Moderne Templateerstellung, Teil 1 (MTB_de/1)

Extension key: **doc_tut_templselect_de**

Inhaltsverzeichnis

Einführung

Worum geht es?

Diese Anleitung zeigt Ihnen von Grund auf, wie Sie mit dem Content Management System TYPO3 eine Website basierend auf einer HTML Vorlage (genannt Template) erstellen.

Für Entwickler auf Anfängerniveau.

Die Website, die Sie erstellen werden, sieht so aus:

Die Seite enthält einen Bereich für das dynamische Menü (links) und den dynamischen Seiteninhalt (rechts) - alles Übrige ist statischer Inhalt des HTML-Templates.

Ziel

Ziel dieser Anleitung ist es, Ihnen das Rüstzeug zu vermitteln um mit modernen Techniken TYPO3 basierte Websites zu erstellen. Es vermittelt Ihnen schnell praktische Erfahrungen mit dem System und ein Gesamtverständnis aller Faktoren für die Websiteerstellung mit TYPO3.
Wenn Sie den Eindruck haben, diese Anleitung sei zu lang und sich etwas kürzeres wünschen, dann sollten Sie sich vielleicht ein anderes CMS suchen. Ein leistungsfähiges und komplexes Werkzeug wie TYPO3 kann auf weniger Seiten nicht beschrieben werden. Und auch diese Anleitung kann nur einen ersten Eindruck vermitteln.

Auch wenn TYPO3 kostenlos ist, heißt das nicht, dass der Entwickler sich keine Zeit zum Kennenlernen nehmen muss. Es erfordert genau soviel Zeit und Einsatz wie jede kommerzielle Alternative auf gleichem Niveau. Seien Sie also vorgewarnt: einen Düsenjet fliegen zu können verlangt verschiedenste Fähigkeiten und folglich auch Zeit zum Erlernen. Mit dieser Anleitung werden sie aber hoffentlich so schnell wie möglich auf das Rollfeld gebracht werden um starten und bald abheben zu können.

Gliederung

Die Anleitung besteht aus vier Teilen:

Die Grundlagen: - eine Anleitung für Anfänger zum Erstellen von Websites mit TYPO3-Standardvorlagen, TypoScript und Content Objects (cObjects, also den Inhaltsobjekten). Jeder, der mit TYPO3 entwickeln möchte, sollte mit den hier beschriebenen Konzepten vertraut sein.

Teil 1: Integration eines HTML-Templates - dieser Teilt richtet sich speziell an beteiligte HTML-Webdesigner mit einem begrenzten technischen Wissen.

Teil 2: Erstellen einer Vorlagenauswahl - dieser Teil richtet sich an Webdesigner mit guten Kenntnissen von PHP, SQL und allgemeinen Programmierkonzepten.

Teil 3: Erweitern des integrierten Zugriffschemas: - für erfahrene TYPO3/PHP-Entwickler.

Hinweis: Teil 2 und Teil 3 finden Sie in einem weiterem Dokument in der extension (Erweiterung)"doc_tut_templselect2"

Sie müssen diese Anleitung nicht komplett durcharbeiten sondern können direkt zu zu dem für sie relevanten Thema springen. Wenn Sie die Anleitung jedoch komplett durcharbeiten, werden Sie Abschnitte finden, die aufeinander aufbauen und dadurch Schritt für Schritt Ihre Fähigkeiten verbessern. Das Verständnis der Inhalte beim Springen zwischen verschiedenen Anleitungen kann möglicherweise verschiedenes Grundwissen und Erfahrungen vorraussetzen

Die Extension

Alle Dateien dieser Anleitung finden Sie in ein einer TYPO3-Extension (Erweiterung). Normalerweise enthalten Extensions Skripte und Hilfsmittel, die die Möglichkeiten von TYPO3 erweitern. Diese Extension hat allerdings keine Auswirkungen auf die Kernfunktionalität von TYPO3 sondern wird nur dazu benutzt, die Dateien dieser Anleitung auf Ihren Server zu bringen und um dieses Dokument online auf typo3.org zu präsentieren.

Um also mit diesem Dokument arbeiten zu können, installieren Sie zuerst das das dummy-package (siehe Grundlagenteil) und importieren Sie dann die extension “doc_tut_templselect“ vom TER (TYPO3 Extension Repository) mit dem Erweiterungsmanager. Dann haben Sie alle Dateien auf Ihrem Server zur Verfügung wenn Sie sie benötigen.

Dieses Dokument kann entweder online gelesen werden oder Sie können es als SXW-Datei (OpenOffice-Dokument) von typo3.org downloaden.

Kosten

Um dieses Dokument lesen zu können müssen Sie nichts bezahlen. Es hat jedoch den Autor, Kasper Skårhøj, eine komplette Woche gekostet es vorzubereiten, zu schreiben und zu vervollständigen.
Wenn diese Anleitung für Sie oder Ihr Unternehmen hilfreich ist und Ihnen dabei hilft, leistungsfähige Websites für Ihre Kunden zu erstellen, erwägen Sie bitte die Möglichkeit einer Aufmerksamkeit. Reines Schulterklopfen ernährt niemanden und möglicherweise könnte diese Anleitung dann die letzte gewesen sein...

Die Grundlagen

Einleitung

Dieser Abschnitt wird den Anfänger schnell damit vertraut machen, wie das TYPO3 Frontend (also die Website, die TYPO3 erzeugt) funktioniert. Es erklärt was TypoScript Templates, Content Objects und HTML-Templates sind. Wenn Sie diese Grundlagen schon kennen, können Sie schon zum nächsten Abschnitt übergehen. Auf jeden Fall sollten Sie das dummy package installieren und einige Seiten darin erstellen. Dies wird im folgenden besprochen:

Zielgruppe

Alle Entwickler, für die TYPO3 neu ist, unabhängig von ihren sonstigen Erfahrungen.

Einige Teile beziehen sich auf Technologien wie SQL, HTML, CSS und PHP. Der Inhalt ist im allgemeinen technischer Art, da er Sie durch die Installation des ”dummy”-Paketes führt und die Grundlagen der Templatefunktionaltät in TYPO3 erklärt.

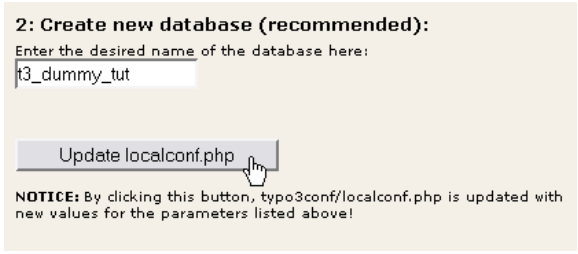
Los geht's - Installation des ”dummy”-Pakets

Um schnell loslegen zu können werden wir ein neues Dateiframework und eine leere Datenbank für TYPO3 installieren. Am einfachsten geht das mit dem ”dummy“-Paket. Ein Paket oder ”package” enthält den allgemeinen TYPO3 Quellcode sowie die lokal benötigten Dateien und ggf. Datenbankinhalte - und somit die komplette Website. Im Falle des dummy-Pakets handelt es sich um eine leere Website ohne Inhalt. Aber dies ist für eine neues Projekt sehr nützlich.

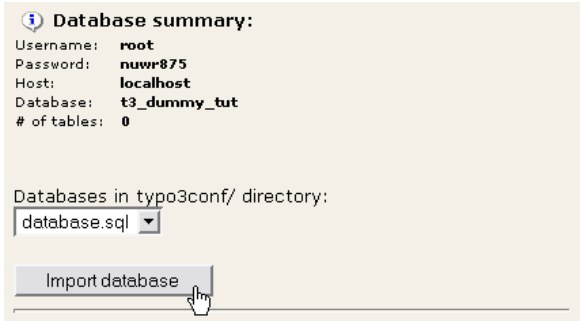
Also laden Sie sich das “dummy”-Paket herunter und installieren Sie es auf Ihrem Webserver. Nach dem Entpacken rufen Sie einfach die Datei index.php auf und Sie werden durch folgende Schritte geführt:

Eingabe der Datenbankinformationen:

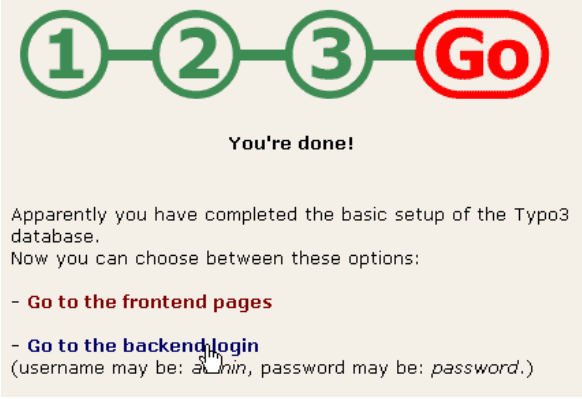
Erstellen einer neuen Tabelle:



Import der dummy Datenbank, "database.sql":



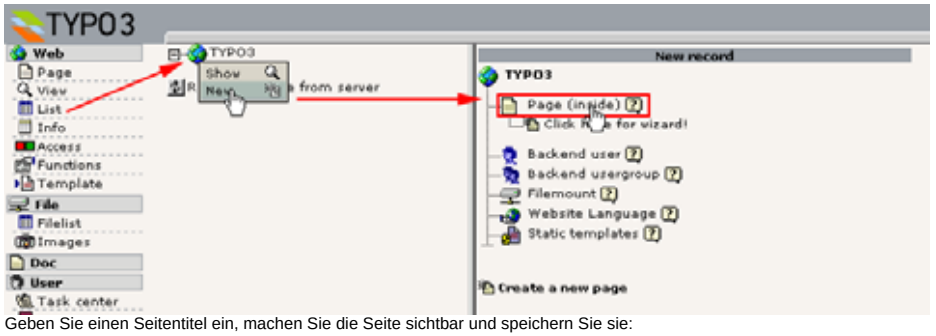
Das Backend aufrufen:



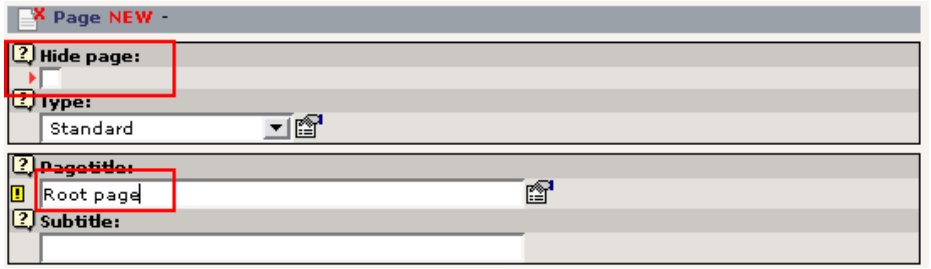
(Der Begriff Backend bezeichnet Anwendungen die auf einem Server ausgeführt werden, in diesem Fall das Administrationsinterface von Typo3.)
Um diese Schritte erfolgreich durchlaufen zu können muss der Webserver Benutzer für die Datei localconf.php Schreibrechte besitzen (Sie werden darauf hingewiesen, falls dies nicht der Fall sein sollte) und Sie müssen weiterhin den Benutzernamen und das Passwort des Datenbankbenutzers kennen.

Erstellen einer Seitenstruktur

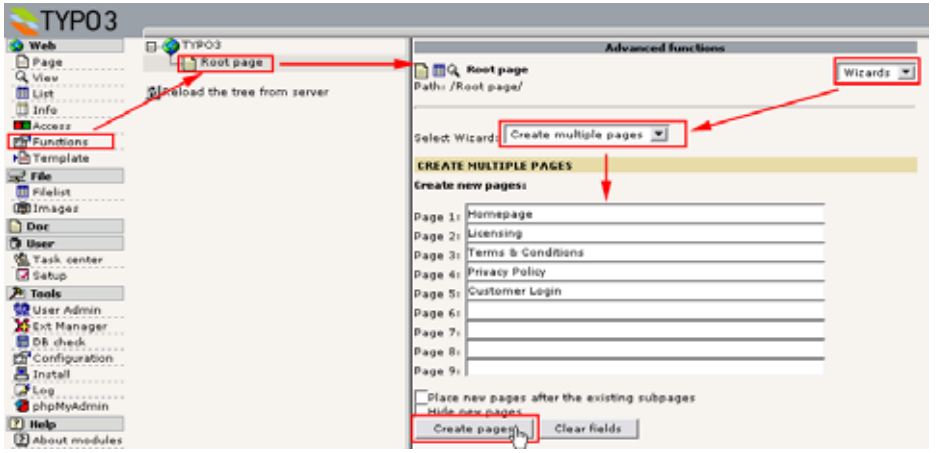
Das erste was jetzt zu tun ist, ist die Seitenstruktur festzulegen. Eine gute Seitenstruktur enthält Seiten auf mindestens zwei Ebenen. Auf jeden Fall müssen wir mit einer Startseite der untersten (root) Ebene beginnen.



Geben Sie einen Seitentitel ein, machen Sie die Seite sichtbar und speichern Sie sie:

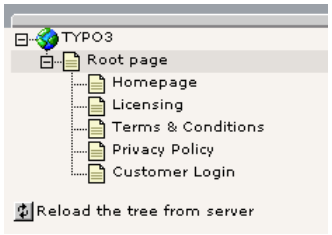


Jetzt könnten Sie so weiter machen und die gesamte Seitenstruktur erstellen. Es gibt es aber einen Assistenten, um diese Schritte viel einfacher durchzuführen. Wechseln Sie in das Modul: "Funktionen", wählen Sie den Assistenten "Erzeuge mehrere Seiten", geben Sie mehrere Seitentitel ein und klicken Sie anschließend auf die Schaltfläche "Seiten erstellen".

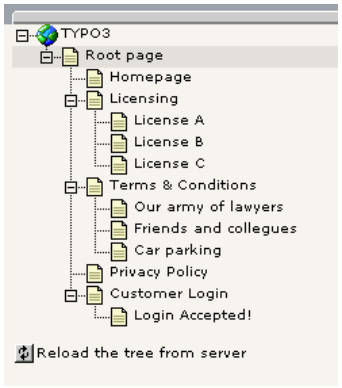


(Für diesen Assistenten muss die Extension "Create multiple pages" installiert sein.)

Als Ergebnis sollten Sie einen solchen Seitenbaum erhalten:

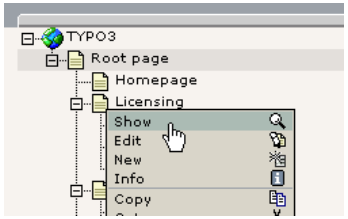


Jetzt erstellen Sie weitere Seiten auf der nächsten Ebene, das Ergebnis sollte dann so aussehen:



Seitenbaum und Template Datensätze (template records) erklärt

Klicken Sie auf das Icon einer Seite und wählen Sie dann "Anzeigen" aus:



Das Ergebnis der Ansicht der Seite "Licensing" sieht wie folgt aus:

Die Seite "Licensing" hat die uid "5" und zum Betrachten des Seiteninhalts muss der Parameter "id=5" an das index.php Skript des Frontends übergeben werden.

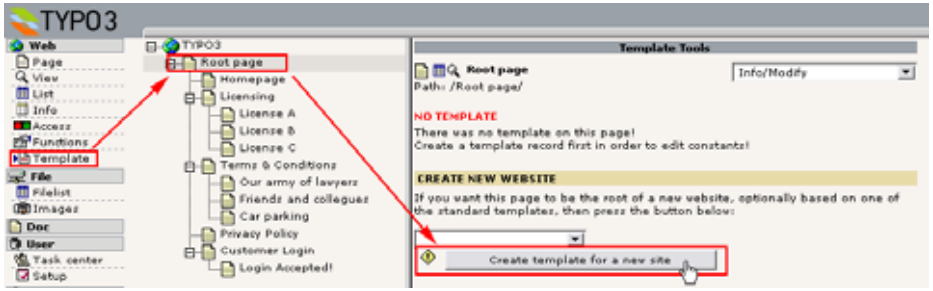
Die Ausgabe besagt nun aber, dass kein Template gefunden wurde. Das heißt, dass bei der Erzeugung der Seite kein Template gefunden wurde - ausgehend von der untersten Ebene des Seitenbaums für die Seite mit der id 5. Zum besseren Verständnis hier etwas Theorie zur Erklärung:

TYPO3 ist ein CMS für mehrere Websites. Das bedeutet, dass Sie in einer Seitenstruktur unendlich viele Websites haben können. Jede Site in der Seitenstruktur muss eine Startseite haben, die dadurch gekennzeichnet ist, dass Sie ein Template enthält. Dieses Template muss als "Rootlevel" gekennzeichnet sein. Wenn also ein Template mit dieser "Rootlevel"-Kennzeichnung in einer Seite enthalten ist, heißt das, dass "ab jetzt eine neue Website innerhalb der Seitenstruktur beginnt".

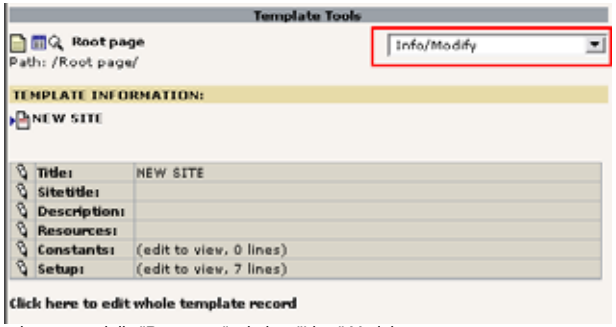
Die "root line" (Ursprungspunkt) ist ein Synonym für "Pfad". Man kann zum Beispiel auch sagen, dass die Seite "License A" in der Seitenstruktur den Pfad "TYPO3 > Root page > Licensing > License A" hat.
Wenn wir die id von "License A" an das Skript index.php übergeben, wird bei der Erzeugung der Seite, ausgehend von der Position von "License A" rückwärts bis zum Ursprungspunkt des Seitenbaumes (dem "page tree root") nach einem Template Datensatz gesucht, der die Darstellung der Seite definiert.

Erstellen eines Template Datensatzes (template record)

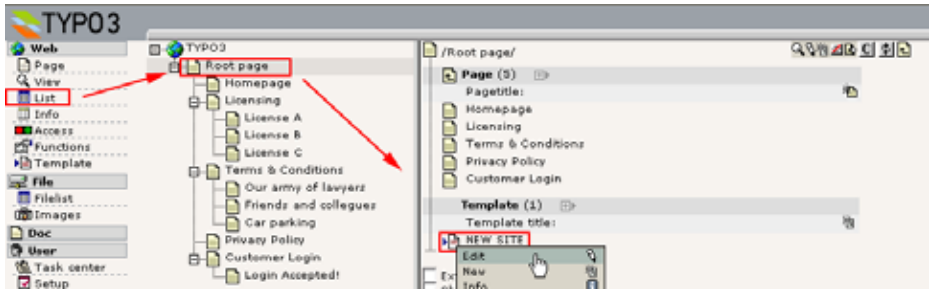
Was wir also machen müssen ist einen Datensatz für ein Template auf der "Root page" zu erstellen. Der einfachste Weg dafür ist das "Template" Modul zu benutzen.



Die nächste Abbildung zeigt Ihnen diese Ansicht des "Template" Moduls, das Sie später zur Bearbeitung der Vorlage verwenden werden:



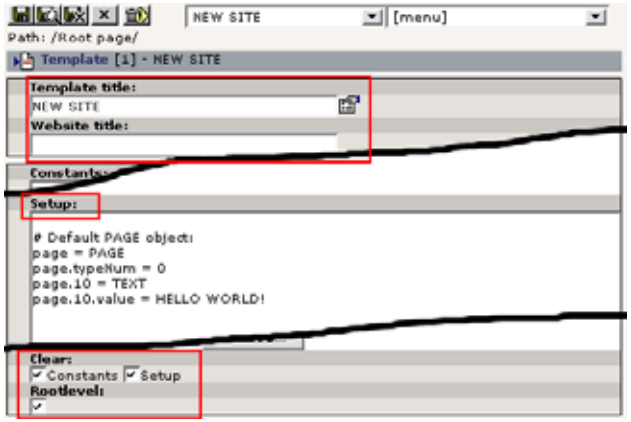
Jetzt schauen wir uns aber erst mal die "Root page" mit dem "Liste" Modul an:



Wie Sie sehen gibt es dort ein "Template" auf der "Root page".

Diese Vorlage enthält die *grundlegenden Anweisungen* für die Generierung der Website durch die Frontend Engine ab diesem Punkt, welche Funktionen sie enthält, wie sie konfiguriert ist usw. Ein Template wird *immer* benötigt um mit einer neuen Website in TYPO3 zu beginnen (unter Verwendung der Standard Frontend Engine). Sie können selbstverständlich individuell festlegen, wie viel von Ihrem Webdesign das Template direkt kontrolliert; Sie können eine komplette Website in einem Template programmieren. Sie können das Template aber auch anweisen eine selbst programmierte PHP-Funktion aufzurufen, die Ihre Seite rendert und den gesamten Seiteninhalt liefert (basierend auf XML, XSLT oder anderen Datenbanken - wie immer Sie mögen. Sie können ebenso einen Mittelweg beschreiten, der das beste beider Welten vereinigt. Und genau das ist es, was wir hier beschreiben.

Schauen wir uns den Inhalt des Templates an. Dort gibt es drei Felder von besonderem Interesse:



Der "Template title" ist nur der Titel des Templates und wird sonst nirgendwo angezeigt.

Der "Website title" wird den Seitentiteln vorangestellt, so wie zwischen den <title> Tags jeder herkömmlichen Seite.

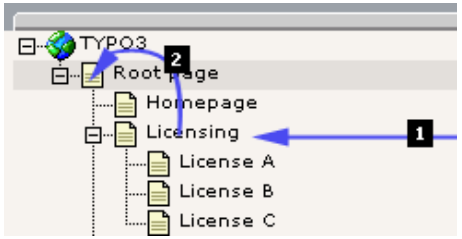
Im "Setup" Feld fügen Sie die *Anweisungen* ein, wie die Seite auf verschiedene Weisen dargestellt werden soll. Diese Anweisungen werden in einer hierarchischen Informationsstrukur eingebunden, die von der TypoScript Syntax vorgegeben wird. Da die Anweisungen des Feldes "Setup" durch die TypoScript-Syntax definiert sind, werden Templates häufig als "TypoScript Templates" bezeichnet.

Die Einstellungen "Clear constants" und "Clear setup" bedeuten, dass kein TypoScript Konfigurationscode, falls vorhanden, von den Templates höher im Baum (näher an der Wurzel) übernommen wird. Diese Checkboxen sollten für die Templates angekreuzt werden, welche als Basistemplate dienen sollen ("Rootlevel").

Die "Rootlevel" Checkbox sagt nur aus: "diese Vorlage kennzeichnet den Beginn einer neuen Website".

Nun zurück zur Ansicht der der Seite "Licensing":

Jetzt erscheint etwas anderes. Folgendes ist passiert:

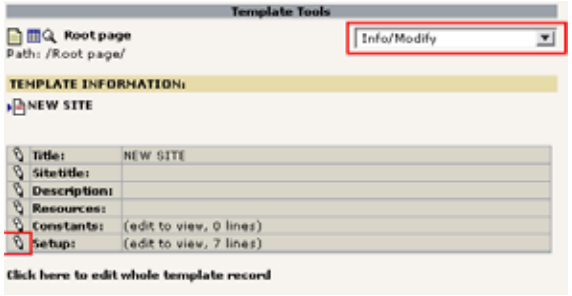


- Anfrage der Seite mit der uid "5" (1)
- Auswahl des ersten Template Datensatzes für diese Seite (pid=5); nicht gefunden!
- Da für die Seite mit der uid "5" kein Template gefunden wurde, wird die vorherige Seite in der Hierarchie überprüft (2)
- Die "Root page" (id=1) hat ein Template. Also wurde ein Template gefunden!
- Da ein Template für die Seite mit der uid "1" gefunden wurde, wird überprüft, ob "Rootlevel" gesetzt ist ...
- Ja "Rootlevel" ist gesetzt. Gut, das ist der Anfang der Website, also ...
  - Parsen (Analyse und Verarbeitung) der TypoScript Konfiguration im Feld "Setup".
  - Führe die Anweisungen im Feld "Setup" aus.

Grundlagen der Anweisungen im Feld "Setup"

Wenn ein Template Datensatz vorhanden ist, liegt das Hauptinteresse beim Feld "Setup" dieses Templates. In dieses Feld tragen Sie einen Satz Anweisungen ein, die festlegen was geschieht, wenn wir eine Seite im Frontend aufrufen.

Die beste Möglichkeit dieses Feld zu bearbeiten ist im "Template" Modul mit der Ansicht "Info/Modify":



Klicken Sie auf das "Bearbeiten" Symbol und es geht los:

Hier eine kurze Erklärung:

Das PAGE Object (1)

Zuerst definieren wir eine neues Objekt auf oberster Ebene ("Toplevel Object" - TLO), "page" genannt. Objekttyp ist "PAGE" (sie können auch einen Frameset definieren; siehe TSref). Anstatt "page" könnten Sie auch andere Bezeichnungen wählen, mit Ausnahme reservierter TLOs, siehe TSref about Toplevel Objects.
Als Eigenschaft für das "page" Objekt definieren wir die vorgeschriebene Eigenschaft "typeNum" und setzen ihren Wert auf 0 (Null).

Das bedeutet jetzt, dass das Toplevel Objekt "page" die *Standardbehandlung* jedes Seitenaufrufs innerhalb eines Zweiges des Seitenbaums vorgibt, in dem unser Template den Beginn der Website kennzeichnet.

Nun wollen wir dem "page" Objekt Anweisungen geben, was es zu tun hat. Dazu müssen wir einen Blick auf die möglichen Eigenschaften für PAGE Objekte werfen. Einige Eigenschaften sind Metaeigenschaften wie die "Config-" oder "includeLibs" Eigenschaften. Andere dienen dazu, zu definieren, welche HTML-Ausgaben dieses PAGE Objekt erzeugen soll.

Der bedeutendste Satz der Eigenschaften für PAGE Objekte ist die numerische Liste der Inhaltsobjekte (content objects - cObjects). Jedes dieser Content Objects kann eine Zeichenkette mit HTML-Inhalt übertragen, der zusammen den Inhalt zwischen den <body> Tags der Seite bildet.

Das Content Object (2)

Im Beispiel oben ist ein sehr einfaches Content Object - TEXT - definiert, mit dem Index "10" innerhalb der numerischen Liste der Content Objects im PAGE Object "page". Dieses cObject hat die Eigenschaft "value", die auf "HELLO WORLD" gesetzt wurde. Dies hat zur Folge, dass dieser Text vom cObject zurückgegeben wird und zum Inhalt auf der Website zwischen den <body> Tags wird:

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

Die möglichen Eigenschaften für das TEXT cObject finden Sie hier.

TypoScript - keine Programmiersprache

Bevor wir fortfahren, beachten Sie bitte, dass TypoScript keine prozedurale Programmiersprache ist. Sie könnten leicht zu dieser Annahme kommen. Wenn Sie Erfahrung mit anderen Programmiersprachen haben und wissen wie sie arbeiten, kann dies ein gefährlicher Cocktail werden mit dem Ergebnis totaler Verwirrung. TypoScript *konfiguriert* tatsächlich nur das System, um in einer bestimmten Weise zu agieren.

Um zu wissen, was TypoScript wirklich ist, um den Unterschied zu verdeutlichen und zum Verständnis, wie es in TYPO3 eingesetzt wird, empfiehlt sich ein Blick in dieses Dokument, das die Grundlagen vermittelt.

Weitere Beispiele zu PAGE und cObjects

Dieser Abschnitt stellt Ihnen ein paar weitere Beispiele zu PAGE und cObjects vor, damit Sie das Konzept völlig erfassen. Wenn Sie dies verstehen, dann verstehen Sie auch "TypoScript Templates" wenn die Strukturen komplexer werden.

Zwei cObjects

Was passiert, wenn Sie ein weiteres Content Object einfügen:

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

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

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

(Übrigens, Zeilen die mit "#" oder "/" beginnen sind Kommentare.)

Ausgabe:

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

So wie erwartet werden mehrere Content Objects einfach aneinander gehängt.

Ändern der Reihenfolge

Was passiert, wenn die Reihenfolge der Definitionen geändert wird?

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

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

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

Ausgabe:

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

Das gleiche: die Reihenfolge der Ausgabe entspricht dem Index in der Liste.

Dynamischer Inhalt

Was ist aber nun, wenn wir wollen, dass das TEXT cObject den Seitentitel einfügt?

Schauen Sie dazu in die TSref um festzustellen, ob das TEXT cObject Eigenschaften besitzt, die es Ihnen erlauben, dieses zu realisieren. Sieht ganz so aus. Außer der "value" Eigenschaft hat das TEXT cObject auf der gleichen Ebene auch "stdWrap" Eigenschaften (klicken Sie hier), die benutzt werden können, um Daten dynamisch zu holen und zu verarbeiten.

Da die Übertragung des cObjects TEXT im Hauptbereich des PAGE Objekts erfolgt, wird der aktuelle Datensatz der Datensatz der aktuellen Seite. Wenn sich also die stdWrap Eigenschaft "field" auf einen Feldnamen bezieht, so wird der Inhalt vom aktuellen Seitendatensatz geholt. An dieser Stelle wird das Feld "title" den Seitennamen beinhalten (dieses erfordert selbstverständlich Wissen darüber, welche Felder in der Seitentabelle verfügbar sind!)

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

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

Ausgabe:

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

Da die aktuelle Seite die "Licensing" Seite war, war das auch der Titel bei der Ausgabe.

stdWrap Eigenschaften

Da wir nun mit den "stdWrap" Eigenschaften experimentieren, können wir den Wert mit weiterem Ausgabetext versehen und in den unterschiedlichen Weisen verarbeiten etc.:

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

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

(Beachten Sie bitte wie ich die Formatierung in TypoScript verändert habe, so dass die Eigenschaften in geschweifte Klammern gesetzt wurden. Diese Syntax von TypoScript macht es einfacher, viele Eigenschaften auf der gleichen Ebene zuzuweisen. Im Endergebnis macht es aber keinen Unterschied, alle Eigenschaften mit dem vollen Objektpfad "page.10" festzulegen.)

Ausgabe:

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

Beachten Sie auch, wie der Text den Seitentitel einschliesst und desweiteren in Großbuchstaben umgewandelt und auf acht Zeichen begrenzt wurde und wie das "..." angehängt wurde.

Die selbe Ausgabe hätten wir auch mit einem anderen cObject erreichen können, nämlich genannt "HTML" - der einzige Unterschied ist, dass alle stdWrap Eigenschaften zur "value" Eigenschaft des HTML cObjects gehören und nicht Eigenschaften des cObjects selbst sind:

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

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

Es ist nur eine Frage der persönlichen Vorlieben und Ihres Stils ob Sie TEXT oder HTML cObjects einsetzen.

Inhalt aus PHP Skripten

Wenn Sie möchten können Sie den Inhalt auch sehr einfach über PHP Skripte einfügen. Dies ist eine *sehr empfohlene* Vorgehensweise, wenn Sie eine programmgesteuerte (bedingte) Ausgaben benötigen, was TypoScript nicht kann, da die cObjects und ihre Eigenschaften keine prozedurale Programmierung erlauben. Sie erinnern sich, dass dies nur vorprogrammierte Objekte sind, die auf die gesetzten Eigenschaften reagieren, und nichts weiter.

Was Sie dazu brauchen ist das cObject USER.

Zuerst erstellen wir eine PHP -Datei in **fileadmin/userfunctions.php**:

```
<?php

class user_functions {

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

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

Dann konfigurieren wir ein cObject vom Typ USER, welches diese Funktion mit einem einfachen Parameter "factor" aufruft:

```
# 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
```

Und die Ausgabe sollte so aussehen:

Wie Sie sehen, haben wir einige "Metaeigenschaften" definiert, die nicht direkt mit der Inhaltsausgabe in Beziehung stehen:

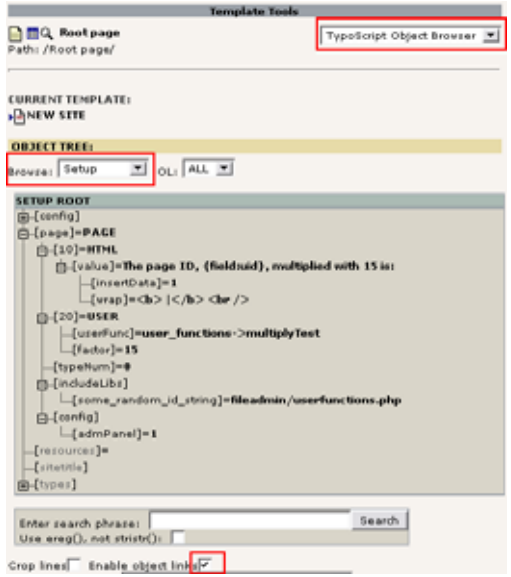
```
page.includeLibs.some_random_id_string = fileadmin/userfunctions.php
page.config.admPanel = 1
```

"includeLibs" ist eine Eigenschaft, die es erlaubt, eine Liste von einzubindenden PHP-Dateien zusammenzustellen (Klassen- und Funktionsbibilotheken!), die eingebunden werden, bevor die Seite erzeugt wird. "config" erlaubt eine Menge an Konfigurationsoptionen für das allgemeine Verhalten dieses PAGE Objekts. In diesem Falle wurde das "Admin Panel" am Seitenende eingefügt.

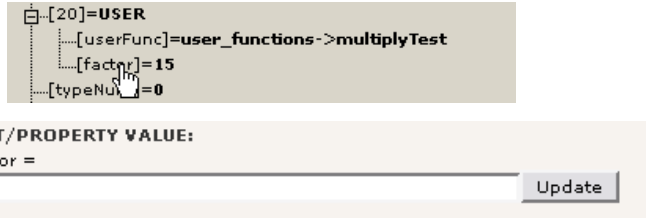
Für dieses Beispiel sollten Sie sich ein paar Minuten Zeit nehmen. Beachten Sie wie die Eigenschaft "factor" des Objekts USER in der PHP-Funktion verfügbar ist. Beachten Sie, wie sich die PHP-Funktion die Seiten-ID "holt". Beachten Sie, wie das "Admin Panel" eingebunden wurde. Sie können eine Menge davon lernen, wenn Sie mit diesem kleinen Beispiel spielen. Lassen Sie ihrer Phantasie freien Lauf, da Sie dadurch ein gutes Gefühl der Steuerung bekommen.

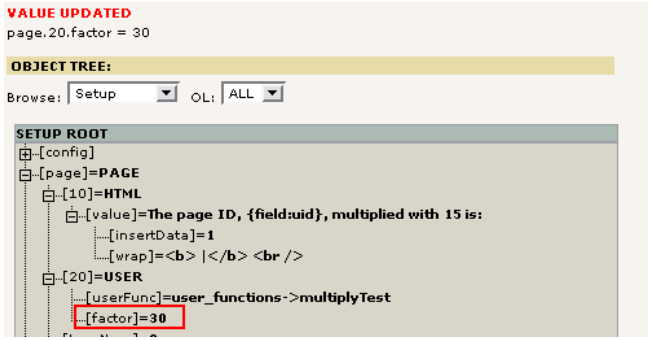
Der Object Browser

Der Object Browser im Template Modul ist ein unentbehrliches Werkzeug zum Überprüfen der hierarchischen Struktur Ihrer Template Datensätze.

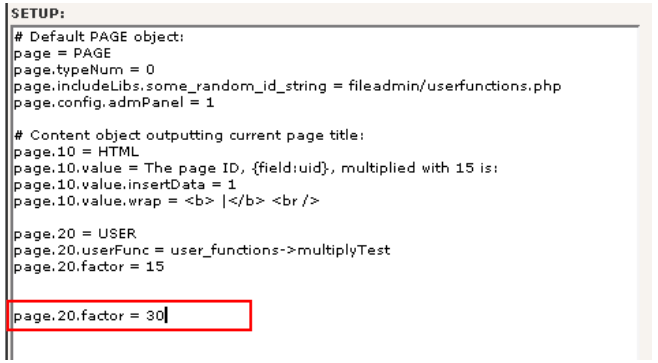


Wie Sie sehen können, werden die gelesenen TypoScript Werte in dieser wohl organisierten Baumstruktur ausgegeben. Somit können Sie überprüfen, ob jede Eigenschaft, die Sie eingestellt haben, sich an der jeweils korrekten Position befindet und es ist Ihnen sogar erlaubt, einen einzelnen Wert zu bearbeiten, wenn Sie sicherstellen, dass die Option "Enable object links" angewählt ist.





Wenn Sie zu "Info/Modify" zurückgehen sehen Sie, dass der Inhalt des Feldes Template Setup verändert wurde.



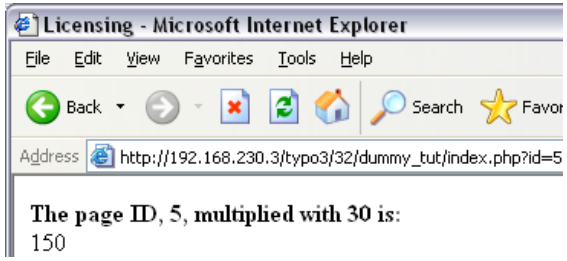
Der Object Browser ist nicht intelligent genug die Zeile mit "page.20.factor = 15" abzuändern. Er fügt einfach eine neue Zeile am Ende ein, die den Inhalt des vorher gesetzten Wertes überschreibt. Sie können dies manuell bereinigen. Daraus ergibt sich, dass der Object Browser eine sichere Möglichkeit für zur Einstellung dieser Werte bietet, da immer der richtige Objektpfad genutzt wird.

Eine Bedingung ausprobieren

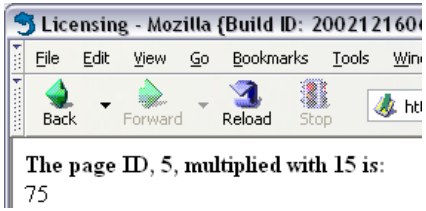
Wir könnten diese Situation tatsächlich nutzen, um die Anwendung von "Bedingungen" zu demonstrieren. Eine sehr einfache Bedingung soll soll überprüfen, welcher Webbrowser verwendet wird:

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

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

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

[browser = msie]

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

[global]
```

Im Microsoft Internet Explorer sieht das dann so aus:



In allen anderen Browsern so:



Nochmals zu PAGE Objekten

OK, dies soll kein allgemeiner Kurs sein um Templates nur mit cObjects zu gestalten - in diesem Handbuch benutzen wir hauptsächlich TEMPLATE, USER und HMENU Objekte und kombinieren sie mit vordefiniertem TypoScript Code in statischen Templates um unsere Ziele zu erreichen. Die eigentliche Beschreibung von Objekten und ihren Eigenschaften finden Sie im TSref. Wenn Sie Beispiele für die Benutzung dieser Objekte suchen, schauen Sie bitte in TypoScript by Example.

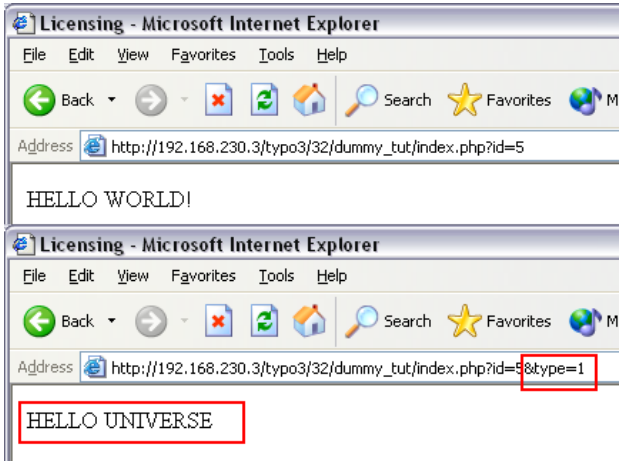
Mit diesen zwei Punkten möchte ich dann im Übrigen zum Ende kommen:

Die Relevanz von "&type="

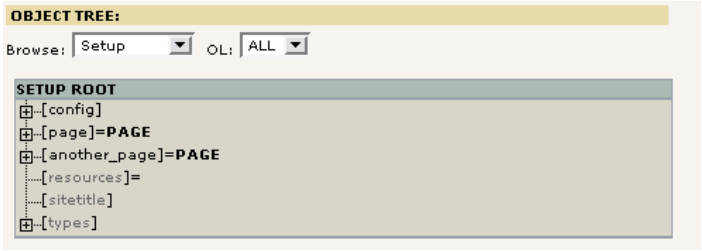
Im folgenden Beispiel ist nicht nur ein PAGE Objekt definiert. Desweiteren ist noch das TLO "another_page" als PAGE Objekt definiert, aber "typeNum" ist auf "1" gesetzt. Das bedeutet, dass die PAGE "page" als Standardbearbeitung von Seitenanfragen definiert ist, während "another_page" jene Seiten betrifft, welche mit dem Parameter "&type=1" zusätzlich zum id-Parameter aufgerufen werden.

```
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
```

Die Ausgabe ist dann:



Hier der Objektbaum:



Kopieren von Objekten

Es kann schnell sehr praktisch werden, den TypoScript Konfigurationscode Ihres Templates in Einheiten verschiedener Bereiche/Funktionen aufzuteilen um diese dann am Ende des Templates zusammenzufügen, indem sie an die richtige Positionen kopiert werden. Dies schließt ggf. ein Aufspalten der Einheiten in eine Hierarchie von eingebundenen Template Datensätzen ein, die somit eine Wiederverwendung von TypoScript zulässt. Darüber erfahren Sie später mehr.

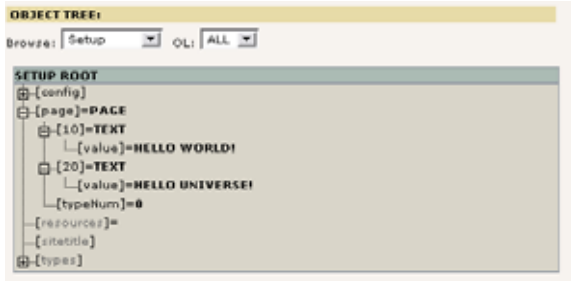
Die Haupteigenschaft von TypoScript, die dieses erlaubt, ist die Syntax zum Kopieren der Objekte von einem Zweig des Objektbaums in einen anderen. Dazu dieses Beispiel:

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


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


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

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

Das Ergebnis ist - selbstverständlich - dieses:

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

Wenn Sie in den Object Browser schauen, werden Sie bemerken, dass die Objekte von "temp. ..." in die Pfade "page.10" und "page.20" kopiert wurden. Wir könnten natürlich erwarten, dass die TLOs von "temp" ebenfalls im Baum zu sehen wären, aber das ist nicht der Fall. Der Grund hierfür ist, dass die TLOs "temp." und "styles." *gelöscht* werden, da sie während des Parsens in einem *temporären Objekt-Bereich* verfügbar sind, danach aber nicht mehr erreichbar sind.

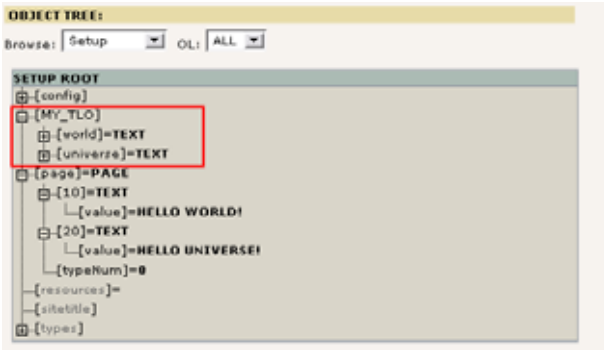


Wenn wir unsere Objekte innerhalb des Baums sehen möchten, können wir ein anderes TLO benutzen:

```
# Make temporary version of the first cObject:
```

```
MY_TLO.world = TEXT
MY_TLO.world.value = HELLO WORLD!


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


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

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

... und das TLO "MY_TLO" ist noch vorhanden

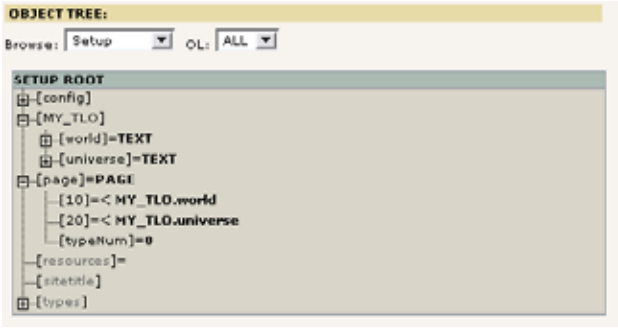


Referenzen - keine Kopien

Dies bedeutet auch, dass wir eine Referenz auf die Objekte in MY_TLO.* erstellen können, da für Referenzen auf cObjects erforderlich ist, dass sie in der Struktur auch noch nach dem Parsen vorhanden sind (im Gegensatz zu den "temp" und "styles" TLOs). Das Skript kann dazu wie folgt umgeschrieben werden:

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


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


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

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

Es sieht nur wie eine kleine Veränderung aus, hat aber eine immense Auswirkungen, da Objekte NICHT kopiert werden, sondern referenziert.



Aus dieser Methode ergeben sich für uns einige praktische Auswirkungen, aber auch Beeinträchtigungen. Das überzeugende Argument für Referenzen ist jedoch, dass ein cObject, welches in einem Baum definiert wird, mehrfach innerhalb einer Struktur ohne weiteren Speicherverbrauch für Duplikate genutzt werden kann. Ebenfalls schwierig zu handhaben wäre auch das Überschreiben der Eigenschaften aller Duplikate in einem Baum. Ein Beispiel zu diesem Problem finden Sie am Ende von Teil 2 in diesem Tutorial.

Hinweis: Referenzen wie diese sind *nicht* Teil der eigentlichen TypoScript-Syntax. Es handelt sich hierbei um eine Möglichkeit Eigenschaft, die intern von den cObjects bereitgestellt wird. Folglich können Referenzen nur bezogen auf cObjects genutzt werden; in Verbindung mit *keinem* anderen Objekttyp oder und *keiner* anderen Objekteigenschaft können Referenzen eingesetzt werden (es sei denn, dies wird ausdrücklich angemerkt).

Löschen des Cache (Zwischenspeichers)

Abschliessend sollten Sie noch wissen, dass wenn Sie Änderungen an einem Template Datensatz vorgenommen haben, der gesamte Zwischenspeicher (cache) gelöscht wird. Dies ist erforderlich da während des Parsens eines TypoScripts durch Typo3 die als Ergebnis zurückgelieferte Struktur im Cache gespeichert wird, so dass sie beim nächsten Seitenaufbau erneut "hervorgeholt" werden kann. Jedoch bedeutet dies auch, dass Änderungen, die direkt im List-Modul an den Templates vorgenommen werden, den Cache *nicht* leeren - das müssen Sie manuell tun, über den Link unterhalb des Menüs im linken Frame.

Bevor Sie also ein merkwürdiges Verhalten als Bug melden, vergewissern Sie sich grundsätzlich, dass es nicht an immer noch zwischengespeicherten, inzwischen veränderten Templates oder Extensions liegt.

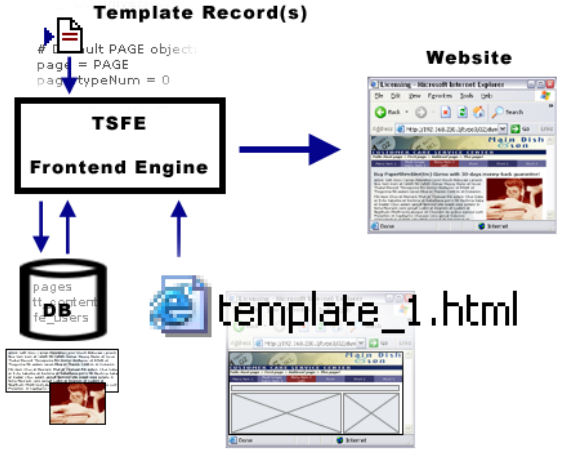
Teil 1: Integration einer HTML Vorlage

Implementierung eines CMS

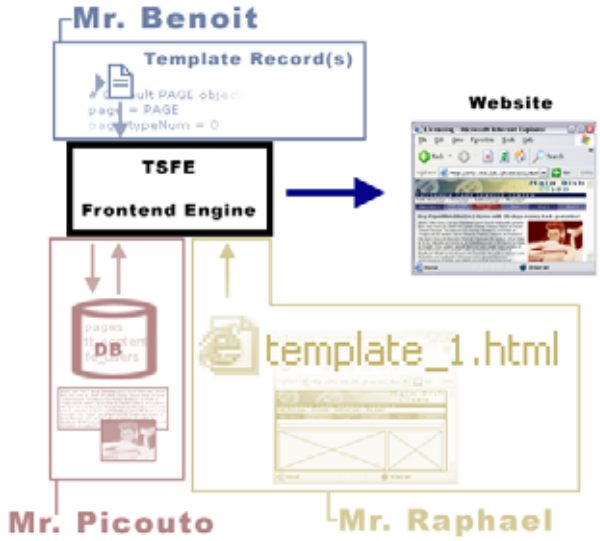
Die Erstellung einer Website mit einem CMS erfordert die Einbeziehung von mehr Komponenten, als wenn die Site nur mit reinem statischem HTML erstellt würde. Die Idee hinter der Verwendung eines CMS ist den eigentlichen Inhalt getrennt von den Definitionen zu dessen Darstellung zu speichern. Einige CMS speichern den Inhalt in XML Dateien, andere nutzen relationale Datenbanken. Die Unterschiede haben verschiedene Für und Wider, aber die Idee dahinter ist die selbe: "Der Inhalt wird von den Farben getrennt".

Wenn TYPO3 Webseiten generiert, wird unformatierter Inhalt aus einer Datenquelle (Datenbank) mit den Formatanweisungen aus einem HTML-Template kombiniert. In diesem Prozess dient das Template als *Kontrollelement*, das festlegt in welcher Form Typo3 diese Kombination durchführen soll.

In der nächsten Abbildung sehen Sie, wie der Template Datensatz als Steuerelement - also sozusagen als "Programm" hinter der Erstellung des Frontends fungiert: schrittweise wird der Inhalt in der Datenbank gefunden, das Template gelesen, der Inhalt an den dafür vorgesehenen Platzhaltern eingefügt und eine ansehnliche Webseite wird ausgegeben!



In den meisten Webagenturen arbeiten bei der Erstellung einer Website eine ganze Anzahl von Leuten zusammen. In dieser Gruppe finden wir einen Grafikdesigner, einen Programmierer und einen Texter (zumindest für den ursprünglichen Inhalt zum Beginn). Jede dieser Personen hat unterschiedliche Kenntnisse und arbeitet folglich bei der Produktion der Site in unterschiedlichen Bereichen:



Wie in obigen Abbildung wird jeder Person von den verschiedenen Komponenten der vom CMS erzeugten Wesite eine zugewiesen:

- **Mr. Raphael** ist der Künstler. Er versteht viel von visueller Darstellung, er kennt Photoshop, Dreamweaver, CSS, HTML usw. Raphael jongliert mit Flashfilmen, hat aber keine Ahnung von PHP, TypoScript, SQL und anderen technischen Fragen. Also erstellt Raphael die HTML-Templates für uns!
- **Mr. Benoit** ist ein Entwickler. Er liebt Bits und Bytes, er mag Compilerananweisungen, reguläre Ausdrücke, Logik, PHP, SQL - und bald wird er auch TypoScript lieben. Wie jeder andere gute Programmierer glaubt er mehr an gute Blue Jeans als an extravagantes Design. Farben und Benutzbarkeit zu kombinieren ist nicht in seine Wiege gelegt worden. Daher ist Benoit für die Konfiguration von TypoScript in den Templates verantwortlich.
- **Mr. Picouto** erstellt die Inhalte. Er versteht was von Marketing. Die Arbeit von Raphael und Benoit bewundert er, da sie in seinen Augen wahre Wunder vollbringen. Er selbst ist weder ein Designer noch ein Programmierer, aber er hat die Fähigkeit zu kommunizieren. Und mit dem Backend von TYPO3 kann er den Inhalt erstellen, ohne mehr Kenntnisse besitzen zu müssen als für die Bedienung einer Textverarbeitung.

So haben wir also drei Charaktere mit jeweils speziellen Fähigkeiten, ohne welche die Erstellung einer CMS basierten Website *kein Spaziergang* ist, es sei denn Sie besitzen alle diese Fähigkeiten: Darstellung, Technik und Marketing. Normalerweise ist dies nur in einem Team der Fall, bei einzelnen Personen ist es eher unüblich. Seien Sie also gewarnt (wenn Sie weniger qualifiziert sind, müssen Sie entweder viel dazulernen oder Sie benutzen einfach Standardtemplates mit einem vorgegebenen Design, welches Sie nur noch etwas anpassen müssen (das beschreibt diese Anleitung nicht).

In diesem Tutorial zeige ich Ihnen, wie Raphael, Benoit und Picouto zusammenarbeiten müssen um moderne Websites erstellen zu können. Jedes Teammitglied kann sich in seinem eigenen Bereich entfalten und Werkzeuge nutzen die es mag, um leicht zu pflegende Websites mit einem traumhaften Designerstellen zu können. Mit TYPO3.

Fachliche Voraussetzungen

Webdesigner mit HTML/CSS Wissen, nicht notwendigerweise Webentwickler.

Damit Sie diesen Abschnitt vollständig durcharbeiten können, sollten Sie über gute HTML- und CSS-Kenntnisse verfügen. Weiterhin brauchen Sie eine funktionierende TYPO3-Datenbank, wie wir sie bereits in einem früheren Abschnitt dieses Tutorials angelegt haben. Schließlich sollten Sie mit Konzepten der Programmierung vertraut sein, um alles vollständig verstehen zu können. Aber keine Angst, Ihnen wird alles erklärt was Sie wissen müssen - seien sie einfach aufmerksam und nehmen Sie sich die Zeit, die Beispiele zu nachzuvollziehen.

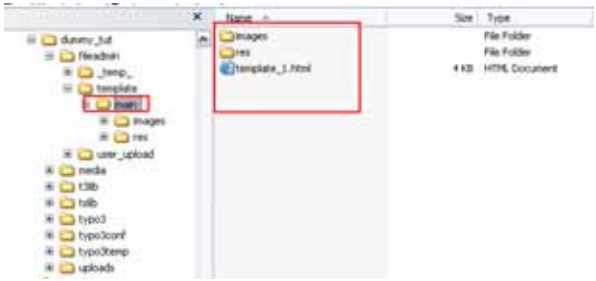
Das HTML Template

Das Webteam hat einen neuen Kunden: Main Dish & Son - und Raphael, der Künstler im Team, hat eine Vorlage in Form einer regulären HTML-Datei erstellt:

Diese HTML-Datei wird in dem Verzeichnis "fileadmin/template/main" relativ zur TYPO3-Installation (dummy package) gespeichert.

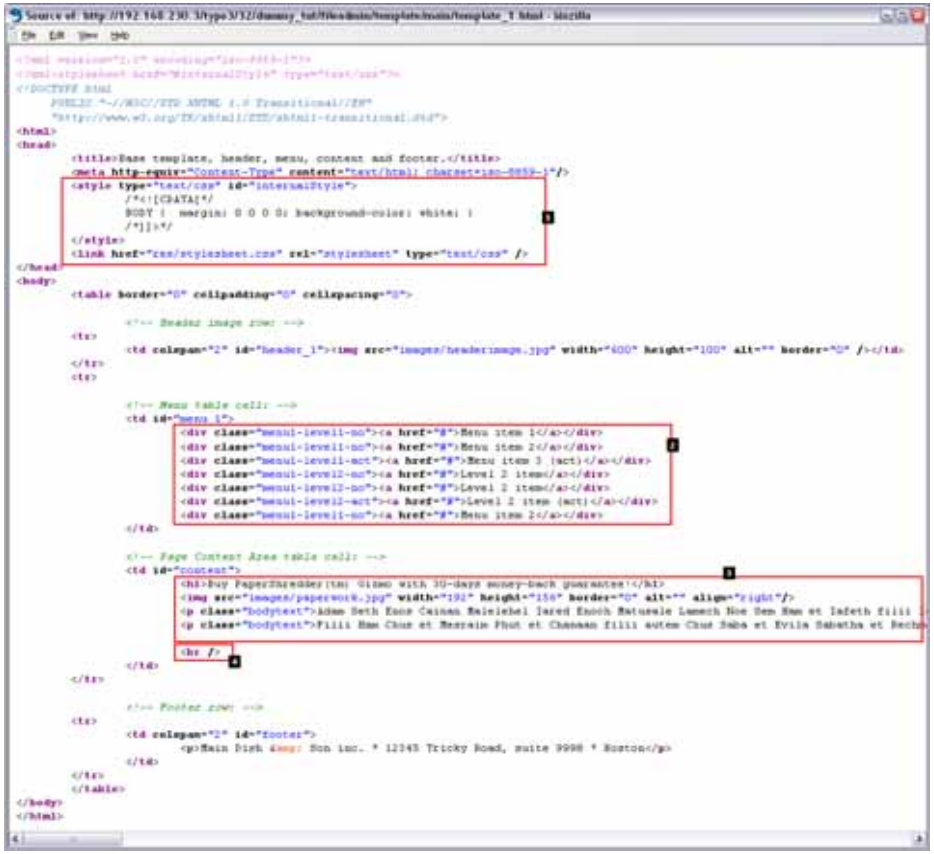
Um dieser Anleitung folgen zu können, sollten Sie jetzt den Inhalt des Ordners "part1/" von diesem Tutorial in das Verzeichnis "fileadmin/template/main" kopieren. Falls Sie die Extension zu diesem Tutorial noch nicht importiert haben, sollten Sie dies jetzt nachholen.

Falls Sie den vorigen Abschnitt über TypoScript übersprungen haben, sollten Sie zumindest den Seitenbaum wie im Kapitel "Erstellen einer Seitenstruktur" und ggf. ein leeres Template erstellen .



Zurück zu Raphaels Arbeit: die Vorlage ist tatsächlich lediglich nur eine reguläre HTML-Seite. Wenn aber TYPO3 diese Datei als Template nutzen soll, müssen einige Teile daraus *dynamisch* werden. Dazu gehört das Menü auf der linken Seite und der Bereich mit dem Blindtext im rechten Bereich in der Mitte.

Wenn Sie sich den Quelltext des HTML-Templates anschauen, werden Sie bemerken, dass es sich um ein XHTML-Dokument handelt, welches ein Stylesheet als Formatvorlage verwendet und mittels einer Tabelle verschiedenen Elemente anordnet.



Hier einige Kommentare zu diesem HTML-Template und den Herausforderungen, die auf uns zukommen:

1. Der Abschnitt ab dem Header des Dokuments muss in unsere Webseite übernommen werden, da hier der Bezug zu dem Stylesheet eingetragen ist.
   Aufgabe: Wir müssen überprüfen, ob dieser Teil in die von TYPO3 generierte Seite übernommen wird.
2. Das Menü auf der linken Seite wurde mit einem Abschnitt pro Menüpunkt erstellt. Jedes dieser Elemente hat eine ihm zugewiesene Klasse. Über diesen Namen der Klasse wird das Aussehen mittels der CSS-Formatvorlage gesteuert.
   Dies ist ein äußerst vernünftiges Prinzip einer Menügestaltung, da jedes Element nur aus wenig HTML-Code besteht (gut für die TypoScript-Implementierung), es ist leicht reproduzierbar (notwendig für ein dynamisches Menü).
   Aufgabe: Der Platzhalter für das Menü muss durch ein dynamisch generiertes Menü ersetzt werden.
3. Den Blindtext im Inhaltsbereich hat Raphael eingesetzt um das Erscheinungsbild darzustellen. Beachten Sie, wie dieser formatiert ist: mit <h1> und <p> Tags (der Klasse "bodytext", das ist ebenfalls klug gewählt, da auch TYPO3 diese Tags/Klassen bei der Erzeugung der Seite verwenden wird! Raphael hat wohl schon TYPO3-Erfahrung, oder?)
   Aufgabe: Austausch des Blindtextes gegen dynamisch erzeugten Seiteninhalt.
4. Dieser break-Tag für den Zeilenumbruch sorgt dafür, dass der Fußbereich nicht zu dicht am eigentlichen Text der Seite liegt.

Letztendlich sollten Sie noch beachten, wie die Zellen der Tabelle für das Menü und den Inhalt mit Tags mit id-Attributen versehen sind. Dies ist nicht nur für das Stylesheet von Bedeutung. Es gibt dafür noch einen anderen wichtigen Grund. Aber zuerst nochmal etwas Theorie zu HTML-Templates:

Grundlagen zum TEMPLATE cObject

Erstellen Sie zuerst eine neue Datei (fileadmin/template/test.html) mit folgendem Inhalt:

```
<!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>
```

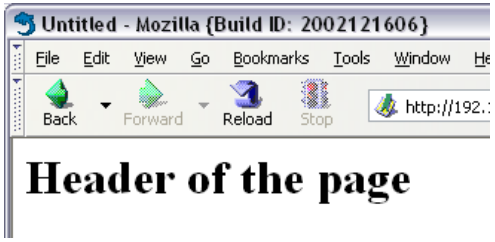
(Diese Datei finden Sie in der Tutorial Extension als "misc/test.html")

Dann fügen Sie bitte das folgende in das Setup-Feld Ihres Template Datensatzes:

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


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

page.10 < temp.mainTemplate
```

Dies fügt ein cObject vom Typ "TEMPLATE" an Position "page.10" ein. Die "template"-Eigenschaft dieses TEMPLATE cObjects ist als weiteres cObject vom Typ "FILE" definiert, welches die soeben erstellte Datei "fileadmin/template/test.html" einliest. Die Eigenschaften des TEMPLATE cObject finden Sie hier.

Wenn Sie jetzt die Änderungen speichern und das Frontend aufrufen, sollten Sie folgende Ausgabe sehen:

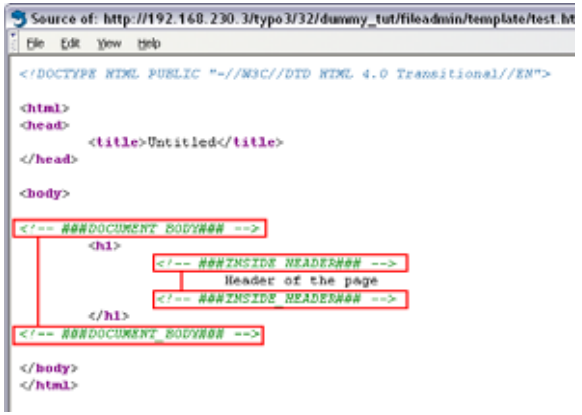


Und wenn wir den Quellcode betrachten, sehen wir, dass das TEMPLATE cObject die Datei, so wie sie ist, liest und zurückliefert:


```
<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>
```

Nun zum wichtigsten Punkt des TEMPLATE cObjects: es liest nicht nur die HTML-Datei - es erlaubt uns auch, Teile daraus zu extrahieren und durch dynamischen Inhalt zu ersetzen!

Ein Teilabschnitt wird als Inhalt zwischen zwei gleichen Markierungen definiert, die von ### umschlossen sind und wiederum (optional) in HTML-Kommentaren enthalten sind. Die Datei "test.html" Datei hat zwei Teilabschnitte, den "DOCUMENT BODY" und den Abschnitt "INSIDE HEADER". Wie Sie sehen können, werden die Markierungen dieser Teilabschnitte in HTML-Kommentaren eingeschlossen, so dass sie nicht sichtbar sind.



Nun ändern Sie bitte das Setup-Feld des Template Datensatzes wie folgt:

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


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

page.10 < temp.mainTemplate
```

Im Frontend sollte der HTML-Quelltext dann wie folgt aussehen:

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

        <h1>
                HELLO WORLD!
        </h1>

</body>
</html>
```

Die Veränderungen an den Eigenschaften des TEMPLATE cObjects haben folgendes bewirkt:

- In erster Linie wurde das TEMPLATE cObject angewiesen, mit dem Teilabschnitt zu arbeiten, der mit "###DOCUMENT_BODY###" markiert ist - folglich wurde die überflüssige Überschrift und die Body-Markierungen entfernt.
- Danach wurde der Teilabschnitt, der durch "###INSIDE_HEADER###" gekennzeichnet war, durch ein TEXT cObject ersetzt, das die Eigenschaften des "subparts.INSIDE_HEADER" des TEMPLATE cObjects definiert.

Das sollte leicht zu verstehen sein. Was wir jetzt noch tun müssen ist auf Raphaels Vorlage zu verweisen, nämlich "fileadmin/template/main/template_1.html, ähnliche Markierungen für die Teilabschnitte einzufügen und dann das Blindmenü und den Blindtext durch ein dynamisch erzeugtes Menü und den tatsächlichen Seiteninhalt zu ersetzen.

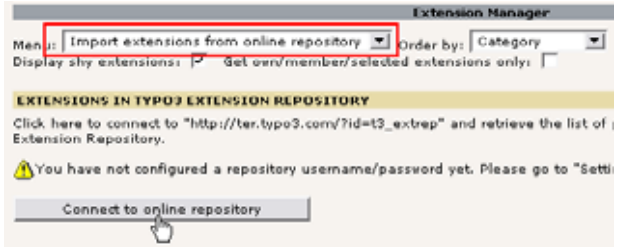
Die Template Auto-parser Erweiterung

Man könnte nun meinen, man müsse nur Raphaels Vorlage bearbeiten. Meiner Erfahrung nach können Sie nicht immer blind auf HTML-Autoren/Editoren vertrauen, d.h. es kann durchaus passieren, dass der HTML-Autor/Editor bei einer späteren Bearbeitung die Kommentar-Tags im Dokument verschiebt oder sogar entfernt. Stellen Sie sich vor Raphael nimmt einige Änderungen an der Vorlage mit Dreamweaver vor und die Markierungen der Teilabschnitte würden still und heimlich verschoben - das Template würde umgehend ausser Funktion gesetzt sein. Ferner ist der Pfad zu den Formatanweisungen des Stylesheets *relativ* zum Verzeichnis [domain]/fileadmin/template/main - nicht [domain]/, wo der HTML-Inhalt für das Frontend ausgegeben wird. Alle Pfadangabe müssten korrigiert werden.

Dies verlangt eine bessere Lösung. Dazu importieren Sie bitte die Template Auto-parser Erweiterung Extension - als Antwort auf alle Ihre Fragen...

Installation der Extension

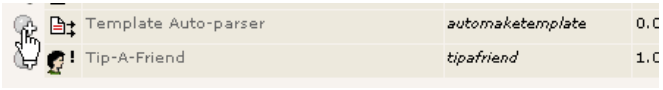
Gehen Sie in den Extension Manager und importieren Sie die Erweiterung aus dem Online Repository.



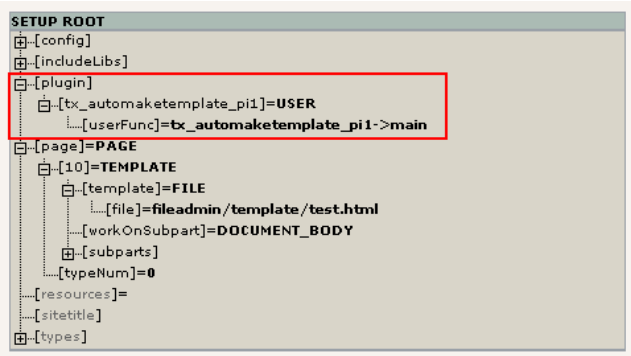
Finden Sie die Erweiterung "automaketemplate" und klicken Sie auf das Importsymbol:

Dies sollte im Falle des Erfolges erscheinen:

Gehen Sie zurück, wählen Sie "Available Extensions to Install" und installieren Sie die Extension.



Nach dem Klicken auf die "Make updates" Schaltfläche gehen Sie zurück in das Template Modul und überprüfen Sie den Object Browser:



Die Extension sollte ein USER cObject im Objektpfad "plugin.tx_automaketemplate_pi1" eingefügt haben, wie Sie oben sehen. Dieses cObject kann nun anstelle des FILE cObjects verwendet werden, um Raphaels Template einzulesen, die Markierungen automatisch einzufügen und die Pfade anzupassen.

Konfiguration des Template Auto-parsers

Wie für jede Extension, die wir nutzen möchten, können wir das Handbuch auf typo3.org zu Rate ziehen. Klicken Sie auf diesen Link und Sie erhalten um eine Tabelle mit den Eigenschaften dieses cObjects einzusehen.

Um Ihnen zu verdeutlichen, was der Template Auto-parser macht, füge ich hier die Ausgabe des Plugins als einzigen Inhalt ein und werde danach das Plugin so konfigurieren, dass es den üblichen Kopf- und Fußbereich *nicht* ausgeben wird.

Hier das Setup-Feld des Template Datensatzes:

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

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

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

    TD.all = 1
  }

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


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

page.10 =< plugin.tx_automaketemplate_pi1
```

Sichern sie das Template und rufen Sie das Frontend auf. Sie sollten eine exakte Darstellung von fileadmin/template/main/template_1.html sehen:

"Na und...?"werden Sie denken. Aber das ist, wenn Sie mal in den Quellcode schauen, schon genial:

Zwei wichtige Dinge sehen Sie dort:

- Eine Menge von Abschnitten des Template-Datensatzes wurden automatisch in Teilabschnitte gegliedert! (1 u. 2)
- Allen relativen Links/Verweisen wurde der Pfad "fileadmin/template/main/" vorangestellt. (3)

Dies geschah weil der Auto-parser dazu angewiesen wurde "elements" wie folgt zu behandeln:

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

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

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

("Elements" sind alle Tags, die von einem Start- und Endtag eingeschlossen sind, wie z.B. beim <td> Tag. Tags ohne

Endtag, wie z.B. <img>), müssen mit der "single" Eigenschaft des Template Auto-parsers definiert sein.

Die Konfiguration dieser Elemente bedeutet also, dass

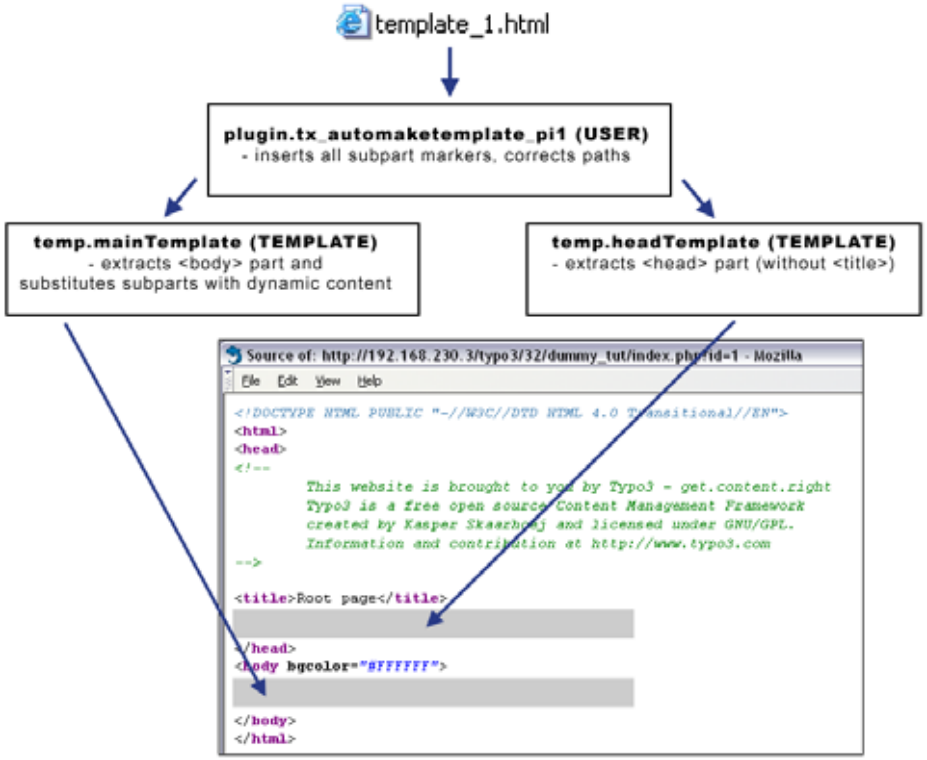
- a) das Element <body> soll mit der Markierung "###DOCUMENT_BODY###" als Teilabschnitt gekennzeichnet werden. soll
- b) das Element <head> soll mit der Teilabschnittsmarkierung "###DOCUMENT_HEADER###" umgeben werden. soll. Weiterhin sollen alle <title> Bereiche gelöscht werden. (Wir möchten nicht den Titel von Raphaels Vorlage auf allen unseren TYPO3-Seiten).
- c) Alle <td> Elemente, die gefunden wurden sollen markiert werden, solange aber keine Unterabschnitte definiert sind, werden nur <td>-Tags mit einem "id"- oder "class"-Attribut bearbeitet und mit einer Unterabschnittsmarkierung versehen. So wird der Tag <td id ="menu_1">den Inhalt aufnehmen und von den Unterabschnittsmarkierungen "<!--###menu_1###-->...<!--###menu_1###-->" umschlossen.

Was machen wir nun damit?

Die Antwort lautet, dass der Auto-parser es Raphael erlaubt, auf modernen CSS-Techniken basierte Templates *mit bewussten Gebrauch von id- und class-Attributen* zu erstellen. Gleichzeitig dienen diese Attribute als "Markierungen" für TYPO3, um Teile der Vorlage des Templates durch dynamischen Inhalt zu ersetzen. Einfach für Raphael, den Designer, klarer für Benoit, den Entwickler, und gut für Picoutos Finanzen, da für die Umwandlung und manuelle Anpassung der Vorlagen weniger Zeit benötigt wird.

Alles zusammensetzen

Schauen Sie sich folgende Abbildung an. Das ist, was wir machen möchten:

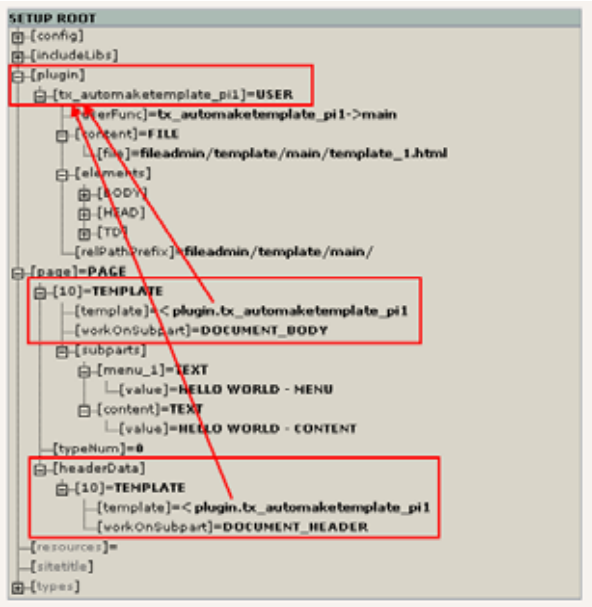


Die Template-Datei wird vom Auto-parser gelesen, die Ausgabe wird an das TEMPLATE cObject weitergeleitet, welches die Unterabschnitte ersetzt und letztendlich werden die Body- und Headerabschnitte der TYPO3-Seite eingefügt.

Dies wird über das unten gezeigte TypoScript erledigt, das in das Setup-Feld des Template Datensatzes eingegeben wird. Es ist ein längeres Listing, aber nehmen Sie sich die Zeit es zu verstehen:


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

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

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

    TD.all = 1
  }

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


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

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

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


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



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

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

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

Das führt dann zu dieser Baumstruktur.



hello world - Menü

Wie Sie sehen, werden die cObjects "temp.mainTemplate" und "temp.headTemplate" an Ihre jeweilige Position im Objektbaum kopiert, da jedes dieser cObjects eine Referenz auf das cObject USER des Template Auto-parser Plugins beinhaltet.

Beachten Sie bitte, dass der Objektpfad "page.headerData" eine numerische Liste der Content Objects ist, die den Inhalt für den <head>-Bereich der Seite definiert.

Ergebnis

Das Ergebnis sieht dann so aus:

Beachten Sie, dass die Bereiche des Menüs und des Inhaltes mit den Inhalten der cObjects TEXT ersetzt wurden, die nur zum Testen dienten.

Der HTML-Quellcode sieht jetzt folgendermaßen aus:

1. Der <head> Bereich ist ohne <title> Tag eingefügt worden.
2. Der <body> Bereich mit dem ersetzten Inhalt eingefügt.
3. Die Unterabschnitte ###header### und ###footer### wurden *nicht* ersetzt, da sie im TEMPLATE cObject "temp.mainTemplate" nicht definiert wurden! Das ist also kein Fehler.
4. Die Unterabschnitte ###menu_1### und ###content### wurden richtig ersetzt, da das TEMPLATE cObject "temp.mainTemplate" dafür konfiguriert war.

Zusammenfassung

TYPO3 hat aus Raphaels Vorlage automatisch den Inhalt für den <head> und <body> Bereich extrahiert, hat die relativen Pfade für die Grafiken, Links, Formulare und Formatvorlagen angepasst, hat den dynamischen Inhalt eingefügt, sowohl im <head> als auch im <body> Bereich der ausgegebenen Seite.

Der grundlegende Rahmenkonstrukt (Framework) ist jetzt betriebsbereit - wir müssen nur noch etwas dynamischen Inhalt hinzufügen, und zwar für das Menü und den Seiteninhalt! Dies geschieht am effektivsten mit TypoScript cObjects um die Menü- und Inhaltselemente zu generieren.

Das Menü erstellen

Das zweistufige Menü auf der linken Seite des Templates muss dynamisch erstellt werden um die Seitenstruktur des Backends wiederzugeben. Obwohl uns Raphael eine gute und klare Vorlage mit einfachen Bezeichnern für die Menüeinträge erstellt hat, ist es *nicht* die beste Lösung, einfach diese zu verwenden und diese Bereiche vom Auto-parser als Teilabschnitte extrahieren zu lassen. Stattdessen sollten wir dies explizit programmieren. Das bedeutet, dass Mr. Benoit Raphaels Vorlage manuell überprüfen muss. Dabei muss er sorgfältig herausfinden, was *ein einzelner Menüeintrag* enthält und dann diese Teile herausfiltern und dazu benutzen, das Objekt für die Generierung des Menüs einzurichten. Das bedeutet desweiteren, dass Raphael die zugrunde liegende Struktur nicht mehr verändern kann, ohne Benoit diese Änderungen mitzuteilen. Benoit muss dann diese Änderungen am Templatedatensatz entsprechend vornehmen. Aber wenn Raphael seine Arbeit ordentlich gemacht hat wird dies nicht nötig sein. Die CSS Formatvorlage sollte dann für alle Änderungen am Design genutzt werden.

Menüobjekte und Zustände der Einträge

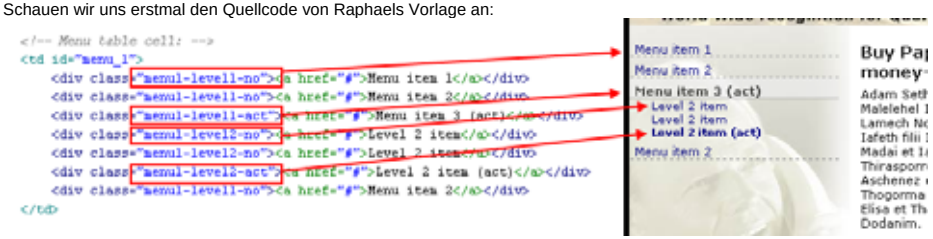
Zuerst werden wir das cObject HMENU zum Generieren des Menüs verwenden. Dieses Objekt nutzt "Menüobjekte" vom Typ

TMENU, GMENU, GMENU_LAYERS usw. für jede gewünschte Menüebene (abhängig von der Art des Menüs, das Sie verwenden möchten - textbasiert, grafisch, mit Ebenen usw.).

Für jedes "Menüobjekt" (also z.B. für TMENU oder GMENU) legen Sie für die jeweilige Ebene allgemeine Eigenschaften fest (z.B. Höhe und Breite für ein grafisches GMENU Element). Die jeweiligen Eigenschaften des Elements sind immer für einen speziellen *Zustand* definiert. Der Normalzustand (NO) muss immer definiert sein, aber zusätzlich können wir z.B. einen aktiven Zustand definieren (ACT) - dies bedeutet "wie soll der Menüeintrag aussehen, wenn wir auf einer Seite auf der selben Ebene oder unterhalb dieses Eintrag sind".

Dieses Tutorial soll nun nicht auf alle Einzelheiten des HMENU cObjects und aller davon abgeleiteten Möglichkeiten eingehen. Lesen Sie dazu bitte TypoScript by Example.

Schauen wir uns erstmal den Quellcode von Raphaels Vorlage an:



Wie Sie sehen, hat er für jeden Menüeintrag unabhängig von dessen Ebene und Zustand einen Tag eingefügt. Der Unterschied liegt ausschließlich im Wert des Attributs der Klasse! Das ist ein *kluges* Design, da das Einfügen der Abschnittsmarkierungen so einfach ist und Benoit eine Menge Zeit bei der Implementierung spart! :-) Und die eigentliche visuelle Steuerung liegt außerhalb von TYPO3, nämlich in der CSS-Formatvorlage.

Achten Sie darauf, wie Raphael "aktive" Menüeinträge gekennzeichnet hat.

Die Implementierung dieses Menüs ist sehr einfach. Und so geht's:

Zuerst definieren Sie ein temporäres Objekt am Anfang des Template-Datensatzes (vor der Definition von "temp.mainTemplate"):

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

Beachten Sie die roten Zeilen, die dem HTML-Code entsprechen, den Mr. Benoit aus Mr. Raphaels Vorlage kopiert hat. Nun ist er fest im Template Datensatz gespeichert.

Das einzige, was Sie jetzt noch tun müssen, ist dieses Objekt zu kopieren, so dass es das cObject des Teilabschnitts "menu_1" wird:

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

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

    # Substitute the ###content### subpart with some example content:
  subparts.content = TEXT
```

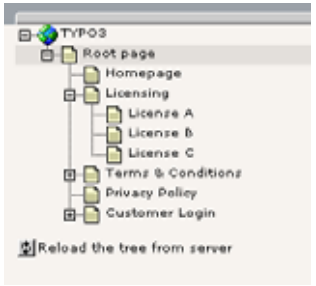


```
  subparts.content.value = HELLO WORLD - CONTENT
...
```

(Die geänderten Zeilen sind rot dargestellt)

... und das Ergebnis spricht für sich selbst:

Die Seitenstruktur (unten) wird exakt vom Menü oben wiedergegeben!



Und wie einfach ist es das Menüdesign zu ändern? Versuchen Sie es und ändern Sie es im Stylesheet "fileadmin/template/main/res/stylesheet.css":

```
/* Menu 1 column */
TD#menu_1 {
    vertical-align: top;
    width: 200px;
    background-image: url(../images/menubackground.jpg);
    background-repeat : no-repeat;
    padding-top: 10px;
}
TD#menu_1 DIV {
    width: 95%;
}
TD#menu_1 DIV A {
    color: navy;
    text-decoration: none;
}
TD#menu_1 DIV A:hover {
    text-decoration: underline;
}

/*  MENU 1, level 1, normal state (NO) */
TD#menu_1 DIV.menu1-level1-no {
  border-bottom: 1px dotted #999999;
  font-size: 11px;
  padding-top:5px;
  padding-left:5px;
}
/*  MENU 1, level 1, active state (ACT) */
TD#menu_1 DIV.menu1-level1-act {
  border-bottom: 1px solid #999999;
  font-weight:bold;
  font-size: 11px;
  padding-top:5px;
  padding-left:5px;

  background-color: #eeeeee;
  filter:alpha(opacity='70', style='0');
}
TD#menu_1 DIV.menu1-level1-act A {
  color:black;
}

/*  MENU 1, level 2, normal state (NO) */
TD#menu_1 DIV.menu1-level2-no {
  font-size: 10px;
  padding-left:20px;
}
/*  MENU 1, level 2, active state (ACT) */
TD#menu_1 DIV.menu1-level2-act {
  font-size: 10px;
  font-weight:bold;
  padding-left:20px;
}
```

Wie Sie sehen erlaubt diese Methode die Hauptfaktoren der visuelllen Gestaltung (das HTML-Template und die Formatvorgaben im Stylesheet) außerhalb von TYPO3 anzusiedeln. Es gibt hierbei einen direkten und bequemen Zugang für Mr. Raphael, den HTML/CSS-Designer. Und Mr. Benoit musste nur wenig Aufwand betreiben um das Frontend mittels TypoScript so zu konfigurieren, dass es Raphaels Design verwendet. Und beide können in der individuell für sie angenehmsten Weise entwickeln.

Wenn Sie mehr mit dem HMENUObjekt und damit zusammenhängendenDingen experimentieren möchten, schauen Sie dazu wieder in TypoScript by Example.

Seiteninhalte einfügen

Der beliebteste Weg, Seiteninhalte TYPO3 zu verwalten, ist *Inhaltselemente* in der Tabelle *tt_content* zu erstellen. Tatsächlich kenne ich Niemandem der dieses Konzept *nicht* verwendet, da es so flexibel, leistungsfähig und einfach einzubinden ist. Jedoch könnte dieses Konzept möglicherweise mit Ihren Vorstellungen von der Einbindung strukturierter Inhalte in HTML-Templates kollidieren.

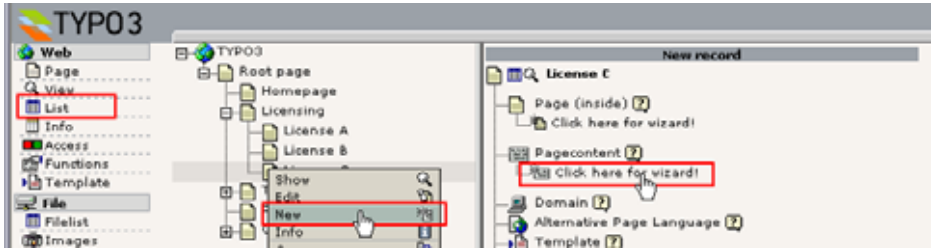
Inhaltselemente haben zahlreiche vordefinierte Typen (Text, Text mit Bild, Listen, Eingabefeldern usw.) und ein statisches Template mit vordefinierten TypoScript wird für gewöhnlich ohne weiteres Konfigurieren diese Elemente für Sie generieren. Das ist wirklich genial! Da Inhaltselemente einen bestimmten Typ haben und jeder Typ zusätzliche Optionen, ist es unmöglich Inhaltselemente mit dem festen Konzept einer HTML-Vorlage zu generieren. Das passt einfach nicht zusammen und Sie werden das erkennen, wenn Sie etwas mehr damit experimentiert haben. Folglich werden mit PHP-Funktionen kombinierte cObjects genutzt, um Inhalte mit komplexen Bedingungen zu generieren.

Da wir keine Inhaltselemente durch die Verwendung von HTML-Templates darstellen können, wie gehen wir also in diesem Fall vor? Indem Sie die neueste Herangehensweise zur Darstellung von Inhaltselementen nutzen, die Extension "css_styled_content". Hiermit wird der gesamte Inhalt in einfachen HTML-Elementen eingebunden - und das Festlegen der Formatvorgaben bleibt Ihrer externen CSS-Formatvorlage überlassen.

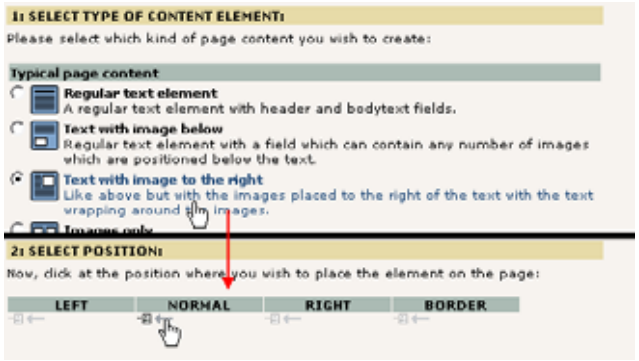
Schauen wir, wie das geht. Zuerst erstellen wir ein Inhaltselement:

Ein Inhaltselement

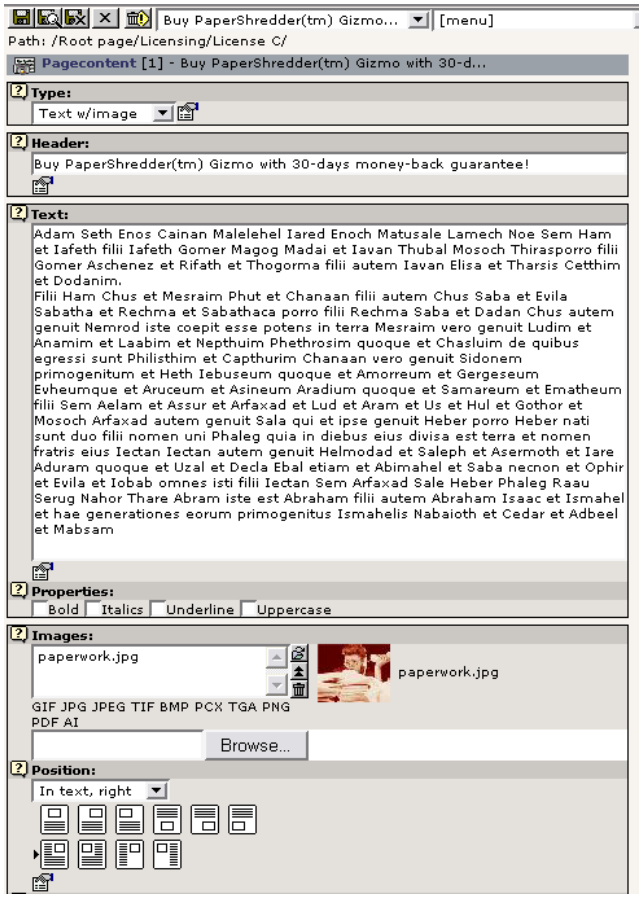
Im Listenmodul klicken Sie bitte auf das Icon der Seite "License C" und wählen dann "Neu". Im rechten Frame klicken Sie dann auf den Link des Assistenten für den Seiteninhalt:



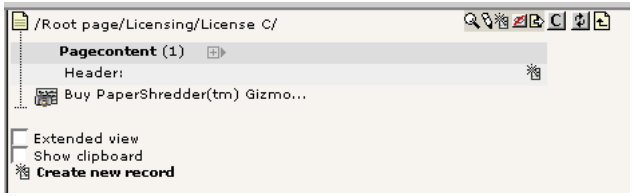
Wählen Sie "Text mit rechts liegendem Bild" und dann die Spalte "Normal":



Nachdem Sie den Inhalt eingegeben und gespeichert haben sollten Sie folgende Ausgabe sehen:



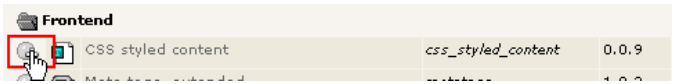
Jetzt haben wir auch ein Inhaltselement auf der Seite "License C" verfügbar:



Einfügen eines statischen Templates

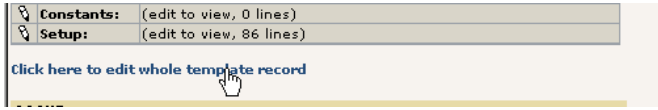
Um das Inhaltselement anzeigen zu lassen, müssen wir ein *statisches Template* einfügen, welches die vielen hundert Zeilen TypoScript-Code für die spätere Ausgabe erstellt. Dazu ist der gesamte Template-Datensatz zu bearbeiten.

Zuerst müssen Sie natürlich die Extension "css_styled_content" installieren.

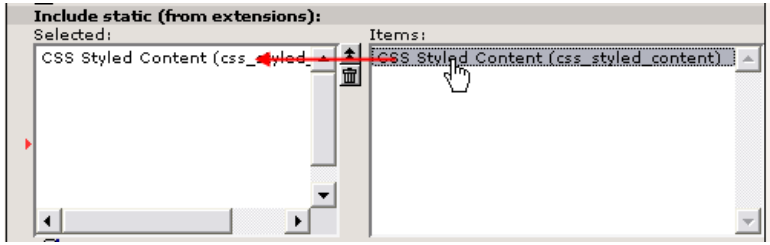


Klicken Sie auf das Installationssymbol und bestätigen Sie die Änderungen auf der nächsten Seite.

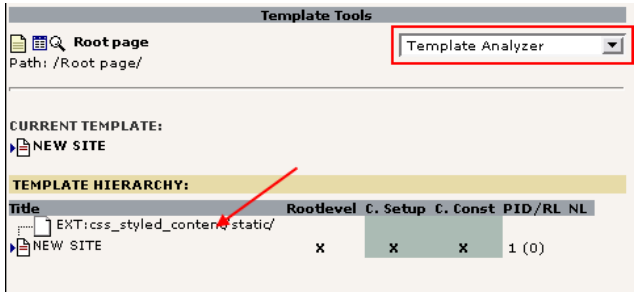
Zurück zum Template: Im Template Modul klicken Sie auf diesen Link:



Dann sollte der "CSS Styled Content" als "static file" im Bereich "Include Static (from extensions)" verfügbar sein:



Klicken Sie darauf und sichern Sie das Template. Danach wechseln in den "Template Analyzer" und betrachten dort das Template:



Der "Template Analyzer" zeigt die Hierarchie der Template-Datensätze und statischen Templates, die im aktuellen "main template record" eingebunden sind. Durch Anwahl des statischen Templates wird dieses vor Generierung des TypoScript-codes im Template-Datensatz "NEW SITE" eingebunden. Das bedeutet, dass jedes Objekt, das innerhalb von "EXT:css_styled_content/....." definiert ist, im Template-Datensatz "NEW SITE" benutzt und dort hineinkopiert werden kann. Das werden wir jetzt auch tun, da die statische Vorlage ein Objekt "styles.content.get" enthält. Dabei handelt es sich um ein Inhaltselement, das alle Inhaltselemente aus der Spalte "Normal" der aktuellen Seite auswählt.

Ändern Sie also diesen Teil der Definition des "temp.mainTemplate" Objekts im Template-Datensatz:

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

```
  workOnSubpart = DOCUMENT_BODY

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

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

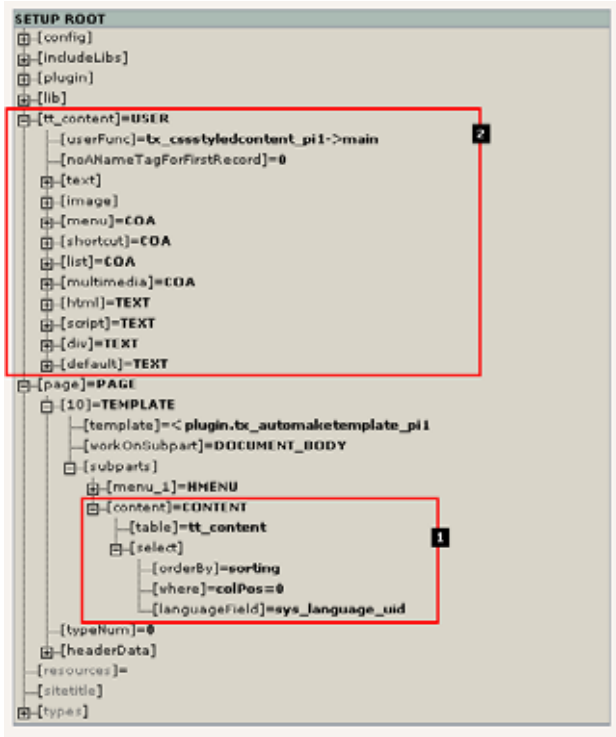
Nach einem Aktualisieren des Frontends im Browsers sollte die Seite "License C" anschließend so aussehen:

Und der HTML-Quelltext sieht dann so aus:

1. Anscheinend wurde die Überschrift des Inhaltselements in <h1> Tags eingeschlossen generiert. Wenn Sie einen anderen "Layout"-Typ wählen, kann dies auch <h2> (für Layout 2) sein.
2. Jede Zeile des Textes im Body wird in mit <p> Tags eingeschlossen und erhält die Zuweisung der Klasse "bodytext". So können Sie die Darstellung von Seiteninhalten über Ihre Stylesheet-Formatvorlage steuern. Beachten Sie bitte wie Raphael bereits schon den Blindtext in der Template-Datei unter Verwendung von "<p class="bodytext">" Tags gestaltet hat!
3. Jedes eingefügte Inhaltselement erhält einen <a> Tag, welcher als Wert des Attributes name seine uid beinhaltet. Dieser Ankerpunkt kann zum direkten Aufruf einer bestimmten Position innerhalb der Seite genutzt werden.
4. Dies hat das HMENU Objekt beim Aufbau des zweistufigen Menüs eingefügt. Beachten Sie auch, wie der Link automatisch auf die richtige Seite verweist und wie eine zusätzliche "onFocus" Routine eingefügt ist.

Der Objektbaum?

Was haben wir jetzt durch das Kopieren von diesem angeblichen Inhaltselement "styles.content.get" wirklich in den Objektbaum eingefügt? Schauen wir im Object Browser nach:



Hier sind zwei interessant Punkte:

1. Der Objektpfad "styles.content.get" beinhaltete offensichtlich ein cObject vom Typ CONTENT. Wenn wir uns die Attribute anschauen, sieht es danach aus, dass dieses cObject Datensätze aus der Tabelle "tt_content" auswählt und sie nach dem Feldeintrag "sorting" geordnet ausgibt. Für weitere Informationen zum CONTENT cObject folgend sie diesem Link.
2. Desweiteren wurde ein komplett neues Toplevel Object (TLO) definiert - "tt_content", Standardmäßig nutzt das cObject CONTENT dieses TLO (ebenfalls als cObject definiert) um die Ausgabe der gefundenen Datensätze zu generieren (1). Wie wir sehen, wird das "tt_content" TLO als USER cObject definiert, das eine PHP-Funktion der "css_styled_content" Extension aufruft - dafür haben wir ja auch diese Erweiterung installiert. Für weitere Einzelheiten schauen Sie bitte in die Dokumentation zur "css_styled_content" Extension.

Die ganz Neugierigen können auch mal in die Klasse "tx_cssstyledcontent_pi1" schauen und die dahinter steckende Logik betrachten. Hier dazu ein Ausschnitt aus der Funktion main():

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

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



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

XHTML Konformität

Wenn Sie XHTML konforme Webseiten erstellen möchten, ist auch das mit TYPO3 möglich. Die Version 3.6.0 von TYPO3 zielt darauf ab, XHTML-Konformität so weit als möglich zu integrieren (Level: XHTML transitional).

In diesem Tutorial müssen wir sicherstellen, dass die HTML-Vorlage mit XHTML konform ist. Das ist der erste Schritt und darauf müssen *Sie* achten.

Der nächste Schritt ist dafür zu sorgen, dass TYPO3 den XHTML Dokumententyp ausgibt. Dies wird durch eine einfache Zeile im Setup-Feld des Haupttemplate-Datensatzes erreicht:

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

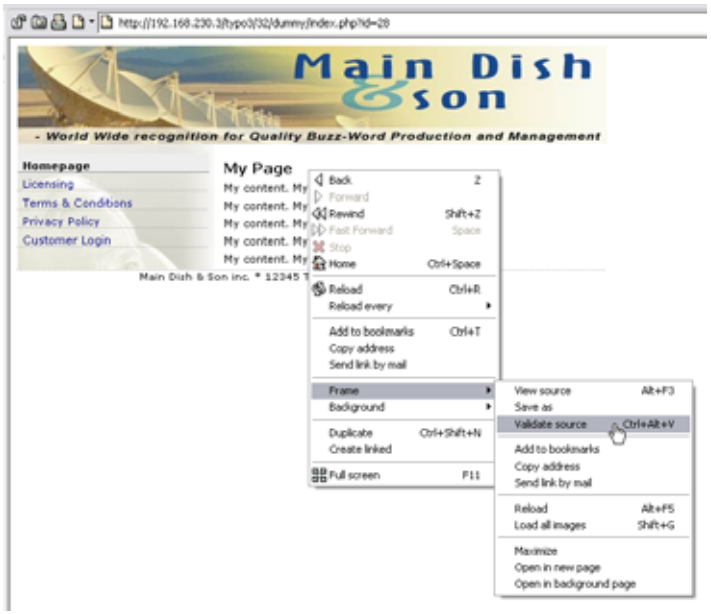
Die Menüs müssen natürlich auch entsprechend konform sein. Zum jetzigen Zeitpunkt werden Zeichen wie "&" auch als solche ausgegeben. Das sollte vermieden werden. In den TypoScript-Code für das Menü sollten diese vier Zeilen (in rot dargestellt) eingefügt werden:

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

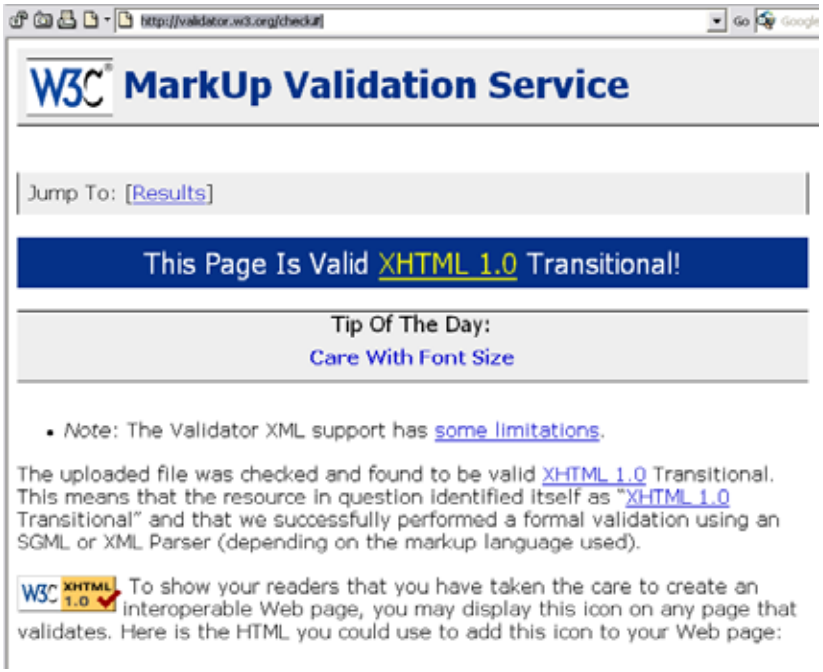
Dadurch wird gewährleistet, das die Bezeichnungen der Menüeinträge die PHP-Funktion htmlspecialchars() durchlaufen, die z.B. "&" in "&" umwandelt.

Danach können Sie die Site z.B. mit Opera 7 testen. Opera besitzt eine angenehme Abkürzung zum Überprüfen der Formatkonformität der Seite.

Klicken Sie dazu mit der rechten Maustaste in die HTML-Seite, unter "Frame" finden Sie dann den Eintrag "Validate source":



Opera sendet dann den Quelltext automatisch an die Seite "http://validator.w3.org/" und bei Erfolg sollten Sie folgendes Ergebnis sehen:



Aufräumarbeiten

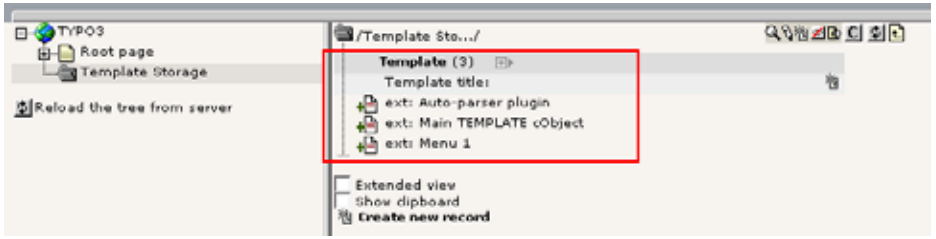
Wir haben nun unsere Website fertiggestellt. Mr. Benoit und Mr. Raphael haben ihre Aufgaben erledigt und vielleicht hat Mr. Picouto bereits nun den gesamten Inhalt der einzelnen Unterseiten des Baumes eingefügt. Das wissen wir nicht. Aber die Grundlage der Website selbst ist fertig. Sie basiert auf einer regulären HTML-Datei im Verzeichnis "fileadmin/template/main" und nur das Menü und der Inhalt werden wie benötigt dynamisch ersetzt.

Bessere Struktur des Templates

Das letzte was wir noch tun müssen ist den Template-Datensatz etwas besser zu organisieren. Auch wenn nur wenig TypoScript genutzt wird, sehen wir doch, dass der Datensatz schnell groß und unübersichtlich wird. Die Lösung ist eine Reihe von Template-Datensätzen zu erstellen, um diese einzubinden.

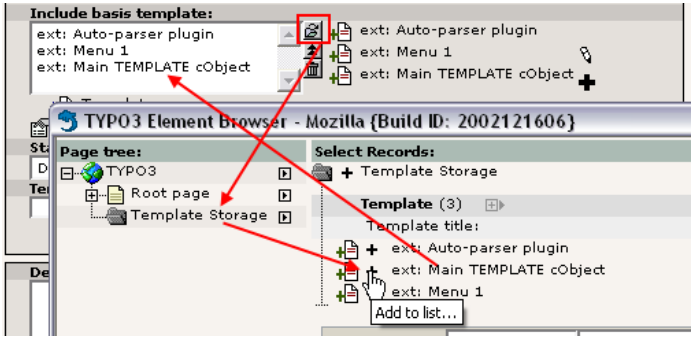
Dazu sollten Sie zuerst eine komplett neue Seite auf der untersten Ebene (root level) erstellen. Wählen Sie den Typ "sysFolder" (das bedeutet "Speicherplatz für jedes gewünschte Datenbankelement")..

Dann erstellen Sie drei Template-Datensätze wie in dieser Abbildung:



Kreuzen Sie nirgendwo die "Clear" -Anwahlfelder an! Kopieren Sie lediglich nur die passenden Objektdefinitionen "temp.xxxx" des Templates "NEW SITE" in die drei neuerstellten Template-Datensätze. aus der "NEW SITE" in die neue Vorlage. Die Reihenfolge spielt dabei keine Rolle.

Der nächste Schritt ist diese drei Templates in das Haupt-Template der "NEW SITE" einzubinden. Also bearbeiten Sie das Template von "NEW SITE".



Die Reihenfolge ist wichtig: das zuerst aufgeführte Template wird zuerst eingefügt und da der Vorlagensatz "ext: Main TEMPLATE cObject" Kopien von z.B. dem Objekt "temp.menu_1" erstellt, muss dieses dann als letztes eingefügt werden.

Speichern Sie den Template-Datensatz und betrachten Sie die neue Struktur im Template Analyzer:

Wie Sie sehen können wird erst das statische Template eingebunden, dann die drei "Basisvorlagen" aus unserem Ordner und zum Schluss der Haupttemplatesatz (main template record). Sie können den Titel jeder Vorlage anklicken und darunter deren Inhalt ansehen.

Weitere HTML-Design Betrachtungen

Wie Sie in diesem Abschnitt gesehen haben, können Sie sehr komfortabel das HTML-Template eines Webdesigners implementieren. Volle *Rückwärtskompatibilität* zur zum ursprünglichen Template ist dabei sichergestellt (da keine Markierungen von Hand in die Vorlage eingefügt werden müssen). Weiterhin werden nur vertraute Techniken wie HTML und CSS genutzt - es werden keine verwirrenden XSLT-Vorlagen benötigt (obwohl Mr. Benoit natürlich auch XSLT für die Ausgabe auf PHP-Ebene hätte nutzen können).

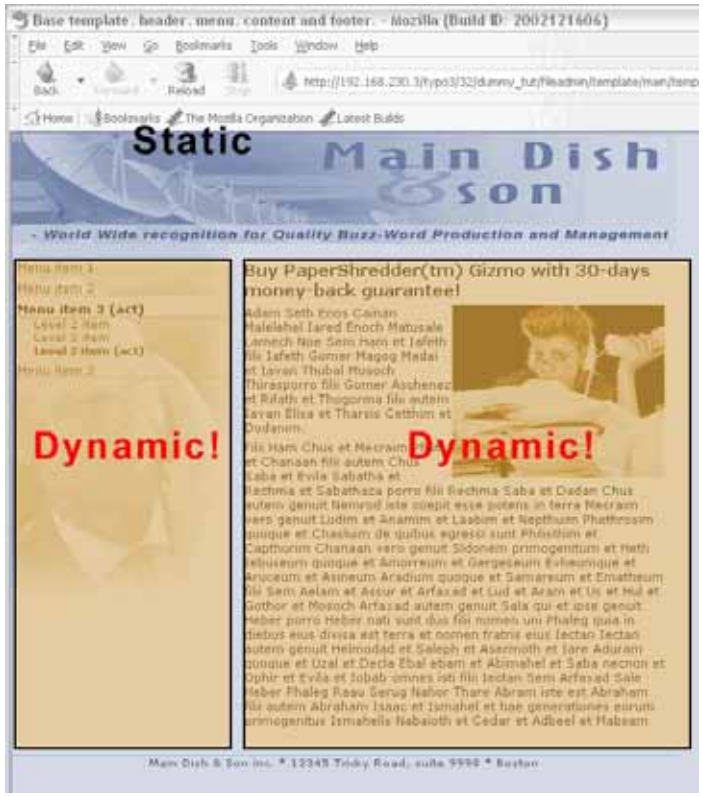
Auch wenn der Designer, Mr. Raphael, fast grenzenlose Freiheit hat, sollte er doch die Grundlagen von statischem und dynamischem Inhalt kennen.

Klarstellen welche Teile dynamisch und welche statisch sind

Mr. Raphael muss in erster Linie verstehen - vermutlich zusammen mit dem ganzen Team - welche Teile der Seite statisch und welche dynamisch sind. Statische Teile bleiben von TYPO3 unberührt. Die dynamischen Teile sollen jedoch mit dem Inhalt ersetzt werden, der aus den Datenquellen innerhalb von TYPO3 kommt.

Wenn dynamische Bereiche erkannt werden, muss Raphael folgendes unternehmen:

- Dieser Bereich muss von einem einzelnen HTML-Element umschlossen sein, dabei kann es sich um , <td> oder Elemente handeln.
- Das umschließende Element muss entweder ein id oder class Attribut besitzen, welches das Template Auto-parser plugin finden und in Unterabschnittmarkierungen einschließen muss, z.B. <td id="content"> für die Tabellenzelle, die den Inhalt aufnehmen soll.



Es ist völlig in Ordnung den Blindtext in der Vorlage zu belassen - dieser wird ohnehin schließlich durch die dynamischen Teile ersetzt.

Vereinfachen Sie die Darstellungsformatierung, nutzen Sie CSS Formatvorlagen

Wir schreiben das Jahr 2004 und es wird Zeit dem <font> Tag und den bgcolor-Attributen auf Wiedersehen zu sagen. Packen Sie das alles in eine externe Formatvorlage. Für den Einsatz eines CMS wird dadurch erreicht so viele externe Techniken wie irgend möglich nutzen zu können und nur soviel CMS-spezifische Technologien (wie TypoScript Template-Datensätze) wie wirklich benötigt zu verwenden.

In diesem Abschnitt dieses Tutorials war das beste Beispiel hierfür das zweistufige Menü:

```
<!-- 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>
```

Schauen wir uns ein einzelnes Element an (oben rot dargestellt). Es hätte auch so oder ähnlich aussehen können:

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

```
                <tr>
                    <td><img src="gray_dotted_line.gif" width="180" height="1" alt=""/></td>
                </tr>
                </table>
                    <!-- Menu item level 1, end -->
                ...
            </td>
```

Jetzt denken Sie wieder zurück an unser TMENUITEM Object im Template-Datensatz. Es war so definiert, dass es Elemente ausschließlich mit diesem einfachen Bereich umschließen sollte:

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

... und ja, Sie haben es erraten: mit der oben genannten Implementierung würden wir die Elemente in etwa wie folgt eingeschlossen haben müssen:

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

Aber auch das reichte noch nicht. Sie müssten noch den Klassennamen für den <a> Tag um eine weitere Eigenschaft des "No" Status Objektes sowie den Pfad der Grafik der grau gepunkteten Linie ergänzen - irgendwas mit "fileadmin/template/main/".

Benutzen Sie keine class Attribute für dynamische Links

Viele Designer sind der Meinung, es wäre am besten, wenn sie ihre Menüelemente und andere Links mit Klassenattributen für den <a> Tag versehen, z.B. so:

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

Jedoch ist das nicht der Fall, da *genau* dieser <a> Tag dynamisch von TYPO3 erzeugt wird! Damit dies mit unserem TMENUITEM funktioniert benötigen wir weitere TypoScript Eigenschaften:

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

Das führt zu komplexerem TypoScript Code und die Stylesheet-Formatvorlage wird unflexibler. Um z.B. das Aussehen von Links über CSS zu steuern, können Sie folgendes nutzen:

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

... aber Sie können dann nicht die Bearbeitung der Elemente kontrollieren, da diese *nicht* mit class oder id markiert sind! Wenn die den Weg, wie wir ihn oben beschritten haben, gehen, wo das <a> Element keine Klassenattribute , sondern nur das Elemente welche hatte, dann haben Sie beide im Griff: sowohl die Ebene *als auch* alle darin enthaltenen Links:

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

Finger weg von row/colspans!

Das ist der Nummer Eins Killer für dynamischen Inhalt - colspans im dynamischen Inhalt zu benutzen! Das wird dadurch geradezu ausgeschlossen. Zum Verständnis schauen Sie sich diese Implementierung des Menüs an:

```
    <tr>
```



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

        <!-- Page Content Area table cell: -->
        <td id="content" rowspan="7">
            . . .
        </td>
    </tr>
    <tr>
        <td class="menu1-level1-no"><a href="#">Menu item 2</a></td>
    </tr>
    <tr>
        <td class="menu1-level1-act"><a href="#">Menu item 3 (act)</a></td>
    </tr>
    <tr>
        <td class="menu1-level2-no"><a href="#">Level 2 item</a></td>
    </tr>
    <tr>
        <td class="menu1-level2-no"><a href="#">Level 2 item</a></td>
    </tr>
    <tr>
        <td class="menu1-level2-act"><a href="#">Level 2 item (act)</a></td>
    </tr>
    <tr>
        <td class="menu1-level1-no"><a href="#">Menu item 2</a></td>
    </tr>
```

Das ist nahezu unmöglich zu implementieren. Der erste Menüeintrag ist in einer Tabellenzeile zusammen mit der Zelle für den Inhalt. Die restlichen sechs Menüeinträge befinden sich in eigenen Zeilen. Weiterhin benötigt die Zelle mit dem Inhalt ein colspan="7", um alle Zeilen des Menüs zu umspannen!

Aus technischer Sicht ist dies ein sehr schlechtes Design, da:

1. die Beschreibung des Menüs auf verschiedene Bereiche des HTML-Codes verteilt ist
2. es eine Abhängigkeit zwischen dem colspan Attribut und der (dynamischen!) Anzahl der Menüeinträge gibt.

Designer sollten solches Design vermeiden. Stattdessen sollten Designer wie Mr. Raphael lernen, die Beschreibung für dynamische Elemente wie Menüs so zu erstellen, dass sie *leicht reproduzierbar* sind. Hier ein gutes Beispiel:

Von unserer aktuellen Vorlage:

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

Alternativ mit einer Tabelle:

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

Die Beispiele könnten auch noch komplexer sein, ähnlich dem obigen, das *jedes* Element in einer eigenen Tabelle einschloß. Entscheidend ist aber nicht die Menge an HTML pro Eintrag, entscheidend ist die Reproduzierbarkeit.

Mr. Raphael sollte diesen Umständen Rechnung tragen, damit Mr. Benoit nicht Stunden damit verbringen muss eine mangelhafte HTML-Struktur zu implementieren. Eine schnelle Umsetzung des Projektes als Ergebnis der symbiotischen Zusammenarbeit von Raphael und Benoit könnte auch Mr. Picouto glücklich machen - vielleicht lädt er sie ja zum Mittagessen ein!

Schlussbemerkung:

Zu dieser Anleitung gibt es noch einen Teil 2 und 3 in einem anderen Dokument. Diese Teile - besonders Teil 2 - sind eine Fortsetzung von diesem (Teil 1) und gehen ebenfalls von dem Team mit Raphael, Benoit und Picouto aus. Bevor Sie aber weitermachen bedenken Sie bitte, dass Sie dazu zusätzliche Voraussetzungen benötigen. In der Zwischenzeit wäre es sicherlich nicht falsch, wenn Sie ihr jetziges Wissen anwenden, andere Anleitungen lesen und somit ein höheres Niveau erreichen, speziell in den technischen und die Entwicklung betreffenden Bereichen.