровневое меню:

```
<!-- Таблица меню(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>
```

Давайте просто разберем здесь один элемент (красный наверху). Его можно представить чем-то вроде этого:

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

Теперь запомним, что наш объект TMENUITEM внутри записи шаблона был определен для помещения элементов в простые секции <div>:

```
temp.menu_1.1 {
  # Нормальное состояние свойств
  NO.allWrap = <div class="menu1-level1-no"> | </div>
  ...
}
```

... и, конечно, Вы уже догадались; в приведенном выше представлении нам необходимо заключать элементы как-то вот так:

```
temp.menu_1.1 {
  # Нормальное состояние свойств
  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></td>
    </tr>
  </table>
)
  ...
}
```

И этого все еще недостаточно: необходимо также добавить имя класса тега <A> для другого свойства состояния объекта NO и необходимо предусмотреть правильный префикс пути для файла "gray_dotted_line.gif" – что-то вроде "fileadmin/template/main".

Не используйте атрибуты класса для динамических ссылок

Многие дизайнеры могут считать лучшим способом выделения элементов меню и других ссылок использование атрибутов класса тега <A>. Например:

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

Тем не менее, это не выход, так как тег <A> *автоматически* генерируется TYPO3! Поэтому, для нашего TMENUITEM необходимы дополнительные свойства TypoScript:

```
temp.menu_1.1 {
  # Нормальное состояние свойств
  NO.allWrap = <div> |</div>
  NO.ATagParams = class="menu1-level1-no"
  ...
}
```

Добавляется больше сложных свойств TypoScript и меньше гибкости в таблице стилей. Например, если Вам необходимо управлять внешним видом ссылки из CSS, в этом случае Вы можете использовать

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

... но Вы не можете управлять вхождением элементов <div>, так как они *не* связаны ни с каким из классов или id! Если бы Вы придерживались нормального подхода, для которого элемент <a> не имеет атрибута класса, а элемент <div> имеет, тогда Вы сможете управлять *как* слоем <div>, так *и* всеми внутренними ссылками:

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

Держитесь подальше от row/colspans!

Обычно, убийца номер один динамического содержимого – использование атрибута colspan при его создании! В этом нет никаких сомнений. Например, обсудим это альтернативное представление меню:

```
<tr>

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

    <!-- Таблица области страничного содержимого: -->
    <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>
```

Это вообще не возможно представить! Первый пункт меню находится в той же строке таблицы, что и ячейка с содержимым, остальные шесть пунктов расположены в отдельных строках таблицы. Далее, ячейка таблицы с содержимым потребует атрибут colspan="7", поэтому займет все строки предназначенные для меню!

Это действительно очень плохой дизайн с технической точки зрения, потому что:

1. разметка меню разбросана по разным местам в коде HTML
2. существует зависимость между атрибутом colspan и количеством пунктов меню (динамических!)

Поэтому дизайнерам следует воздерживаться от такого подхода. Вместо этого, дизайнеры вроде Рафаэля должны научиться создавать разметку для динамических элементов, вроде меню, в такой форме, которую *легко повторить*. Вот Вам хорошие примеры:

Пример из нашего текущего шаблона:

```
<!-- Таблица меню: -->
<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>
```

Альтернативное представление с помощью таблицы:

```
<!-- Таблица меню: -->
<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>
```

Примеры могут быть и более сложными, вроде рассмотренных ранее, в которых обрабатывался *каждый* элемент таблицы! Цель – не количество кода HTML на пункт, а *повторяемость разметки*!

Если Рафаэль поймет эти соображения, Бенуа не будет терять время на представление дефектной структуры HTML, а, со временем, Писака будет счастлив от такой скорости разработки, за счет органичной кооперации Рафаэля и Бенуа, и в благодарность купит им что-нибудь перекусить!

Последнее замечание

У этого руководства есть [Часть 2](#) и [Часть 3 в другом документе](#). Эти части, особенно Часть 2, продолжают эту (Часть 1), в основе все та же история веб команды Рафаэль, Бенуа и Писаки. Но перед продолжением, Вам может понадобиться дополнительный опыт для закрепления материала! Поэтому сейчас, было бы хорошо закрепить полученные знания, почитать другие руководства и перебраться на уровень выше, особенно в плане техники и разработки.