

**in2code GmbH**  
Kunstmühlstraße 12a  
83026 Rosenheim  
  
t +49 8031 8873983  
f +49 8031 8873985  
  
info@in2code.de  
www.in2code.de

# TYPO3 Administration und Integration mit Hilfe von FLUID und TypoScript

# 1. Inhalt

1.	Inhalt .....	2
1	Administration und Integration .....	9
1.1	Einführung .....	9
1.2	Begriffserklärung .....	10
1.2.1	Integration .....	10
1.2.2	Fluid .....	11
1.2.3	TypoScript .....	12
1.2.4	Frontend und Backend .....	13
1.2.5	Die Aufteilung im Backend .....	14
1.2.6	Composer .....	15
1.3	Installation .....	16
1.3.1	Voraussetzungen .....	16
1.3.2	Installation von TYPO3 .....	17
1.3.3	Best practice .....	19
1.3.3.1	AdditionalConfiguration .....	19
1.3.3.2	DevelopmentConfiguration .....	20
1.4	Konfiguration mit Hilfe von TypoScript .....	23

1.4.1	Einführung .....	23
1.4.2	Hello World – die erste Ausgabe .....	24
1.4.2.1	Ersten Seitenbaum erstellen.....	24
1.4.2.2	Erstellung eines ersten TypoScript Templates .....	25
1.4.2.3	Felderklärung im TypoScript Template.....	28
	Reiter „Allgemein“ .....	29
	Reiter „Optionen“.....	30
	Reiter „Enthält“ .....	31
	Zugriff .....	32
	Hinweise.....	32
1.4.3	TypoScript Grundlagen .....	33
1.1.1.1	Einführung .....	33
	Erste Zeile .....	33
	Zweite Zeile .....	33
	Dritte Zeile .....	34
	Vierte Zeile.....	34
1.4.3.1	TEXT .....	35
1.4.3.2	IMAGE .....	36
1.4.3.3	COA / COA_INT .....	37

1.4.3.4	CONTENT .....	38
1.4.3.5	USER / USER_INT .....	39
1.4.3.6	Verwendung der TypoScript Konstanten .....	41
1.4.3.7	Kopieren von Elementen .....	42
1.4.3.8	„styles.content.get“ – Das Geheimnis .....	43
2	TypoScript – Templating .....	44
2.1	Assets einbinden .....	44
2.1.1	Einbindung von CSS – Dateien .....	44
2.1.2	Einbindung von Javascript – Dateien .....	44
2.1.3	Manuelle Einbindung von Tags in den HTML-Header .....	45
2.1.4	Meta-Tags im HTML Header .....	45
2.2	Allgemeine Konfiguration .....	46
2.3	Erweiterte Anforderungen .....	47
2.3.1	HMENU .....	47
2.3.2	TypoLink .....	47
2.3.3	If .....	49
2.3.4	CASE .....	50
2.3.5	Conditions .....	51
2.4	Verwendung von Backend-Layouts .....	53

2.4.1	Erstellung von Backend-Layouts .....	53
2.4.1.1	Unterscheidung der Ausgabe im Frontend.....	56
2.4.2	Gridelements als Ergänzung zu Backend-Layouts.....	57
2.4.3	Templates komplett auslagern .....	58
2.4.3.1	Auslagern in eigene Extension.....	58
2.4.4	Der TypoScript-Object-Browser.....	59
3	Fluid – Templating.....	60
3.1	Einführung.....	60
3.1.1	Klassisches Templating.....	60
3.1.2	Vorteile von Fluid gegenüber. klassischem Templating.....	61
3.1.3	Grundsätzliches Vorgehen.....	61
3.1.3.1	Erstellung einer HTML Datei.....	61
3.1.3.2	Aufruf im TypoScript.....	62
3.2	Unterteilung in Templates/Layouts/Partials.....	63
3.3	Variablen.....	64
3.4	ViewHelper .....	66
1.1.1.	Inline- und Outline-Schreibweise .....	66
3.4.1.1	ViewHelper mit beginnendem und schließendem Tag .....	66
3.4.1.2	Beispiel für verschachtelte View-Helper.....	67

3.4.2 Auswahl häufig genutzter ViewHelper .....	67
3.4.2.1 Debug – ViewHelper (TYPO3 Fluid) .....	67
3.4.2.2 Comment – ViewHelper (TYPO3 Fluid) .....	68
3.4.2.3 Viewhelper für die Struktur (TYPO3 Fluid) .....	68
Render – ViewHelper (TYPO3 Fluid) .....	69
Layout – ViewHelper (TYPO3 Fluid) .....	69
3.4.2.4 Link – ViewHelper .....	70
Page – ViewHelper .....	70
Typolink _ViewHelper .....	70
E-Mail-ViewHelper .....	70
External-ViewHelper .....	71
Action- ViewHelper .....	71
3.4.2.5 Image – ViewHelper .....	71
3.4.2.6 Format – ViewHelper .....	72
ViewHelper "format.html" (TYPO3 Fluid) .....	72
Viewhelper „format.raw“ (TYPO3 Fluid) .....	72
Viewhelper „format.htmlspecialchars“ (TYPO3 Fluid) .....	72
ViewHelper "format.printf" (TYPO3 Fluid) .....	73
ViewHelper "format.crop" (TYPO3 CMS) .....	73

3.4.2.7	Translate – ViewHelper (TYPO3 CMS) .....	73
3.4.2.8	ViewHelper „cObject“ (TYPO3 CMS).....	73
3.4.2.9	Spaceless – ViewHelper (TYPO3 Fluid).....	74
3.4.2.10	Viewhelper für If-Statements (TYPO3 Fluid).....	74
3.4.2.11	Then – und Else ViewHelper (TYPO3 Fluid) .....	75
3.4.2.12	Switch – ViewHelper (TYPO3 Fluid).....	75
3.4.2.13	Form – ViewHelper (TYPO3 CMS).....	76
3.4.2.14	Variable – ViewHelper (TYPO3 Fluid).....	77
3.4.2.15	Alias – ViewHelper (TYPO3 Fluid).....	77
4	Fluid-Styled-Content.....	78
4.1	Installation der Extension „fluid_styled_content“.....	78
4.2	Content-Elemente anpassen .....	79
4.2.1	Verzeichnisstruktur.....	80
4.2.1.1	Das Verzeichnis „Layout“.....	80
4.2.1.2	Das Verzeichnis „Templates“.....	80
4.2.1.3	Das Verzeichnis „Partials“+ .....	80
4.2.2	Layouts, Templates und Partials überschreiben.....	80
4.2.3	Aufbau eines Templates oder Partials .....	81
5	Links und Hilfe .....	83

5.1	Links .....	83
5.2	Kontakt .....	84

# 1 Administration und Integration

## 1.1 Einführung

Nachfolgende Beispiele und Screenshots beziehen sich ausschließlich auf **TYPO3 8 LTS**.

Sämtliche gezeigte Konfiguration und Screenshots beruhen auf Bordmitteln und benötigen keine weiteren Extensions. Falls doch, wird dies ausdrücklich erwähnt.



## 1.2 Begriffserklärung

### 1.2.1 Integration

Unter Integration versteht der TYPO3-Administrator die Arbeiten rund um TYPO3, um ein HTML Layout in TYPO3 einzubinden. Letztlich also die Einbindung eines oder mehrerer Layout-Gerüste, die durch TYPO3 dynamisch mit Inhalt befüllt werden soll.

Im Idealfall geht der Weg also vom Design über den HTML-Prototyp (oftmals auch Klickdummy genannt) hin zur Einbindung in das CMS, um Inhalte dynamisch anzeigen zu können.

Ein Großteil der Integration in TYPO3 beruht auf dem Wissen der Konfiguration (TypoScript) und dem Templating (Fluid).

Upd.	A/I	Extension	Key	Version	State	Actions
		Fluid Templating Engine	fluid	8.7.0	stable	

*Tipp:* Die Extension fluid ist bereits standardmäßig installiert und aktiviert

## 1.2.2 Fluid

Bei Fluid handelt es sich um eine sehr flexible und moderne Template-Engine, die als Package vom TYPO3-CMS und anderen Frameworks (Flow, Neos, etc..) genutzt werden kann.

Als Brücke dient die TYPO3-Erweiterung fluid, die das Standalone-Package (siehe Verzeichnis vendor/typo3fluid) einbindet und um weitere Funktionen (hauptsächlich eigene ViewHelper) ergänzt.

Die alte Template Engine von TYPO3 (**cObject TEMPLATE**) gilt mittlerweile als überholt und eignet sich nicht im Einsatz mit MVC, da keine Logiken, Schleifen, etc... abbildbar sind. Die Skalierbarkeit der Website ist hier deutlich eingeschränkt.

Um Fluid direkt in der TYPO3-Integration verwenden zu können, kann man das Content Object (**cObject FLUIDTEMPLATE**) verwenden.

```
Page = PAGE
Page {
    typeNum = 0
    10 = FLUIDTEMPLATE
    10 {
        templateName < lib.templateName
        templateRootPaths {
            0 = EXT:in2template/Resources/Private/Templates/Page/
        }
        partialRootPaths {
            0 = EXT:in2template/Resources/Private/Partials/Page/
        }
        layoutRootPaths {
            0 = EXT:in2template/Resources/Private/Layouts/Page/
        }
    }
    settings {
        baseUrl = ${PROTOCOL}${BASEURL}
        uid =
    }
}
```

*Tipp:* Achten Sie beim Schreiben von TypoScript auf saubere Einrückungen, um mögliche Fehlersuchen so einfach wie möglich zu machen.

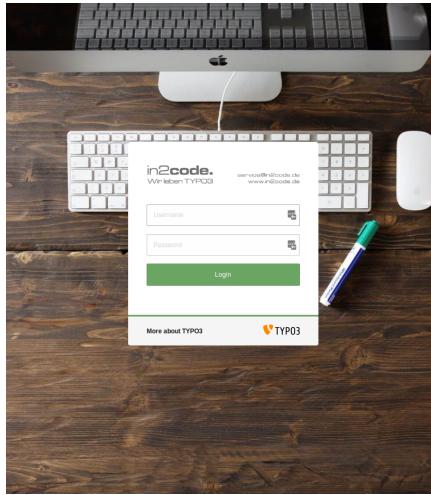
### 1.2.3 TypoScript

TypoScript ist eine Anweisungs- oder Konfigurationssprache, die TYPO3 (einmal abgesehen von TSconfig oder Sonderfälle mit Extbase Backend-Modulen) dazu anweist, wie es sich im Frontend zu verhalten hat. Hierbei handelt es sich um ein Hauptmerkmal von TYPO3.

Auf der einen Seite macht TypoScript TYPO3 extrem flexibel und erweiterbar, auf der anderen Seite stellt TS eine nicht zu unterschätzende Einstiegshürde dar.

Ähnliche Konfigurationen kennt man als XML oder YAML in vergleichbaren Systemen. Durch Conditions, if-Bedingungen und verschiedenen cObjects kann es passieren, dass man TypoScript als Scriptsprache interpretieren will. Dies ist jedoch falsch.

Mit Hilfe von TypoScript alleine lässt sich die Ausgabe einer Seite beliebig steuern. So können ganze Webseiten durch Änderung eines Parameters schnell zu einer XML-Ansicht werden.

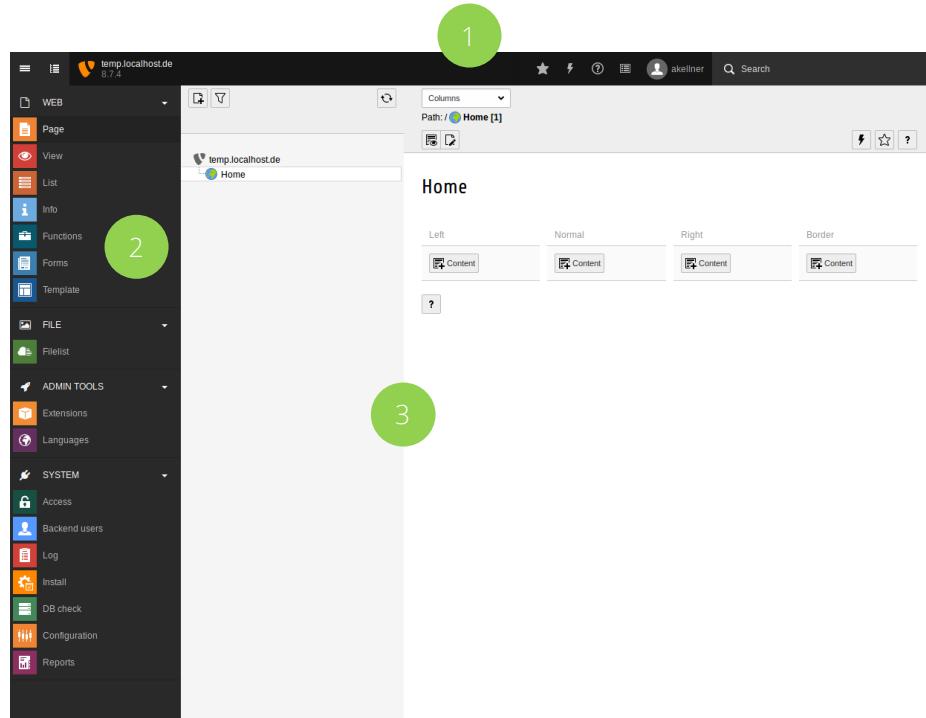


## 1.2.4 Frontend und Backend

Wenn Sie in der täglichen Benutzung Ihres Computers durch das Internet surfen, bekommen Sie in der Regel lediglich das Frontend verschiedener Seiten zu Gesicht. Das **Frontend** ist der Bereich einer Webseite, der die Informationen für die Besucher bereitstellt.

Jedoch haben viele Internetseiten, die in einem CMS laufen, eine geschützte Administrationsoberfläche. Im Fall von TYPO3 bezeichnet man diesen Administrationsbereich als **Backend**. Im Backend lassen sich u.a. Seiten- und Seiteninhalte erstellen, löschen oder verändern.

*Tipp: Das Backend lässt sich üblicherweise im Browser über [domainname.org/typo3](http://domainname.org/typo3) aufrufen*



## 1.2.5 Die Aufteilung im Backend

- 1 Der obere, dunkle Bereich ermöglicht einen direkten Zugriff auf Benutzereinstellungen, Suche, Shortcuts und weitere nützliche Grundfunktionen
- 2 Alle Links im linken, dunklen Bereich ermöglichen den Zugriff auf verschiedene Backend-Module
- 3 Der Bereich rechts zeigt die Übersicht aus dem gewählten Modul. Alle Module innerhalb vom Überpunkt „WEB“ zeigen einen Seitenbaum. Alle Module innerhalb von „FILE“ zeigen eine Baumstruktur des Dateisystems des Verzeichnisses „fileadmin“.



*Tipp:* Wird TYPO3 über composer installiert, befindet sich die Instanz im **composer-mode**. Wird TYPO3 über den herkömmlichen Weg installieren, spricht man vom **classic-mode**

## 1.2.6 Composer

Composer ist ein PHP-basierter Paketmanager, der es ermöglicht, gewünschte Pakete und deren Abhängigkeiten über die Kommandozeile zu installieren. Seit TYPO3 7.6 (bzw. 6.2) kann man Composer stabil nutzen, um TYPO3 und benötigte Erweiterungen einfach zu installieren.

Darüber hinaus kümmert sich der Composer um das Inkludieren von PHP-Dateien aus TYPO3 und diversen Erweiterungen.

Falls TYPO3 über Composer installiert wird, müssen auch Erweiterungen über Composer (nach-)installiert werden. Ein Einfügen über den Extension-Manager ist hierbei nicht möglich.

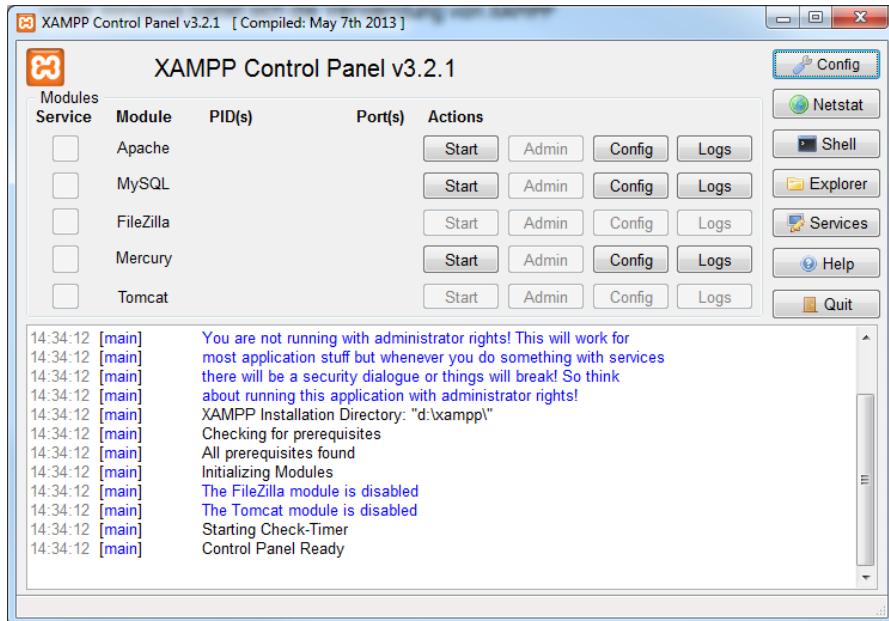


Abb. 1. Beispiel: XAMPP auf Windows 7

```

einpraegsam@zen: /var/www
einpraegsam@zen ➔ /var/www php -v
PHP 7.0.15-0ubuntu0.16.04.4 (cli) ( NTS )
Copyright (c) 1997-2017 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2017 Zend Technologies
with Zend OPcache v7.0.15-0ubuntu0.16.04.4, Copyright (c) 1999-2017, by Zend
Technologies
with Xdebug v2.4.1, Copyright (c) 2002-2016, by Derick Rethans
  
```

Abb. 2. Beispiel: PHP 7 unter Ubuntu

## 1.3 Installation

Eine lokale Testumgebung ermöglicht die Entwicklung einer Website ohne großen Zeitverlust – auch offline.

### 1.3.1 Voraussetzungen

Damit TYPO3 optimal funktioniert, sollte lokal (z.B.) ein Apache-Server, MySQL und PHP installiert sein (z.B. mit Hilfe von XAMPP – siehe <https://www.apachefriends.org/de/index.html>). Zusätzlich empfiehlt sich die Installation von Composer. Entsprechende VHost und hosts Einträge müssen gemacht werden, damit die Seite(n) später lokal unter einer bestimmten Domain verfügbar ist (z.B. soll beim Aufruf von [typo3.localhost.de](http://typo3.localhost.de) im Browser der lokale Apache angesprochen werden – siehe <http://wiki.schmidtmarcel.de/xampp-vhosts-einrichten/>). Darüber hinaus benötigt TYPO3 eine Datenbank, die man z.B. mit dem Programm PhpMyAdmin erstellen kann. Hierbei ist darauf zu achten, dass diese mit der Collation **utf8-general-ci** angelegt wird, um spätere Probleme mit Umlauten oder Sonderzeichen zu umgehen.

### 1.3.2 Installation von TYPO3

Über Composer lässt sich TYPO3 herunterladen und installieren. In einem Verzeichnis der Wahl lässt sich mit Hilfe des Befehls „**composer require typo3/cms**“ die aktuellste Version von TYPO3 mit allen Abhängigkeiten zusammenbauen (Build Process).

Über das Anlegen einer leeren Datei „FIRST\_INSTALL“ im Web-Root, lässt sich das Install-Tool freischalten. In einem Unix-System geht das mit dem Befehl „touch FIRST\_INSTALL“.

Das Install-Tool lässt sich nun im Browser über [typo3.localhost.de/typo3/install/](http://typo3.localhost.de/typo3/install/) öffnen und führt einen durch die Grundkonfiguration (Datenbankzugang, Backendzugang, etc...). Im Anschluss kann man sich wie gewohnt ins Backend über [typo3.localhost.de/typo3/](http://typo3.localhost.de/typo3/) mit den zuvor vergebenen Nutzerdaten einloggen.

Wer es schneller mag, kann mit den folgenden drei Befehlen innerhalb weniger Minuten ein komplettes TYPO3-Projekt erstellen:

```
composer create-project helhum/minimal-typo3-distribution my-project ^8
cd my-project
vendor/bin/typo3cms install:setup
```

*Tipp:* Oberhalb des Seitenbaums befindet sich sein +-Symbol, das bei Klick verschiedene Seitentypen zum Ziehen per Drag and Drop anbietet.



Welcome to a default website made with **TYPO3**

Wurde „Yes, create a base empty page to start from“ gewählt, wird bereits eine Seite im Seitenbaum erzeugt, die sich auch im Frontend ansehen lässt. Im anderen Fall ist TYPO3 noch komplett nackt und muss mit Seiten und Konfiguration ergänzt werden.

Ausgehend von einer leeren Installation sollte nun eine neue Seite im Seitenbaum erzeugt werden.

Über das Backend-Modul „Template“ lässt sich bequem TypoScript-Konfiguration auf der neuen Seite einbringen. Zuerst muss jedoch im Select von „Constant Editor“ auf „Info/Modify“ gewechselt werden. Ein Button „Create template for a new site“ ermöglicht das Einfügen eines TypoScript Datensatzes, das dann wiederum durch Klick auf „Edit the whole template record“ zum Bearbeiten geöffnet werden kann.

TYPO3 hat bereits ein kleines Beispiel TypoScript im Feld „Setup“ eingefügt:

```
# Default PAGE object:  
page = PAGE  
page.10 = TEXT  
page.10.value = HELLO WORLD!
```

Dies erzeugt bereits im Frontend die Ausgabe „HELLO WORLD!“

### 1.3.3 Best practice

#### 1.3.3.1 AdditionalConfiguration

Die Grundkonfiguration von TYPO3 befindet sich in typo3conf/LocalConfiguration.php und wird bei Änderungen im Install Tool automatisch neu erzeugt. Eigene Konfiguration sollte sich unter typo3conf/AdditionalConfiguration.php befinden. Einträge, die sich auch in der anderen Konfiguration befinden, werden hierbei überschrieben.

Wir empfehlen ein Verzeichnis typo3conf/ext/AdditionalConfiguration/ anzulegen. Danach diesen Eintrag in der typo3conf/ext/AdditionalConfiguration.php

```
<?php  
foreach (glob(__DIR__ .  
    '/AdditionalConfiguration/*Configuration.php') as  
$configFile) {  
    include($configFile);  
}
```

Damit werden automatisch alle Dateien im Verzeichnis AdditionalConfiguraiton/ mit dem Aufbau \*Configuration.php inkludiert. Somit lassen sich verschiedene Konfigurationsdateien (wie z.B.

DatabaseConfiguration.php, ImageConfiguration.php oder DevelopmentConfiguration.php) anlegen, die nicht zwingend auf einem späteren Live-Server zum Einsatz kommen.

Die Dateien im Verzeichnis „AdditionalConfiguration“ sollten von der Versionierung ausgenommen werden.

/web/typo3conf/AdditionalConfiguration/

### 1.3.3.2 DevelopmentConfiguration

Über eine typo3conf/ext/AdditionalConfiguration/DevelopmentConfiguration.php können Caches deaktiviert, Fehlermeldungen aktiviert und weitere Konfigurationen gesetzt werden. Nachfolgend ein paar nützliche Zeilen für eine lokale Entwicklungsumgebung.

```
<?php
// PASSWORDS
$saltFactory = \TYPO3\CMS\Saltedpasswords\Salt\SaltFactory::getSaltingInstance();
$GLOBALS['TYPO3_CONF_VARS']['BE']['installToolPassword'] = $saltFactory->getHashedPassword('akellner');

// MISC
$GLOBALS['TYPO3_CONF_VARS']['BE']['sessionTimeout'] = 9999999999;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['enableDeprecationLog'] = '1';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['curlUse'] = 1;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['curlTimeout'] = 10;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['trustedHostsPattern'] = '.*\.\localhost\.de';
```

```
// GRAPHICS
$GLOBALS['TYPO3_CONF_VARS']['GFX']['im_path'] = '/usr/bin/';
$GLOBALS['TYPO3_CONF_VARS']['GFX']['im_path_lzw'] = '/usr/bin/';
$GLOBALS['TYPO3_CONF_VARS']['GFX']['im_version_5'] = 'gm';
$GLOBALS['TYPO3_CONF_VARS']['GFX']['colorspace'] = 'RGB';
```

```
\TYPO3\CMS\Core\Utility\ExtensionManagementUtility::addTypoScript(
    'developmentConfiguration',
    'setup',
    'config.contentObjectExceptionHandler = 0'
);
```

```
// DEBUG
$GLOBALS['TYPO3_CONF_VARS']['BE']['debug'] = 1;
$GLOBALS['TYPO3_CONF_VARS']['BE']['compressionLevel'] = 0;
$GLOBALS['TYPO3_CONF_VARS']['FE']['debug'] = 1;
//GLOBALS['TYPO3_CONF_VARS']['FE']['pageNotFound_handling'] = '';
$GLOBALS['TYPO3_CONF_VARS']['FE']['compressionLevel'] = 0;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['displayErrors'] = '1';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['devIPmask'] = '*';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['sqlDebug'] = '1';
```

```
// CACHE
$GLOBALS['TYPO3_CONF_VARS']['SYS']['clearCacheSystem'] = '1';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfigurations']['cache_hash']['backend'] = 'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfigurations']['cache_pages']['backend'] = 'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfigurations']['cache_pagesection']['backend'] = 'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfigurations']['cache_phpcode']['backend'] = 'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfigurations']['cache_rootline']['backend'] = 'TYPO3\CMS\Core\Cache\Backend\NullBackend';
```

```
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfigurations']['extbase_datamapfactory_datamap']['backend'] = 'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfigurations']['extbase_object']['backend'] = 'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfigurations']['extbase_reflection']['backend'] = 'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfigurations']['extbase_typo3dbbackend_queries']['backend'] = 'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfigurations']['extbase_typo3dbbackend_tablecolumns']['backend'] = 'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfigurations']['l10n']['backend'] = 'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['EXT']['extCache'] = '0';
```

*Hinweis:* Wenn mehrere TYPO3-Umgebungen lokal im Einsatz sind, kann die DevelopmentConfiguration.php überall „versymlinkt“ werden.

```
// MAIL
$GLOBALS['TYPO3_CONF_VARS']['EXT']['powermailDevelopContextEmail'] =
'alexander.kellner@in2code.de';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['defaultMailFromAddress'] = 'alex@in2code.de';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['defaultMailFromName'] = 'defaultmailfromaddress';

/*
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport'] = 'smtp';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_server'] = 'sslout.df.eu:465';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_encrypt'] = 'ssl';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_username'] =
'lokal@einpraegsam.net';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_password'] = 'password';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_port'] = '465';
*/

$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport'] = 'mbox';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_mbox_file'] = '/var/www/html/mbox.txt';
```

## 1.4 Konfiguration mit Hilfe von TypoScript

### 1.4.1 Einführung

Die Konfigurationssprache, die TYPO3 verwendet um das Frontend zu rendern, nennt sich TypoScript. Üblicherweise wird auf der obersten Seite ein TypoScript-Template eingefügt. Diese Konfiguration vererbt sich automatisch auf alle Kindseiten (so lange bis die Konfiguration überschrieben wird).

Man kann sich TypoScript wie ein Array mit Anweisungen für ein System vorstellen. Abgesehen von ein paar dynamischen Befehlen ist TypoScript keine Programmier- oder Scriptssprache sondern lediglich eine extrem flexible Anweisungssprache.

Diese Konfigurationssprache ist eines der Hauptmerkmale bei der Unterscheidung von TYPO3 zu anderen CMS.

TypoScript ist der Grund, der es auf der einen Seite so schwer ist, den administrativen Umgang mit TYPO3 zu erlernen und auf der anderen Seite TYPO3 extrem flexibel und anpassbar macht.

Das System ist mit TypoScript so flexibel, dass die Ausgabe nicht zwingend eine Webseite sein muss, sondern auch ein XML oder ein PDF oder etwas ganz Anderes sein kann.

```
119. ### [Begin] Canonical Duplicate-Content Vermeidung ###
120. lib.canonical = COA
121. lib.canonical {
122. #Eckige Klammern mit spitzen Tag-Klammern ersetzen:
123. wrap = <link rel="canonical" href="#" />
124.   1 = TEXT
125.     1 {
126.       value < config.baseURL
127.       wrap =
128.
129.
130.   }
131.   2 = TEXT
132.     2.typolink {
133.       parameter = {page:uid}
134.       parameter.insertData = 1
135.       useCacheHash = 1
136.       # add all get parameters from the current URL
137.       addQueryString = 0
138.       addQueryString.method = GET
139.       # remove the page id from the parameters so it is not inserted twice
140.       addQueryString.exclude = id
141.       returnLast = url
142.     }
143.   }
144. }
145. page.headerData.200 < lib.canonical
146. ### [END] Canonical Duplicate-Content Vermeidung ###
147.
```

Abb. 3. Beispiel TypoScript

## 1.4.2 Hello World – die erste Ausgabe

Wenn Sie TYPO3 zum ersten Mal installiert und sich gegen das Introduction Package entschieden haben, werden Sie beim Aufruf der Seite im Frontend eine Fehlermeldung erhalten.

Das liegt daran, dass Sie noch keine Anweisungen (TypoScript) für TYPO3 hinterlegt haben, wie dieses sich im Frontend verhalten soll.



Abb. 4. Aufruf einer TYPO3-Seite im Frontend ohne TypoScript Konfiguration

## 1.4.2.1 Ersten Seitenbaum erstellen

Erstellen Sie in Ihrem Seitenbaum eine neue Startseite und eventuell eine Hand voll Unterseiten, mit denen Sie beginnen können (wie Sie Seiten im Backend anlegen können, entnehmen Sie bitte dem Abschnitt **Backend**).

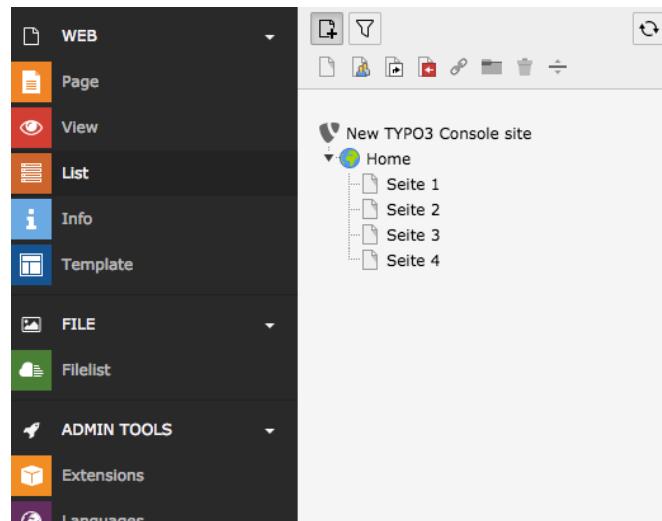
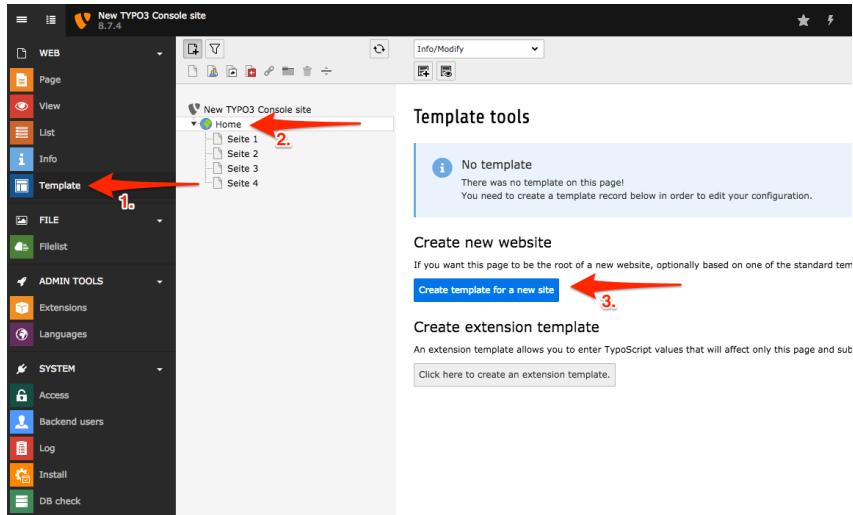


Abb. 5. Startseite mit vier Unterseiten



*Hinweis Template: Etwas verwirrend ist die Bezeichnung **Template** im Backend von TYPO3. Hier müssen Sie zwischen einem **HTML-Template** und einem **Typoscript-Template** unterscheiden. Ersteres definiert das Layout-Gerüst, während Letzteres Anweisungen für TYPO3 bereithält.*

*Das Backend-Modul **Template** bezieht sich auf den Begriff „**Typoscript-Template**“*

#### 1.4.2.2 Erstellung eines ersten Typoscript Templates

Nun wird es Zeit, ein erstes Typoscript Template anzulegen, in das wir die Anweisungen für das Frontend-Rendering platzieren können.

Erstellen Sie über das Backend-Modul „Template“ ein neues Typoscript Template in dem Sie das Modul anklicken (1), danach Ihre Startseite (2) auswählen und dann den Button „Template für neue Website erstellen“ (3) klicken.

Im Anschluss und bei einem Seiten-Reload sehen Sie im rechten Bereich, dass ein Template erstellt wurde.



## Template-Werkzeuge

Konstanten bearbeiten

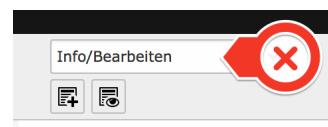
 NEW SITE



Abb. 6. Neu erstelltes Template: Noch keine Konstanten verfügbar

Wechseln Sie im Dropdown vom „Konstanten-Editor“ auf „Info/Bearbeiten“.

Im Anschluss können Sie einzelne Felder bearbeiten oder die Bearbeitungsmaske aller Felder öffnen.



**Hinweis TypoScript Editor:** Wenn Sie später lediglich das TypoScript-Setup bearbeiten wollen, können Sie an dieser Stelle direkt das Feld Setup auswählen. Das bietet Ihnen den Vorteil eines TypoScript Editors mit Code-Vervollständigung.

Bitte klicken Sie auf „Vollständigen Template-Datensatz bearbeiten“.

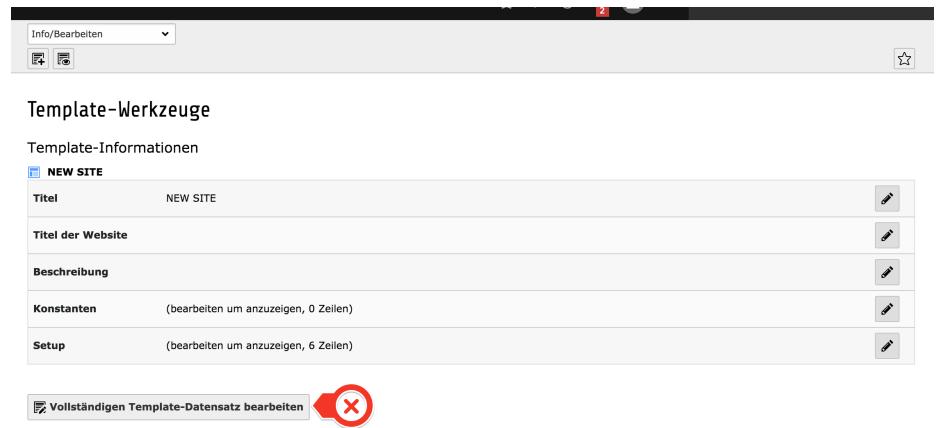


Abb. 7. TypoScript Template im Backend Modul Template

Im nachfolgenden Fenster sehen Sie, dass TYPO3 bereits ein paar Zeilen TypoScript (Feld Setup) für Sie angelegt hat.

The screenshot shows the TYPO3 Backend interface for editing a template. The title bar says "Template 'NEW SITE' auf Seite 'Home' bearbeiten". The top menu has items like "Speichern", "Enthält", "Zugriff", and "Hinweise". Below the title, there are tabs for "Allgemein", "Optionen", "Enthält", "Zugriff", and "Hinweise".

**Template-Titel:**  Verstellbare Zeichen: 243

**Website-Titel:**

**Konstanten:**

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
sys_template > constants

```

**Setup:**

```

1 # Default PAGE object:
2 page = PAGE
3 page.10 = TEXT
4 page.10.value = HELLO WORLD!
5
6
7
8
9
10
11
12
13
14
15
16
sys_template > config

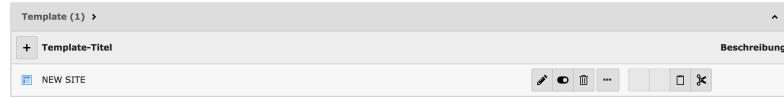
```

At the bottom right of the window, it says "Template [2]".

Abb. 8. Template Datensatz Bearbeitungsmaske

## HELLO WORLD!

Hinweis zur Vererbung: Alle Ihre kommenden Einstellungen in dem Template, das sich in der Seite „Start“ befindet, wird an die Unterseiten vererbt. Wünschen Sie auf einer bestimmten Unterseite eine andere Einstellung, so müssen Sie auf dieser Seite auch ein TypoScript Template einbinden und die gewünschte Stelle wieder überschreiben.



#### 1.4.2.3 Felderklärung im TypoScript Template

Wie im letzten Abschnitt können Sie ein neues Template auf einer Seite anlegen.

Um ein vorhandenes Template zu öffnen, haben Sie zwei Möglichkeiten:

- Nutzen Sie das Backend-Modul Template und wählen Sie die betroffene Seite
- Oder nutzen Sie das Backend-Modul Liste, wählen Sie die betroffene Seite und dann können Sie den Datensatz bearbeiten, den Sie öffnen wollen.

#### Template "NEW SITE" auf Seite "Home" bearbeiten

The screenshot shows the TYPO3 Backend interface for managing a template. The top navigation bar includes 'Allgemein', 'Optionen', 'Enthält', 'Zugriff', and 'Hinweise'. The main content area has three tabs: 'Template-Titel', 'Website-Titel', and 'Konstanten'. The 'Template-Titel' tab contains a field with the value 'NEW SITE'. The 'Website-Titel' tab is empty. The 'Konstanten' tab displays a list of constants from line 1 to 16, with the last entry being 'sys\_template > constants'. The 'Setup' tab at the bottom contains a code editor with the following TypoScript setup:

```
1 # Default PAGE object:  
2 page = PAGE  
3 page.10 = TEXT  
4 page.10.value = HELLO WORLD  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16
```

A small blue icon with the text 'Template [2]' is located at the bottom right of the main content area.

#### Reiter „Allgemein“

**Template-Titel:** Vergeben Sie einen sinnvollen Titel für Ihr Template, damit Sie dieses von anderen Templates unterscheiden können – z.B. „domain.de Main Template“

*Hinweis: Denken Sie daran, dass Ihre Seite im Laufe der Zeit wachsen wird und verwenden Sie von Beginn an eindeutige Begrifflichkeiten und Beschreibungen.*

**Website-Titel:** Wenn Sie im Frontend bereits einen statischen Website-Titel (Title-Tag im Head-Bereich Ihres HTML) vergeben wollen, können Sie dieses Feld befüllen.

*Hinweis: Sie können den Website-Title selbstverständlich auch dynamisch mit TypoScript zusammenbauen. In diesem Fall können Sie das Feld leer lassen.*

**Konstanten:** In diesem Feld können Sie konstante Werte speichern. Im Setup kann man auf diese Werte wieder zugreifen (siehe Verwendung der TypoScript Konstanten).

**Setup:** Dieses Feld ist das wichtigste Feld, denn hier fügen Sie Ihr TypoScript ein.

## Template "NEW SITE" auf Seite "Home" bearbeiten

The screenshot shows the TYPO3 Backend interface for editing a template. The top navigation bar has tabs: Allgemein, Optionen (which is selected), Enthält, Zugriff, and Hinweise. Below the tabs, there are sections for 'Löschen:' (Delete) with checkboxes for 'Constants' and 'Setup'. A section for 'Wurzelebene:' (Root level) has a checkbox. Under 'Template nächste Ebene:' (Template next level), there is a search bar 'Datensätze suchen' and a list with one item 'Template [2]'. At the bottom right of the main area, there is a small blue icon with the text 'Template'.

### Reiter „Optionen“

**Löschen Constants:** Falls sich Ihr Template auf einer Unterseite befindet und Sie alle zuvor gesetzten Konstanten nicht auf diese Seite vererbt haben wollen, können Sie diese hiermit leeren.

**Löschen Setup:** Falls sich Ihr Template auf einer Unterseite befindet und Sie alle zuvor gesetzten Werte im Setup nicht auf diese Seite vererbt haben wollen, können Sie diese hiermit leeren.

**Wurzelebene:** Handelt es sich bei diesem TypoScript Template um ein Master-Template, also ein Template auf der obersten Seite, so können Sie dies TYPO3 an dieser Stelle mitteilen. TYPO3 weiß damit, dass es sich hier um eine Einstiegsseite handeln muss.

**Template nächste Ebene:** Wenn Sie auf allen Unterseiten ein anderes Template einbinden wollen, können Sie hier einen TypoScript Datensatz suchen und auswählen.

*Hinweis: Diese Einstellung kommt oft vor, wenn Sie eine Intro-Page nutzen, die komplett anders aussieht als alle Folgeseiten.*

## Template "NEW SITE" auf Seite "Home" bearbeiten

The screenshot shows the TYPO3 Backend interface for editing a template. The top navigation bar includes 'Allgemein', 'Optionen', 'Enthält' (selected), 'Zugriff', and 'Hinweise'. The main content area has several sections:

- Statische Templates nach Basis-Template einschließen:** A list box containing a single entry with a delete icon.
- Statische Templates einschließen (aus Erweiterungen):** A section titled 'Ausgewählte Objekte' showing a list of Fluid Content Elements (fluid\_styled\_content) and Fluid Content Elements CSS (optional) (fluid\_styled\_content). To the right is a 'Verfügbare Objekte' sidebar with a search bar and a list of available objects.
- Basis-Template einschließen:** A search bar labeled 'Datensätze suchen' and a list of templates with a 'Template' button.
- Statische Template-Dateien aus Erweiterungen:** A dropdown menu set to 'Standard (einschließen vor, wenn Wurzel-Option gesetzt)'.

### Reiter „Enthält“

**Statische Templates nach Basis-Template einschließen:** Statische Templates (nächster Punkt) werden in der Regel vor dem Basis-Template eingebunden. Wenn Sie die Reihenfolge ändern wollen, können Sie diese Einstellung wählen.

**Statische Templates einschließen (aus Erweiterungen):** Viele TYPO3-Erweiterungen bringen Ihre eigene TypoScript-Konfiguration mit, um das Frontend anzupassen. Hier können Sie dieses TypoScript zu- oder wegklicken.

**Basis-Template einschließen:** Wenn Sie Templates in Snippets auslagern wollen, können Sie die Snippets hier einbinden.

**Statische Template-Dateien aus Erweiterungen:** Hier bestimmen Sie die Reihenfolge der Einbindung zwischen Basis- und Static-Templates.

## Template "NEW SITE" auf Seite "Home" bearbeiten

Allgemein Optionen Enthält Zugriff Hinweise

**Inaktiv:**

**Start:**

**Stop:**

**Template [2]**

### Zugriff

**Inaktiv:** De-/Aktivieren Sie ein TypoScript Template

**Start:** Ähnlich wie in anderen Datensätzen, können Sie auch für Templates ein Datum vergeben, ab wann das Template aktiv sein soll.

**Stop:** Ähnlich wie in anderen Datensätzen, können Sie auch für Templates ein Datum vergeben, ab wann das Template nicht mehr aktiv sein soll.

*Hinweis: Start- und Stop-Zeiten machen vor allem Sinn, wenn Sie sich an die Regel halten, ausschließlich kleine TypoScript Templates zu verwenden (siehe Template auslagern in Snippets).*

## Template "NEW SITE" auf Seite "Home" bearbeiten

Allgemein Optionen Enthält Zugriff Hinweise

**Beschreibung:**

**Template [2]**

### Hinweise

**Beschreibung:** Dieses Feld dient der internen Beschreibung und bringt somit keinerlei Frontend- oder Backend-Funktion mit.

## 1.4.3 TypoScript Grundlagen

### 1.1.1 Einführung

Wie im Bereich über die Erstellung von TypoScript Templates (siehe Erstellung eines ersten TypoScript Templates) beschrieben, erzeugt TYPO3 bereits ein paar Zeilen TypoScript im Feld Setup, damit Sie im Frontend bereits eine erste „Hello World“ Ausgabe betrachten können.

Jede Zeile steht für eine Anweisung. Eine Anweisung muss nicht, wie in anderen Sprachen üblich, mit einem sichtbaren Zeichen beendet werden.

Alle TypoScript Objekte und Attribute finden Sie in der TypoScript Referenz (TSref <https://docs.typo3.org/typo3cms/TyposcriptReference/>). Die TSref ist mehr als Nachschlagewerk denn als Tutorial anzusehen.

```
# Default PAGE object:  
page = PAGE  
page {  
    10 = TEXT  
    10.value = HELLO WORLD!  
}
```

#### Erste Zeile

In der ersten Zeile wird ein Kommentar eingefügt. Dass dieser nicht als Anweisung von TYPO3 interpretiert wird, erkennen Sie an der führenden Raute „#“.

#### Zweite Zeile

Hierbei handelt es sich um die erste Zeile, die auch von TYPO3 interpretiert wird. Vor dem Operator „=“ befindet sich ein beliebiger Variablenname. „page“ hat sich in TYPO3 durchgesetzt, weil dieser Variablenname auf das Haupt- und Seitenobjekt schließen lässt. Hinter dem Operator kommt die Zuweisung. In diesem Fall handelt es sich bei der Variable „page“ um ein TypoScript Objekt vom Typ „PAGE“. PAGE-Objekte sind die Basis-Objekte, damit eine Seitenausgabe erfolgen kann. Sie beinhalten bereits einen generierten HTML-Header mit Informationen (z.B. über den Doctype, etc...).

*Hinweis: Objektzuweisungen müssen immer GROßgeschrieben werden.*

### Dritte Zeile

Durch den Zusatz „10“ wird die Variable um ein Unterobjekt erweitert. Dieses ist vom Objekt-Typ „TEXT“. TEXT-Objekte bezeichnen die Ausgabe von Text – hier kann auch bereits HTML zurückgegeben werden.

```
# Default PAGE object:  
page = PAGE  
page {  
    10 = TEXT  
    10.value = HELLO WORLD!  
}  
/*Dies ist ein Kommentar  
mit zwei  
drei  
vier Zeilen  
*/
```

### Vierte Zeile

Durch den Zusatz „value“ bekommt die Variable ein Attribut vom Typ „value“ zugewiesen. Nach dem Operator wird nun ein statischer Text erwartet. In unserem Fall „HELLO WORLD!“.

*Hinweis Kommentare: Mehrzeilige Kommentare werden mit „/\*“ begonnen und am Anfang der letzten Zeile mit „\*/“ abgeschlossen.*

Eine alternative Schreibweise ist die Verwendung von geschweiften Klammern, um sich Zeichen zu sparen.

*Tipp:* Gewöhnen Sie sich an, bei dieser Schreibweise, die Texte richtig einzurücken, damit Sie auch bei größeren Templates die Übersicht nicht verlieren.

### 1.4.3.1 TEXT

Ähnlich wie im letzten Abschnitt beschrieben, sorgt das TypoScript Objekt vom Typ „TEXT“ für die Rückgabe von Text. Natürlich kann der Text (String) bereits HTML-Tags beinhalten.

**Beispiel HTML-Tags 1:** Ausgabe eines statischen Textes mit HTML-Tags.

```
page = PAGE
page {
  10 = TEXT
  10.value = <h1>HALLO <br /> WELT!</h1>
}
```

**Beispiel HTML-Tags 2:** Ausgabe eines statischen Textes mit HTML-Tags eines Bildes (In unserem Beispiel befindet sich im Verzeichnis fileadmin ein Bild mit dem Namen „bild.jpg“ von einem Dalmatiner)

```
# Default PAGE object:
page = PAGE
page {
  10 = TEXT
  10.value = 
}
```

### 1.4.3.2 IMAGE

Zwar lassen sich auch HTML-Tags und damit auch Bilder mit dem TypoScript Objekt „TEXT“ ausgeben. TypoScript bietet Ihnen jedoch auch die Möglichkeit Image-Tags für Bilder für Sie zusammenzubauen.

Der Vorteil hier liegt auf der Hand: Wenn das Bild nur klein benötigt wird, obwohl es in einer hohen Auflösung vorhanden ist, wird automatisch eine optimierte Kopie des Bildes erstellt. Damit reduzieren Sie die Ladezeiten im Frontend.

#### Beispiel Bildausgabe 1: Ausgabe eines von TYPO3 gerenderten Bildes

(In unserem Beispiel befindet sich im Verzeichnis fileadmin ein Bild mit dem Namen „bild.jpg“ von einem Dalmatiner). Die Attribute width, height, border und alt werden automatisch hinzugefügt, auch wenn Sie diese nicht definiert haben.

```
page = PAGE
page {
  10 = IMAGE
  10.file = fileadmin/bild.jpg
}
```

```

```

#### Beispiel Bildausgabe 2: Ausgabe eines Bildes mit weiteren Attributen

```
page = PAGE
page {
  10 = IMAGE
  10 {
    file.width = 100
    file.height = 80
    altText = Dalmatiner
    titleText = Hier sehen Sie einen schönen Dalmatiner
    params = class="picture"
  }
}
```

### 1.4.3.3 COA / COA\_INT

COA steht für Content Object Array und bedeutet, dass Sie unendlich Objekte miteinander verschachteln können.

*Hinweis Cache: Wenn Sie das Objekt COA\_INT anstelle COA verwenden, wird die Ausgabe nicht gecacht und jedes Mal neu gerendert. Das kann dann Sinn machen, wenn sich die Ausgabe aus irgendeinem Grund bei jedem Aufruf ändern soll (weil diese z.B. an einen Frontend-Benutzer oder an eine Uhrzeit geknüpft ist).*

#### Beispiel COA 1: Verknüpfung von mehreren Elementen

```
page = PAGE
page {
    10 = COA
    10 {
        10 = TEXT
        10.value = Hallo
        10.wrap = |<br />
        20 = TEXT
        20.value = Welt
    }
}
```

Hallo<br />Welt

*Hinweis: Das Attribut wrap hilft einen Wert mit einem Text oder HTML-Tags (vor und nach dem Symbol „|“) zu.*

**Beispiel COA 2:** Verknüpfung von mehreren Elementen auf mehreren Ebenen.

```
page = PAGE
page {
    10 = COA
    10 {
        10 = TEXT
        10.value = Dalmatiner:
        20 = IMAGE
        20.file = fileadmin/bild.jpg
        30 = COA
        30 {
            10 = TEXT
            10.value = (Pongo)
        }
    }
}
```

Dalmatiner:(Pongo)

#### 1.4.3.4 CONTENT

Mit dem TypoScript Objekt CONTENT lassen sich Werte aus der Datenbank lesen.

**Hinweis:** Die einzigen zugelassenen Tabellen sind "pages" oder Tabellen mit dem Präfix "tt\_","tx\_","tx\_","fe\_" oder "user\_". Diese Voraussetzung dient der Sicherheit.

**Beispiel CONTENT 1:** Lese Überschrift aus Seiteninhalt mit der UID 1  
(innerhalb der Seite mit der PID 2)

```
page = PAGE
page {
    10 = CONTENT
    10 {
        table = tt_content
        select {
            pidInList = 2
            where = uid = 1
        }
        renderObj = TEXT
        renderObj {
            field = header
        }
    }
}
```

**Beispiel CONTENT 2:** Lese Name aus der Tabelle tt\_address vom Datensatz, dessen UID im GET Parameter vorgegeben wurde (Aufruf mit index.php?id=1&wert=123)

```
lib.nameOfCurUser = CONTENT
lib.nameOfCurUser {
    table = tt_address
    select {
        pidInList = 2
        where {
            data = GP:wert
            wrap = tt_address.uid=|
            intval = 1
        }
    }
    renderObj = COA
    renderObj.10 = TEXT
    renderObj.10 {
        field = name
    }
}
```

**Hinweis 1:**  
intval forciert den übergeben Wert als Integer und beugt somit einer SQL-Injection Lücke vor.

**Hinweis 2:**  
pidInList ist die Page ID in der sich die Adress-Datensätze befinden.

#### 1.4.3.5 USER / USER\_INT

Wenn Sie einmal mit TypoScript nicht weiterkommen und ein ganz spezielles Objekt benötigen, hilft Ihnen vermutlich nur noch eine userFunc weiter. Mit einer userFunc können Sie ein PHP-Script einbinden und die Ausgabe im TypoScript nutzen.

Ähnlich wie bei den COA-Objekten (siehe COA / COA\_INT) können Sie mit der Verwendung von USER\_INT forcieren, dass TYPO3 den Rückgabewert nicht cacht.

Wir zeigen hier nur zwei einfache Beispiele, da der direkte Einsatz mit PHP nicht für alle Administratoren interessant ist.

**Beispiel USER 1:** Aufruf einer einfachen PHP Funktion.

```
lib.doSomething = USER
lib.doSomething {
    userFunc = MyNameSpace\MyUserFuncs\HelloWorld->print
}
```

Die Klasse, die die Funktion enthält, muss in der Datei „composer.json“ im Bereich „autoload“ registriert werden.

```
{
    "repositories": [
        ...
    ],
    "require": {
        "typo3/cms": "8.7.*",
        ...
    },
    "autoload": {
        "psr-4": {
            "MyNameSpace\\MyUserFuncs\\": "fileadmin/php"
        }
    }
}

<?php
namespace MyNameSpace\MyUserFuncs;

/**
 * HelloWorld
 */
class HelloWorld
{

    /**
     * @param string $content
     * @param array $configuration
     * @return string
     */
    public function print(string $content = '', array $configuration = []): string
    {
        return 'Hello World';
    }
}
```

Datei “fileadmin/php/HelloWorld.php”

In einem letzten Schritt muß php noch mitgeteilt werden, dass diese Klasse geladen werden soll.

```
composer dumpautoload
```

**Beispiel USER 2:** Aufruf einer PHP Funktion mit der Übergabe von Werten im TypoScript.

```
lib.doSomething = USER
lib.doSomething {
    userFunc = MyNameSpace\MyUserFuncs\HelloWorld->print2
    irgendeinwert = Das ist jetzt individuell
}
}
```

```
<?php
namespace MyNameSpace\MyUserFuncs;

/**
 * HelloWorld
 */
class HelloWorld
{

    /**
     * @param string $content
     * @param array $configuration
     * @return string
     */
    public function print2(string $content = '', array $configuration = []): string
    {
        return $configuration['irgendeinwert'];
    }
}
```

Hier übergeben wir der Funktion einen erfundenen Key „irgendeinwert“ und ein Wert mit „Hallo Welt“

```
Das ist jetzt individuell
```

#### 1.4.3.6 Verwendung der TypoScript Konstanten

Während sich im TypoScript Setup alle Anweisungen für das Rendern des Frontends befinden, sammeln sich wichtige Werte in den TypoScript Konstanten. Dies wurde eingeführt, um die konstanten Werte einfach austauschen zu können, ohne das komplette Setup durchsuchen zu müssen.

Einmal definierte Werte in den Konstanten können mit {\$name} im Setup wieder aufgerufen werden.

**Beispiel Konstanten:** Nutzung von Werten aus den Constants im Setup.

```
# TypoScript Constants
farbe = red
```

```
# TypoScript Setup
page = PAGE
page {
    10 = TEXT
    10 {
        value = Dies ist der erste Inhalt
        wrap = <div class="content {$farbe}">|</div>
    }
}
```

```
<div class="content red">Dies ist der erste Inhalt</div>
```

#### 1.4.3.7 Kopieren von Elementen

Man kann im TypoScript auf bereits definierte Objekte zugreifen und diese kopieren oder gar referenzieren. Dies kann der Übersichtlichkeit oder Pflegbarkeit entgegenkommen.

- Der Operator < kopiert Elemente
- Der Operator =< referenziert Elemente
- Der Operator > leert Elemente

**Beispiel Kopieren von Elementen:** Elemente werden an einer Stelle definiert und an einer anderen eingefügt.

```
lib.text = TEXT
lib.text.value = Hallo Welt!

page = PAGE
page {
    10 = TEXT
    10.value = <h1>Servus</h1>

    20 < lib.text

    30 = TEXT
    30.value = <br />Bye bye!
}
```

```
<h1>Servus</h1>Hallo Welt <br />Bye bye!
```

*Wichtig ist hierbei, dass das Objekt lib.text vor dem page Objekt definiert wurde, weil TYPO3 das TypoScript Zeile für Zeile abarbeitet.*

**Beispiel Kopieren von Elementen:** Komplizierteres Beispiel

```
page = PAGE
page {
    10 = TEXT
    10 {
        Value = Hallo Welt
    }
    20 < page.10
    20.wrap = <br />|
```

Alternative zu Zeile 7 ist das Kopieren eines Elementes auf gleichen Level mit  
20 <.10

```
Hallo Welt <br />Hallo Welt
```

#### 1.4.3.8 „styles.content.get“ – Das Geheimnis

Den Inhalt des TypoScript Objektes styles.content.get kann man über die Template Analyse betrachten

```
styles.content.get = CONTENT
styles.content.get {
    table = tt_content
    select.orderBy = sorting
    select.where = colPos=0
    select.languageField = sys_language_uid
}
```

Bei dem TypoScript Objekt styles.content.get handelt es sich um nichts anderes als um ein Objekt vom Typ CONTENT und spart uns die Eingabe der aufgeführten 7 Zeilen.

TYPO3 stellt „ab Werk“ bis TYPO3 8LTS bereits vier Spalten zur Verfügung. Die Bezeichnungen sind „Normal“, „Links“, „Rechts“ und „Rand“. Jede Spalte hat eine feste „Spaltennummer“ zugeteilt (aka colPos). Die colPos sind wie folgt:

- Normal = 0
- Links = 1
- Rechts = 2
- Rand = 3

Es können aber beliebige viele Spalten angelegt und über sog. Backend-Layouts verwendet werden.

## 2 TypoScript – Templating

### 2.1 Assets einbinden

Assets sind Elemente, die gebraucht werden, damit die Website letztendlich so aussieht und funktioniert, wie sie soll. Dazu gehören z.B. Bilder, CSS- oder Javascript – Dateien.

#### 2.1.1 Einbindung von CSS – Dateien

CSS – Dateien kann man über TypoScript einfach einbinden.

```
page.stylesheet = fileadmin/stylesheet.css
```

Normalerweise benötigt man aber mehrere Dateien. Dafür steht die Eigenschaft „includeCSS“ zur Verfügung.

```
...  
page.includeCSS.beliebigeBezeichnung1 = fileadmin/css1.css  
page.includeCSS.beliebigeBezeichnung2 = fileadmin/css2.css  
...
```

```
page.includeCSS.print1 = fileadmin/print1.css  
page.includeCSS.print1.media = print
```

#### 2.1.2 Einbindung von Javascript – Dateien

JS Dateien kann man über TypoScript einfach einbinden.

```
page.includeJSlibs.jquery =  
https://ajax.googleapis.com/ajax/libs/jquery/1.8/jquery.min.js  
page.includeJSlibs.jquery.external = 1  
page.includeJSlibs.jqueryUi = fileadmin/jqueryUI.js  
page.includeJS.beliebigeBezeichnung1 = fileadmin/script1.js  
page.includeJS.beliebigeBezeichnung2 = fileadmin/script2.js
```

*Hinweis:* includeJSlibs bedeutet für TypoScript Library was wiederum seine Auswirkung auf die Reihenfolge hat: Libraries werden immer vor dem normalen JS eingebunden.

*Hinweis:* Externe Dateien müssen zusätzlich mit .external=1 gekennzeichnet werden

```
...  
page.includeJSFooterlibs.jQuery =  
https://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js  
page.includeJSFooterlibs.jQuery.external = 1  
page.includeJSFooterlibs.jqueryUi = fileadmin/jqueryUI.js  
page.includeJSFooter.beliebigeBezeichnung1 = fileadmin/script1.js  
page.includeJSFooter.beliebigeBezeichnung2 = fileadmin/script2.js  
...
```

*Vorteil: Bei der Einbindung von JavaScript-Dateien im Fußbereich einer HTML-Seite, kann der Besucher den Seiteninhalt bereits betrachten, selbst wenn der Browser das JS noch lädt.*

## 2.1.4 Meta-Tags im HTML Header

Beliebige Meta-Tags können über das Content Object „meta“ erzeugt werden. Das Objekt verfügt auch über die stdWrap Eigenschaft (dazu später mehr)

```
meta {  
    keywords = Meine, allgemeinen, Keywords  
    description = Das ist die Beschreibung der Seite  
}
```

## 2.1.3 Manuelle Einbindung von Tags in den HTML-Header

Um CSS- und JS-Dateien einzubinden können Sie sich an die beiden letzten Abschnitte halten. Wenn Sie jedoch aus irgendeinem Grund direkt in den HTML-Header schreiben müssen, hilft Ihnen TypoScript auch weiter

```
...  
page.headerData.1 = TEXT  
page.headerData.1.value = Website  
page.headerData.1.wrap = <title>|</title>  
...
```

## 2.2 Allgemeine Konfiguration

Für die Installation der Website können im Top Level Object „config“ globale Einstellungen gespeichert werden. Sie werden ebenfalls gecached.

```
config {  
    absRefPrefix = / # fügt allen Links diese String an den  
    baseURL = https://meine.domain.de # rendert das <baseUrl>  
    compressCss = 1  
    compressJs = 1  
    concatenateCss = 1  
    concatenateJs = 1  
}
```

Anfang  
Tag

## 2.3 Erweiterte Anforderungen

### 2.3.1 HMENU

Mit Hilfe des TypoScript Objekt vom Typ HMENU (Hierarchisches Menü) ist es möglich Menüs aller Arten von TYPO3 erzeugen zu lassen (Navigation, Breadcrumb, Sitemaps, Flyout-Menüs, Grafische Menüs, etc...). An dieser Stelle zeigen wir ein einfaches Beispiel.

Um in unserem HTML-Beispiel Fortzufahren wollen wir das statische Menü im HTML-Template gerne durch ein dynamisch-erzeugtes Menü ersetzen.

```
...
<div id="header">
    {LOGO}
</div>
<div id="menu">
    <f:cObject typoscriptObjectPath="lib.menu">
        <ul>
            <li><a href="#">Home</a></li>
            <li><a href="#">Seite 2</a></li>
            <li><a href="#">Seite 3</a></li>
        </ul>
    </f:cObject>
</div>

<div id="content">
    <h1>Überschrift</h1>
    Seiteninhalt
</div>
...

```

```
...
lib.menu = HMENU
lib.menu {
    1 = TMENU
    1 {
        wrap = <ul>|</ul>
        NO = 1
        NO.allWrap = <li>|</li>
    }
}
...

```

Alle Details zu sämtlichen Menü-Arten finden Sie in der TypoScript Referenz.

### 2.3.2 TypoLink

TYPO3 bietet seine eigenen Funktionen um Links zu generieren an. Natürlich könnte man Links auch statisch im TypoScript einfügen, dies wäre aber kein guter Weg. Wenn Seiten verschoben oder umbenannt werden, soll sich ein Link natürlich entsprechend anpassen. Ein weiterer Punkt ist der Einsatz von Extensions, die statische Seiten simulieren sollen. Das funktioniert nur, wenn diese Erweiterungen in die Linkgenerierung eingreifen können.

```
lib.object = TEXT
lib.object {
    value = Klick mich
    typolink.parameter = www.google.de
}
```

```
lib.object = TEXT
lib.object {
    value = Klick mich
    typolink.parameter = 1
}
```

```
lib.object = TEXT
lib.object {
    typolink.parameter = 4
    typolink.returnLast = url
}
```

```
lib.object = TEXT
lib.object {
    value = Klick mich
    typolink.parameter = info@in2code.de
}
```

```
lib.object = TEXT
lib.object {
    value = Klick mich
    typolink.parameter = 2
    typolink.additionalParams = &wert=halloWelt
}
```

```
lib.object = IMAGE
lib.object {
    file = fileadmin/logo.png
    imageLinkWrap = 1
    imageLinkWrap.enable = 1
    imageLinkWrap.typolink.parameter = 3
}
```

### 2.3.3 If

Mit der If-Funktion kann man etwas mehr Dynamik ins TypoScript bekommen.

**Beispiel:** Ein Objekt bekommt einen wrap zugewiesen. In unserem Beispiel wollen wir den Untertitel (subtitle) einer Seite direkt an einem Marker ausgeben. Mit einem wrap sorgen wir dafür, dass der Subtitle fett dargestellt wird

```
lib.object = TEXT
lib.object.field = subtitle
lib.object.wrap = <b>|</b>
```

Dummerweise wird dieser wrap aber immer ausgeführt, egal ob ein Untertitel vergeben wurde oder nicht. Mit Hilfe von if.isTrue ist das ganz leicht zu ändern:

```
lib.object = TEXT
lib.object.field = subtitle
lib.object.if.isTrue.field = subtitle
lib.object.wrap = <b>|</b>
```

*Tipp 1: Wenn gleich mehrere Objekte dann nicht ausgegeben werden sollen, falls ein Objekt keinen Inhalt hat, kann man alle Objekte in einem Content Object Array fassen und if.isTrue auf das komplette COA anwenden.*

*Tipp 2: Mit if.IsFalse ist das ganze natürlich auch umgekehrt möglich.*

## 2.3.4 CASE

Auch mit CASE lässt sich so etwas Ähnliches wie eine kleine IF Abfrage erstellen. Die Funktion ist in etwa vergleichbar mit der PHP-Funktion SWITCH.

Je nach Wert eines Objektes ist es möglich eine andere Ausgabe zurückgeben. In unserem Fall wird das Feld subtitle missbraucht, um einen konstanten Wert auf einer Seite auszugeben. Wir wollen beispielsweise, dass wenn subtitle den Wert "1" beinhaltet der Wert "Willkommen" und wenn subtitle den Wert "2" beinhaltet den Wert "Tschüss" ausgegeben wird:

```
lib.object = CASE
lib.object {
    key.field = subtitle

    default = TEXT
    default.value = Willkommen

    2 = TEXT
    2.value = Tschüss

    3 = TEXT
    3.value = Schön war es
}
```

### Tipp 2:

Eine konkrete Anwendung kann sich auf Powermail beziehen, wenn man auf der Antwortseite eine spezifische Anrede benötigt (Sehr geehrte Frau Müller / Sehr geehrter Herr Müller).

**Tipp 1:**  
Ein wrap ist bei einem CASE Objekt nicht direkt möglich. Um diesen dennoch zu machen, kann man zuerst ein COA erstellen und diesen wrappen, während dann der Inhalt des COA (z.B. 10) die eigentlich CASE Anweisung enthält.

## 2.3.5 Conditions

Conditions sind eine andere Art der dynamischen Einflussnahme auf das TypoScript. Mit Conditions kann man bestimmte Bereiche im TypoScript aktivieren oder deaktivieren. Sie sollten jedoch mit Bedacht verwendet werden, da sie sich negativ auf die Performance auswirken.

```
lib.object = TEXT
lib.object.value = Willkommen

[globalVar = GP:L=1]
    lib.object.value = Welcome
[end]
```

*Hinweis: Conditions funktionieren nicht innerhalb geschweifter Klammern!*

```
lib.object = TEXT
[usergroup = *]
    lib.object.value = Sie sind eingeloggt
[else]
    lib.object.value = Bitte loggen Sie sich ein
[end]
```

## Alle verfügbaren Conditions

Name	Erklärung
language	Abfrage auf Browsersprache des Besuchers
IP	Abfrage auf IP-Adresse des Besuchers
hostname	Abfrage auf aktuelle URL
hour	Abfrage auf Stunde
minute	Abfrage auf Minute
dayofweek	Abfrage auf Tag der Woche
dayofmonth	Abfrage auf Tag des Monats
dayofyear	Abfrage auf Tag des Jahres
month	Abfrage auf Monat
year	Abfrage auf Jahr
usergroup	Abfrage auf Benutzergruppe des eingeloggten Users
loginUser	Abfrage ob ein bestimmter User eingeloggt ist
treeLevel	Abfrage auf welcher Ebene der User sich befindet
PIDinRootline	Abfrage ob User in einem bestimmten Zweig
PIDupinRootline	Abfrage ob User in einem bestimmten Zweig
globalVar	Abfrage auf globale Variablen
globalString	Abfrage auf globale Variablen

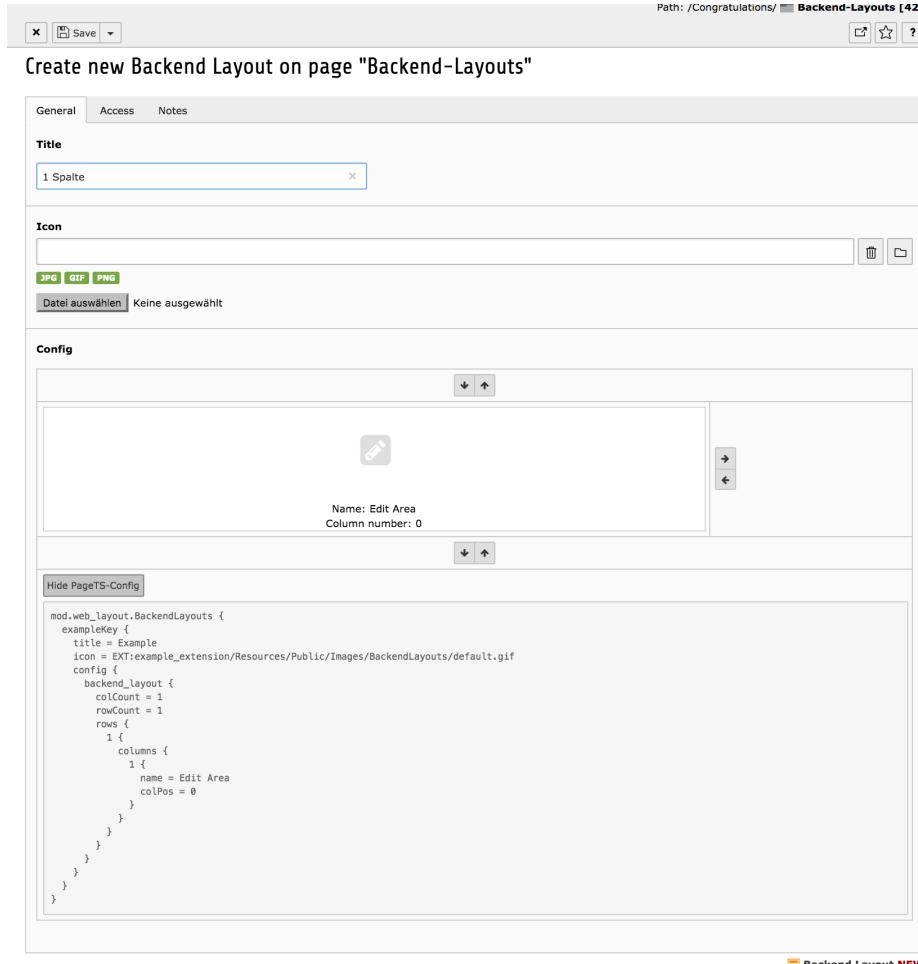
userFunc	Abfrage mit Hilfe einer PHP userFunc
compatVersion	Abfrage auf die TYPO3 – Version
page field	Abfrage auf ein Feld der Seiteneigenschaften
applicationContext	Abfrage auf den TYPO3_CONTEXT

Es ist auch möglich eigene Conditions zu programmieren. Sie müssen dazu

die abstrakte Klasse

`„\TYPO3\CMS\Core\Configuration\TypoScript\ConditionMatching\AbstractCondition“`

erweitern.



The screenshot shows the TYPO3 Backend interface for creating a new Backend Layout. The title bar indicates the path: /Congratulations/ Backend-Layouts [42]. The main area is titled "Create new Backend Layout on page 'Backend-Layouts'". There are three tabs: General (selected), Access, and Notes. The General tab contains fields for Title (set to "1 Spalte"), Icon (with options for JPG, GIF, PNG and a file selection button), and Config (which displays a visual editor with a single column and an "Edit Area"). Below the config editor is a code preview window showing the generated PageTS-Config:

```
Hide PageTS-Config
mod.web_layout.BackendLayouts {
exampleKey {
    title = Example
    icon = EXT:example_extension/Resources/Public/Images/BackendLayouts/default.gif
    config {
        backend_layout {
            colCount = 1
            rowCount = 1
            rows {
                1 {
                    columns {
                        1 {
                            name = Edit Area
                            colPos = 0
                        }
                    }
                }
            }
        }
    }
}
```

At the bottom right of the config editor, there is a "Backend Layout NEW" button.

## 2.4 Verwendung von Backend-Layouts

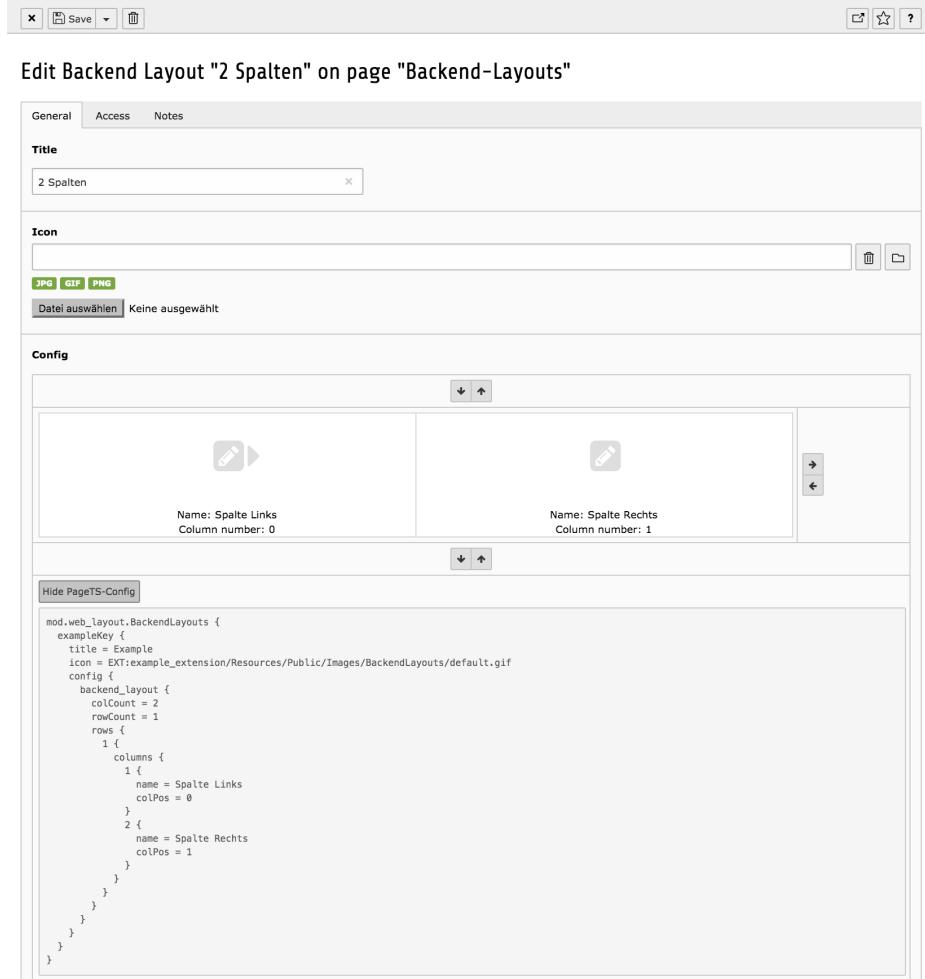
Auch mit einem Content-Management-System soll es unter Umständen möglich sein, verschiedene Seiten, unterschiedlich zu gestalten. Ein häufiger Fall ist die Unterscheidung zwischen einer Spalte oder zwei Spalten auf verschiedenen Seiten.

Schön wäre nun, wenn das Backend sich so verhält wie das Frontend und die Spalten bereits darstellt.

### 2.4.1 Erstellung von Backend-Layouts

Auf einer Seite vom Typ Ordner können über die Listenansicht Backend-Layouts hinzugefügt werden.

Vergeben Sie einen sinnvollen Titel und verwenden Sie den Wizard  um die Einstellung für Ihr Ein- oder Zwei-Spalten-Layout vorzunehmen. Wichtig ist die UID der erstellten Backend-Layouts (sichtbar z.B. bei Mouseover in der Listenansicht) – diese sollten Sie sich merken.



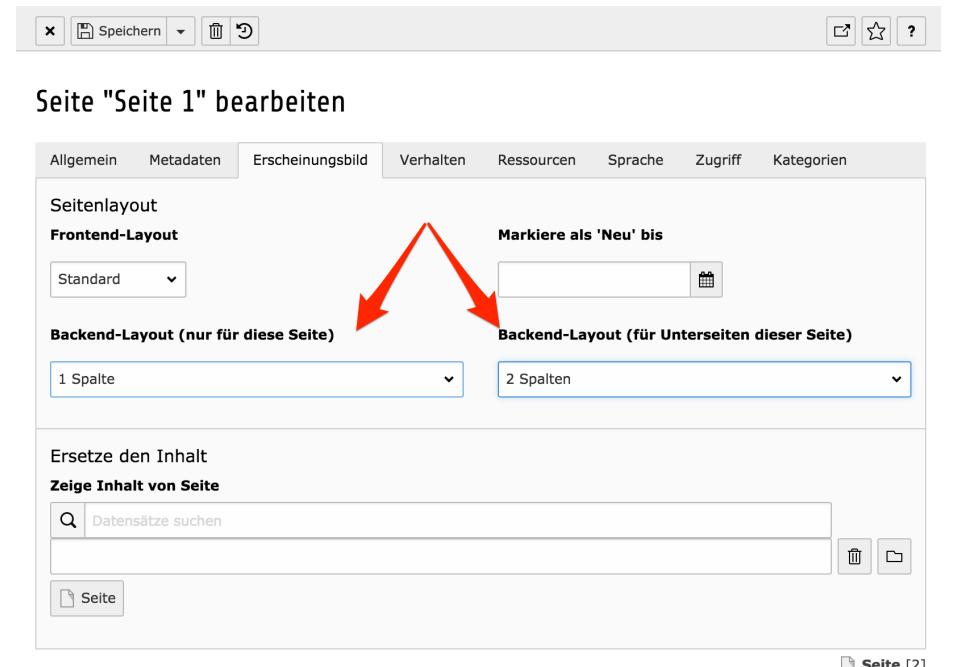
The screenshot shows the 'Edit Backend Layout "2 Spalten" on page "Backend-Layouts"' configuration screen. It includes tabs for General, Access, and Notes. Under General, there is a Title field containing '2 Spalten' and an Icon section with a file browser showing 'Keine ausgewählt'. The Config section displays a grid layout with two columns: 'Name: Spalte Links' (Column number: 0) and 'Name: Spalte Rechts' (Column number: 1). Below this is a code editor window showing the corresponding PageTS-Config:

```

mod.web_layout.BackendLayouts {
    exampleKey {
        title = Example
        icon = EXT:example/Resources/Public/Images/BackendLayouts/default.gif
        config {
            backend_layout {
                colCount = 2
                rowCount = 1
                rows {
                    1 {
                        columns {
                            1 {
                                name = Spalte Links
                                colPos = 0
                            }
                            2 {
                                name = Spalte Rechts
                                colPos = 1
                            }
                        }
                    }
                }
            }
        }
    }
}

```

Nun sollten Sie auf jeder Seite in den Seiteneigenschaften ein entsprechendes Backend-Layout für die aktuelle Seite und für die Unterseiten auswählen können.



The screenshot shows the 'Seite "Seite 1" bearbeiten' (Edit Page "Page 1") screen with the 'Erscheinungsbild' tab selected. In the 'Frontend-Layout' dropdown, 'Standard' is selected. The 'Backend-Layout (nur für diese Seite)' dropdown is set to '1 Spalte' and has a red arrow pointing to it from the text 'Markiere als 'Neu' bis'. The 'Backend-Layout (für Unterseiten dieser Seite)' dropdown is set to '2 Spalten' and also has a red arrow pointing to it from the same text. Below these dropdowns is a search field for content and a 'Seite' button.

Überprüfen Sie nun das Erscheinungsbild im Seitenmodul im Backend – Sie sollten je nach Einstellung eine oder zwei Spalten zu Gesicht bekommen.

## Seite 1

The screenshot shows the TYPO3 backend's page editing interface. It features a main content area divided into two columns: 'Linke Spalte' (Left Column) and 'Rechte Spalte' (Right Column). Each column contains a 'Inhalt' (Content) module. The left column's content module has a rich text editor with placeholder text: 'Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.' The right column's content module includes a 'Bilder' (Images) module with a preview image of a red book and another 'Inhalt' module below it.

#### 2.4.1.1 Unterscheidung der Ausgabe im Frontend

Die Unterscheidung im Frontend wird mit TypoScript getroffen. Die einfachste Möglichkeit wäre je nach Backend-Layout ein anderes HTML-Template zu wählen.

Wir gehen davon aus, dass unser einspaltiges Layout die UID1 und unser zweispaltiges Layout die UID2 beinhaltet. Über ein CASE cObject lässt sich nun einfach ein anderes Template wählen.

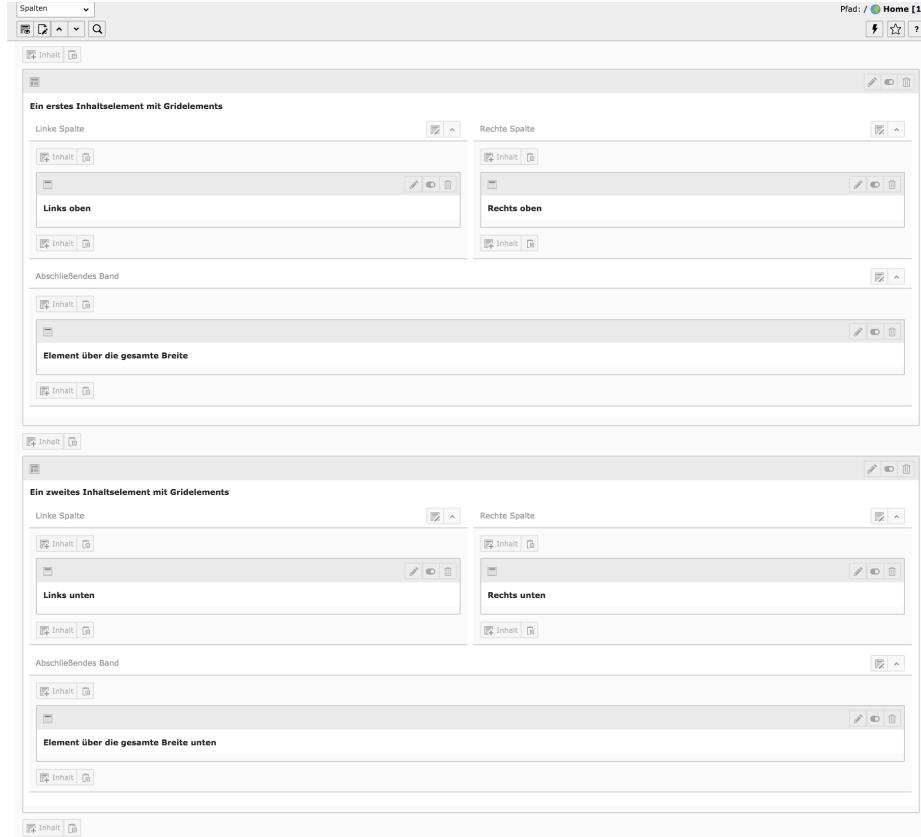
```
page = PAGE
page {
    10 = FLUIDTEMPLATE
    10 {
        file.cObject = CASE
        file.cObject {
            key.field = backend_layout
            key.ifEmpty.data = levelfield:-2, \ backend_layout_next_level, slide
            # 1-col
            default = TEXT
            default.value = fileadmin/1col.html

            # 2-cols
            1 = TEXT
            1.value = fileadmin/2cols.html
        }
    }
}
```

```
lib.content = CASE
lib.content {
    key.field = backend_layout
    key.ifEmpty.data = levelfield:-2, backend_layout_next_level, slide
    default = COA
    default {
        wrap = <div>|</div>
        10 =< styles.content.get
    }
    2 = COA
    2 {
        wrap = <div>|</div>
        10 =< styles.content.get
        10.wrap = <div class="left">|</div>

        20 =< styles.content.get
        20.select.where = colPos = 1
        20.wrap = <div class="right">|</div>
    }
}
```

Alternativ kann man auch das gleiche Template nutzen, aber andere DIV-Container als Wrapper nutzen.



## 2.4.2 Gridelements als Ergänzung zu Backend-Layouts

Bei Gridelements handelt es sich um eine TYPO3-Erweiterung, die zwei Dinge bereitstellt.

Auf der einen Seite ermöglicht die Extension strukturierte Inhaltselemente ähnlich der Backend-Layouts. Die Nutzung von zusätzlichen Containern auf der Ebene der Inhaltselemente wird möglich. Das ist sinnvoll, wenn man nicht immer die gleiche Seitenstruktur benötigt, sondern Redakteure beliebige Spalten und Zeilen mit Inhalten anlegen können.

Grundsätzlich ist auch die Erstellung von Flexible Content Elements (FCE) mit Gridelements möglich, aber inzwischen gibt es mit EXT:mask und EXT:mask\_export eine wesentlich einfachere Möglichkeit, individuelle Content-Elemente zu erstellen.

## 2.4.3 Templates komplett auslagern

Als Alternative zum letzten Punkt kann man TypoScript auch in Dateien auslagern.

### Vorteile:

- Mögliche Versionierung mit GIT
- Zugriff mit Entwicklungsumgebungen und anderen Editoren möglich
- TypoScript kann ohne Backend-Login geändert werden

### Nachteile:

- Eine Gliederung in kleine Snippets ist eher schwierig
- Die TYPO3 Versionierung schlägt fehl – man muss sich um eine eigene Versionierung kümmern
- Eine zeitliche Steuerung über Start- und Stopzeit ist nicht möglich
- Ein temporäres Deaktivieren im Backend ist nicht möglich
- Die TypoScript-Autovervollständigung im Backend kann nicht genutzt werden

```
<INCLUDE_TYPOSCRIPT:source="file:fileadmin/mainTypoScript.typoscript">
```

## 2.4.3.1 Auslagern in eigene Extension

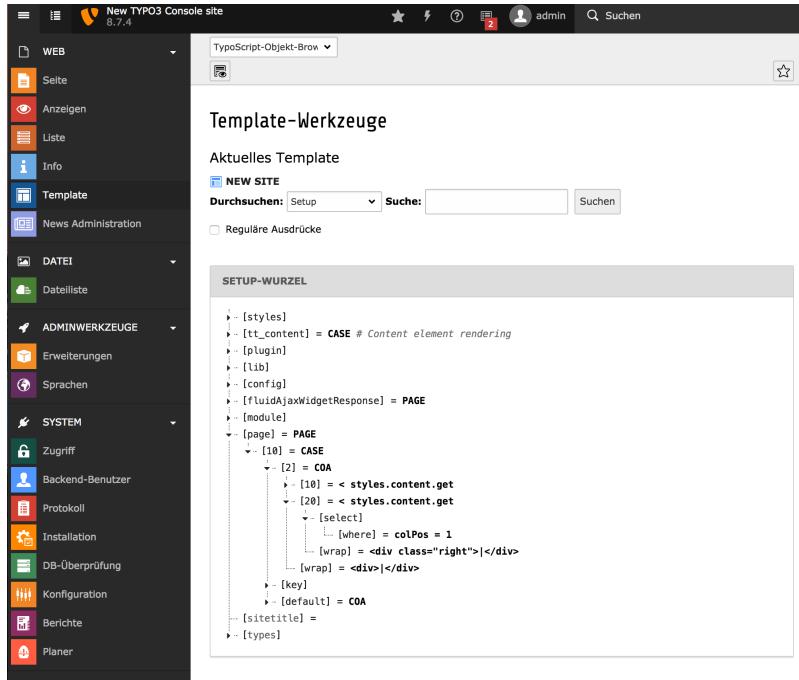
Im Prinzip kann man noch einen Schritt weitergehen und eine TYPO3-Erweiterung erstellen, die sämtliche Layouts, CSS, HTML, TypoScript, JavaScript, Bilder, etc... bei der Installation bereits mitbringt.

Diese Erweiterung lässt sich einfach per Installation in ein leeres TYPO3-System einbringen und würde sofort alle benötigten administrativen Einstellungen mitbringen.

### Vorteile:

- Eine Extension lässt sich durch eine Installation einfach deployen
- Weitere Instanzen mit gleichem Aussehen lassen sich extrem schnell erstellen
- Die Versionsnummer einer Extension lässt sich nutzen, um das Projektmanagement- und Versionierungstool abzulegen
- Mögliche Versionierung mit GIT
- Zugriff mit externen Editoren und Entwicklungsumgebungen
- TypoScript kann ohne Backend-Login geändert werden

*Diese Anmerkungen dienen lediglich als Anregung. Eine exakte Anleitung einer solchen Extension sprengt hier den Rahmen.*



## 2.4.4 Der TypoScript-Object-Browser

Je größer eine Seite wird, desto unübersichtlicher werden die TypoScript Einstellungen. Es kann passieren, dass Sie sich im Backend auf einer Unterseite befinden und sich fragen, warum an dieser Stelle ein komisches Verhalten auftritt, dass Sie auf TypoScript zurückführen.

Hierfür gibt es den TypoScript Object Browser im Backend Modul Template. Dort können Sie die Einstellungen einer Seite betrachten, nach Stichworten suchen oder das Verhalten von Conditions simulieren.

So können Sie eventuellen Fehlern auf den Grund gehen.

# 3 Fluid – Templating

## 3.1 Einführung

Für TYPO3 existieren verschiedene Möglichkeiten die Templates umzusetzen.

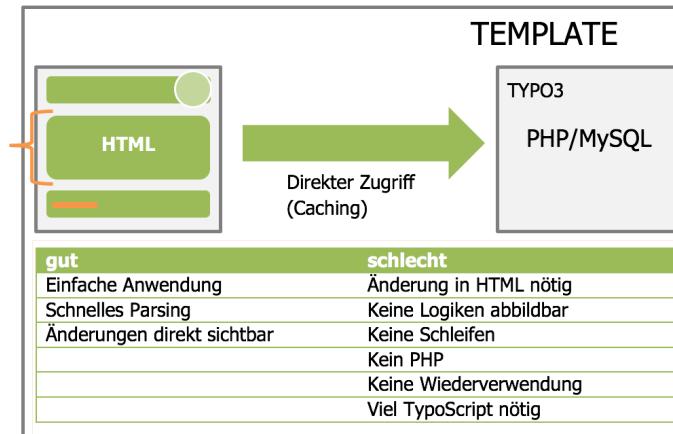
- Marker-based TEMPLATE
- Extension: Automaketemplate
- Extension: TemplaVoila
- Fluid-based Template

Fluid ist eine mächtige Template-Engine, die das veraltete TEMPLATE cObject von TYPO3 vollständig ersetzt und um neue Funktionen und Eigenschaften ergänzt.

Die anderen Möglichkeiten werden Ihnen nur in (sehr) alten Projekten begegnen. Der Einsatz in neuen Projekten ist nicht zu empfehlen.

*Hinweis: Große Teile von Fluid wurden im Laufe der Entwicklung der Version 8 in ein eigenständiges Projekt ausgelagert. Nicht ausgelagerte Funktionen und Viewhelper befinden sich in der System-Extension „fluid“.*

### 3.1.1 Klassisches Templating

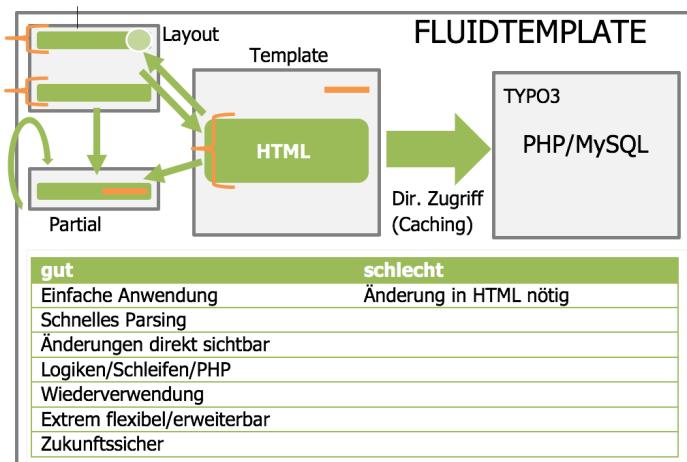


Das klassische Templating ist marker-basiert und ersetzt Marker und Subparts in einer HTML-Datei durch dynamische Inhalte. Dieses Vorgehen ist zwar sehr effizient, aber aus Sicht des Integrators nicht einfach zu handhaben.

Deswegen wurde Fluid als Templating – Engine entwickelt.

### 3.1.2 Vorteile von Fluid gegenüber klassischem Templating

- Teile der Templates können in anderen Templates wiederverwendet werden
- Logik kann im HTML-Template abgebildet werden (If, Schleifen, etc...)
- Nutzen Sie vorhandene ViewHelper (Typolink, Bildfunktionen, etc...)
- Der Einsatz von eigenen ViewHelpers (PHP Funktionen) ist direkt (ohne TypoScript) möglich.



### 3.1.3 Grundsätzliches Vorgehen

Die erfolgreiche Verwendung erfolgt durch folgende zwei Schritte:

### 3.1.3.1 Erstellung einer HTML Datei

In unserem Beispiel nehmen wir ein vereinfachtes HTML-Gerüst, dass nur aus einer Hand voll DIV-Containern besteht.

```
<html>
    <head>
        <title>Neue Website</title>
    </head>
    <body>
        <div id="header">
            
        </div>
        <div id="menu">
            <ul>
                <li><a href="#">Home</a></li>
                <li><a href="#">Seite 2</a></li>
                <li><a href="#">Seite 3</a></li>
            </ul>
        </div>
        <div id="content">
            <h1>Überschrift</h1>
            Seiteninhalt
        </div>
    </body>
</html>
```

### 3.1.3.2 Aufruf im TypoScript

Im TypoScript lässt sich das HTML Template direkt einbinden.

```
page = PAGE
page.10 = FLUIDTEMPLATE
page.10 {
    file = fileadmin/template.html
}
```

Bei FLUIDTEMPLATE handelt es sich um ein TypoScript Content Object, das dafür gedacht ist, Dateien oder andere Quellen als Template mit der Templateengine FLUID einzubinden.

**Das Problem:** Bei dieser Einbindung werden Sie feststellen, dass es im Quelltext im Frontend zwei Mal einen Head-Bereich gibt. Der erste Bereich kommt aus dem HTML-Template, der zweite Bereich wird von TypoScript durch das PAGE Objekt generiert. Zwar lässt sich die Generierung eines Head Bereiches durch TYPO3 unterbinden, wäre aber der falsche Weg - denn nur mit dem generierten Header-Bereich kann TYPO3 verschiedene CSS-, JavaScript- und Meta-Informationen dynamisch einbinden.

**Die Lösung:** Durch den Einsatz einer „Section“ und eines „Layouts“, lässt sich der BODY-Bereich markieren.

```
page = PAGE
page.10 = FLUIDTEMPLATE
page.10 {
    file = fileadmin/template.html
    layoutRootPaths.10 = fileadmin/layouts/
}
```

```
<html>
    <head>
        <title>Neue Website</title>
    </head>
    <body>
        <f:layout name="layout1" />
        <f:section name="body">
            [Alle DIV-Container]
        </f:section>
    </body>
</html>
```

```
<f:render section="body" />
```

*Hinweis: Im HTML-Template wird eine Section mit dem Namen „body“ definiert. Über dem Section-Tag wird der Layout-Name „layout1“ gesetzt. Dadurch, dass im TypoScript der Pfad zu den Layouts definiert wurde, weiß Fluid, dass es die Datei fileadmin/layouts/layout1.html öffnen soll.*

*Im Layout kann das HTML-Template zusätzlich mit einem Header oder Footer umschlossen werden*

### 3.2 Unterteilung in Templates/Layouts/Partials

Fluid verfügt über drei Typen von Template-Dateien: Templates, Partials und Layouts

<b>Templates</b>	<p>Template ist das Haupt-HTML-Template. Dieses wird immer zuerst aufgerufen und beinhaltet in der Regel den Hauptteil der Ausgabe</p> <p>Das Template sollte immer die Angabe eines Templates enthalten.</p>
<b>Partials</b>	<p>Ein Partial ist eine Art Snippet. Hierbei handelt es sich also um einen ausgelagerten Teilbereich. Der Vorteil des Auslagerns ist die Wiederverwendbarkeit.</p> <p>Wenn man beispielsweise eine Suchmaske in ein Partial auslagert, lässt sich diese in verschiedenen Templates wiederverwenden. Partials können von Layouts, Templates und anderen Partials eingebunden werden.</p>

#### Layouts

Bei einem Layout handelt es sich um das Gerüst des Templates. Oft beinhaltet das Layout meist nur einen umschließenden DIV-Container mit einer CSS-Klasse.

Die Notwendigkeit des Anlegens weiterer Layouts ergeben sich, wenn man Views benötigt, die eventuell keinen oder andere DIV-Container benötigen (AJAX-Requests, XML, RSS, etc...)

Die Pfade zu den Templates, Partials und Layouts wird im TypoScript festgelegt:

```
page = PAGE
page.10 = FLUIDTEMPLATE
page.10 {
    file = fileadmin/template.html
    layoutRootPaths {
        10 = fileadmin/layouts/
        20 = EXT:my_ext/Resources/Private/Layouts/
    }
    partialRootPaths {
        10 = fileadmin/partials/
        20 = EXT:my_ext/Resources/Private/Partials/
    }
    templateRootPaths {
        10 = fileadmin/templates/
        20 = EXT:my_ext/Resources/Private/Templates/
    }
}
```

Es ist möglich mehrere Pfade für die Fluid-Templates anzugeben. TYPO3 beginnt in dem Verzeichnis mit der höchsten Nummer zu suchen und geht alle Verzeichnisse in numerisch absteigender Reihenfolge durch, bis es die gesuchte Datei gefunden hat. Falls sie nicht existiert, bekommt man „Ooops, an error occurred“ zu sehen.

### 3.3 Variablen

Variablen sind das Pendant zu den Markern im herkömmlichen Templating. Sie werden innerhalb des TypoScript Templates definiert. Sie sind beliebige TYPO3 Content Objekte.

Innerhalb des Fluid-Templates werden sie mit geschweiften Klammern verwendet.

```
page = PAGE
page.10 = FLUIDTEMPLATE
page.10 {
    file = fileadmin/template.html
    layoutRootPath = fileadmin/layouts/
    partialRootPath = fileadmin/partials/
    variables.LOGO = IMAGE
    variables.LOGO.file = fileadmin/logo.png
}
```

```
<f:layout name="layout1" />
<f:section name="body">
    <div id="header">
        {LOGO}
    </div>
    <div id="menu">
        <ul>
            <li><a href="#">Home</a></li>
            <li><a href="#">Seite 2</a></li>
            <li><a href="#">Seite 3</a></li>
        </ul>
    </div>
    <div id="content">
        <h1>Überschrift</h1>
        Seiteninhalt
    </div>
</f:section>
```

```
<f:layout name="layout1" />
<f:section name="body">
    <div id="header">
        <f:format.html parseFuncTSPPath="lib.parseFunc">
            {LOGO}
        </f:format.html>
    </div>
    <div id="menu">
        <ul>
            <li><a href="#">Home</a></li>
            <li><a href="#">Seite 2</a></li>
            <li><a href="#">Seite 3</a></li>
        </ul>
    </div>
    <div id="content">
        <h1>Überschrift</h1>
        Seiteninhalt
    </div>
</f:section>
```

**Das Problem:** Soll HTML-Code an einer Variable ausgegeben werden, so werden besondere Zeichen zuvor aus Sicherheitsgründen in ASCII-Code (Verwendung der Funktion htmlspecialchars) umgewandelt.

Hiermit soll verhindert werden, dass Administratoren und Integratoren unwissentlich eine XSS-Lücke öffnen.

**Die Lösung:** Die Verwendung des HTML-ViewHelpers unterbindet die Umwandlung der Zeichen in deren ASCII-Code und eignet sich gut um HTML als solches anzuzeigen.

*Hinweis: Nicht angesprochene Variablen werden im Frontend automatisch entfernt.*

## 3.4 ViewHelper

Das Grundkonzept der ViewHelper besteht darin, die Ausgabe um weitere Funktionen und Logiken anzureichern. Im Idealfall sieht ein ViewHelper auch aus wie ein HTML-Tag. Dadurch wird die IDE eine korrekte Einrückung vornehmen. Bei direktem Aufruf einer FLUID-Datei mit dem Browser, werden die ViewHelper ignoriert.

FLUID ist inzwischen ein eigenständiges Projekt und kann auch mit anderen Frameworks, wie Zend oder Laraval genutzt werden. Das offizielle Repository für FLUID ist auf Github: <https://github.com/TYPO3/Fluid>

Neben den Viewhelpers, die darüber zur Verfügung stehen, bringt TYPO3 auch eigene Viewhelper mit, die ausschließlich im Kontext von TYPO3 sinnvoll sind.

### 1.1.1. Inline- und Outline-Schreibweise

Alle ViewHelper sind in der Outline- und Inline-Schreibweise verfügbar. Damit wird es möglich, ViewHelper zu verschachteln.

```
<f:translate key="idFromXml" />
```

#### Outline - Schreibweise

```
{f:translate(key:'idFromXml')}
```

#### Inline-Schreibweise

### 3.4.1.1 ViewHelper mit beginnendem und schließendem Tag

```
<f:format.date format="d.m.Y">{date}</f:format.date>
```

```
{date -> f:format.date(format:'d.m.Y')}
```

### 3.4.1.2 Beispiel für verschachtelte View-Helper

```
<f:image src="fileadmin/bild.jpg" alt="{f:translate(key:'bildText')}" />
```

**Verschachtelte ViewHelper mit gemischter Outline- und Inline-Schreibweise**

*Hinweis: Die Website <http://www.fluid-converter.com/> bietet einen Outline-to-Inline-Converter an und kann ganz nützlich sein*

### 3.4.2 Auswahl häufig genutzter ViewHelper

In TYPO3 werden ViewHelper vom TYPO3 Fluid und von TYPO3 CMS zur Verfügung gestellt. Die Herkunft der Viewhelper sind in der jeweiligen Überschrift mit erfasst, damit die Suche nach Dokumentation und Bugreports einfacher ist.

### 3.4.2.1 Debug – ViewHelper (TYPO3 Fluid)

Mit dem Debug-Viewhelper kann man sich die Inhalte von Objekten, die im Fluid-Template zur Verfügung stehen, ausgeben lassen.

```
<f:debug>
    {myObject}
</f:debug>
```

### 3.4.2.2 Comment – ViewHelper (TYPO3 Fluid)

Mit diesem ViewHelper kann man seinem Template Kommentare hinzufügen.  
Alles was zwischen den Tags steht, wird nicht ausgegeben.

```
<f:comment>
    Mein Kommentar, der später nicht im HTML Quelltext zu
    sehen ist.
</f:comment>
```

Der Section-ViewHelper kann in Templates, Partials und >Layouts verwendet werden.

```
<f:section name="someSection">
    Das ist eine Section mit dem Objekt {foo}
</f:section>
```

### 3.4.2.3 Viewhelper für die Struktur (TYPO3 Fluid)

Diese drei ViewHelper helfen die Fluid Templates zu strukturieren.

#### Section – ViewHelper (TYPO3 Fluid)

Dieser ViewHelper umschließt einen Bereich innerhalb eines Fluid-Templates. Durch einen frei zu gebenden Namen ist dieser Teil identifizierbar und kann an beliebiger Stelle über den Render-Viewhelper ausgegeben werden. Der Einsatz dieses ViewHelpers ist dann sinnvoll, wenn man diesen Bereich zwar separat haben möchte, aber eine Auslagerung in ein Partial nicht sinnvoll ist.

## Render – ViewHelper (TYPO3 Fluid)

Über den Render-Viewhelper werden Partials und Sections innerhalb eines Templates ausgegeben. Über den Namen der Section oder des Partials findet der Viewhelper den Code, den er ausgeben soll.

Der Name der Section muss in dem Template definiert sein, in dem sie gerendert werden soll.

Der Partial muss als HTML-Datei vorhanden sein. TYPO3 durchsucht alle partialRootPaths nach dieser Datei.

Der Render-ViewHelper kann in Templates, Partials und Layouts verwendet werden.

```
<f:render partial="somePartial" arguments="{'_all'}" />
```

```
<f:render section="someSection" arguments="{'foo': someContent}" />
```

## Layout – ViewHelper (TYPO3 Fluid)

Der Layout-ViewHelper verweist auf ein Fluid-Layout. Das Layout muss als HTML – Datei in einem der definierten layoutRootPaths existieren. Er darf nur in dem Template verwendet werden.

```
<f:layout name="main">
```

### 3.4.2.4 Link – ViewHelper

Zum Erzeugen von Links stehen fünf Viewhelper zur Verfügung. Wenn kein vollständiger Link mit a-Tag benötigt wird, kann man in diesen View das „link“ durch „uri“ ersetzen. Man erhält dann nur die URL zurück.

#### Page – ViewHelper

Der Page – ViewHelper erzeugt einen Link auf eine Seite im Seitenbaum. Für die Linkgenerierung können zusätzliche Parameter übergeben werden.

```
<f:link.page pageUid="1" additionalParams="{foo: 'bar'}">  
    page link  
</f:link.page>  
  
<f:uri.page pageUid="1" additionalParams="{foo: 'bar'}">  
    page link  
</f:uri.page>
```

#### Typolink \_ViewHelper

Der Typolink – ViewHelper erstellt Links aus Feldern, die den Link-Wizard im Backend unterstützen.

```
<f:link.typolink parameter="{link}">  
    Linktext  
</f:link.typolink>  
  
<f:uri.typolink parameter="{link}">  
    Linktext  
</f:uri.typolink>
```

#### E-Mail-ViewHelper

Dieser ViewHelper erstellt einen Link zu einer E-Mail-Adresse.

```
<f:link.email email="foo@bar.tld" />  
<f:uri.email email="foo@bar.tld" />
```

## External-ViewHelper

Wenn man auf externe Websites verlinken möchte, muss man den ViewHelper "link.external" verwenden.

```
<f:link.external uri="http://www.typo3.org" target="_blank">  
    external link  
</f:link.external>  
  
<f:uri.external uri="http://www.typo3.org" target="_blank">  
    external link  
</f:uri.external>
```

## Action- ViewHelper

Der Action-Viewhelper erzeugt Links zu Extbase-Actions.

```
<f:link.action action="show" controller="standard" pageUid="25">  
    action link  
</f:link.action>  
  
<f:uri.action action="show" controller="standard" pageUid="25"> action link  
</f:uri.action>
```

```
<a  
href="index.php?id=25&tx_myextension_plugin[action]=show&tx_myextension_plugin[controller]=Standard&cHash=xyz">action link</a>
```

## 3.4.2.5 Image – ViewHelper

Mit dem Image-ViewHelper können Bilder ausgegeben und formatiert werden. Die einfachste Anwendung sieht wie folgt aus:

```
<f:image src="EXT:myext/Resources/Public/typo3_logo.png" alt="alt text" />
```

### Einfaches Beispiel für den ViewHelper "image"

```
<f:image image="{image}" width="1200c" height="400c" alt="{image.alternative}"  
title="{image.title}" treatIdAsReference="true" />
```

### Image-Viewhelper, wenn Datei über FAL ausgeliefert wird

### 3.4.2.6 Format – ViewHelper

Zur Formatierung der Ausgabe gibt es eine Reihe von Viewhelper.

#### ViewHelper "format.html" (TYPO3 Fluid)

Dieser ViewHelper parsed den Input durch die TypoScript - Einstellungen in "lib.parseFunc". Optional ist es möglich eine eigene Konfiguration zu verwenden.

```
<f:format.html>
    foo <b>bar</b>.
    Some <t3://page=1>Link</t3>
</f:format.html>
```

```
foo <b>bar</b>. Some <a href="index.php?id=1" >link</a>.
```

#### Viewhelper „format.raw“ (TYPO3 Fluid)

Dieser ViewHelper gibt die Inhalt so aus, wie er ist. Dies kann Auswirkungen auf die Sicherheit haben und Cross-Site-Scripting ermöglichen.

```
<f:format.raw>{string}</f:format.raw>
```

(Content of {string} without any conversion/escaping)

#### Viewhelper „format.htmlspecialchars“ (TYPO3 Fluid)

Dieser ViewHelper schickt den Inhalt durch die php-Funktion htmlspecialchars und gibt den Inhalt entsprechend formatiert aus.

```
<f:format.htmlspecialchars>{text}</f:format.htmlspecialchars>
{text -> f:format.htmlspecialchars(encoding: 'ISO-8859-1')}
```

## ViewHelper "format.printf" (TYPO3 Fluid)

Dieser Viewhelper verwendet die php-Funktion "printf" zur Ausgabe und formatiert den Content entsprechend.

```
<f:format.printf arguments="{0: 3, 1: 'Kasper'}">
    %2$s is great, TYPO%1$d too. Yes, TYPO%1$d is great and so is %2$s!
</f:format.printf>
```

Kasper is great, TYPO3 too. Yes, TYPO3 is great and so is Kasper!

## ViewHelper "format.crop" (TYPO3 CMS)

Dieser ViewHelper schneidet den Inhalt nach einer festgelegten Zeichenzahl ab und hängt optional eine weitere Zeichenfolge an.

```
<f:format.crop maxCharacters="17" append="&nbsp;[more]">
    This is some very long text
</f:format.crop>
```

This is some&nbsp;[more]

## 3.4.2.7 Translate – ViewHelper (TYPO3 CMS)

Der Translate – ViewHelper bekommt als Argument eine Referenz auf ein Label, das in einer XLIFF-Datei gespeichert ist. Er gibt dann den Wert in der aktuellen Sprache aus.

```
<f:translate key="LLL:EXT:myext/Resources/Private/Language/locallang.xlf:key1" />
```

value of key "key1" in the current website language

## 3.4.2.8 ViewHelper „cObject“ (TYPO3 CMS)

Der cObject – Viewhelper bekommt als Argument einen Verweis auf eine TypoScript-Snippet übergeben. Das Ergebnis wird dann an dieser Stelle ausgegeben.

```
<f:cObject typscriptObjectPath="lib.someLibObject" />
```

rendered lib.someLibObject

### 3.4.2.9 Spaceless – ViewHelper (TYPO3 Fluid)

Nach dem Rendern eines ViewHelpers löscht Fluid die ViewHelper-Kommandos. Die Zeilen bleiben aber in der Ausgabe bestehen, was zu einem hässlichem HTML Output führt. Dieser ViewHelper löscht alle überflüssigen Zeilenumbrüche zwischen den HTML-Tags aus dem Output.

```
<f:spaceless>
  <div>
    <div>
      <div>text
        text</div>
      </div>
    </div>
  </f:spaceless>
```

Beispiel Spaceless-Viewhelper

```
<div><div><div>text
text</div></div></div>
```

"Gesäuberter" Output des Spaceless – Viewhelper

### 3.4.2.10 Viewhelper für If-Statements (TYPO3 Fluid)

Mit dem if-ViewHelper können Bedingungen geprüft werden. Templatecode innerhalb des ViewHelpers wird nur dann ausgeführt, wenn die Bedingung erfolgreich ist.

```
<f:if condition="{rank} > 100">
  Will be shown if rank is > 100
</f:if>
```

Vergleiche für Strings sind nur über den Umweg von Arrays möglich.

```
<f:if condition="{0: foo.bar} == {0: 'stringToCompare'}">
  Will result true if {foo.bar}'s represented value equals 'stringToCompare'.
</f:if>
```

### 3.4.2.11 Then – und Else ViewHelper (TYPO3 Fluid)

Mit einem if – Viewhelper können auch „wenn“ – „dann“ – „sonst“ – Ausgaben realisiert werden.

```
<f:if condition="somecondition">
    <f:then>
        This is being shown in case the condition matches.
    </f:then>
    <f:else>
        This is being displayed in case the condition evaluates to FALSE.
    </f:else>
</f:if>
```

### 3.4.2.12 Switch – ViewHelper (TYPO3 Fluid)

Mit dem Switch – Viewhelper kann man die Ausgabe auf Grund mehrerer Werte ändern. Wenn kein Wert zutrifft, kann eine Standardausgabe erfolgen

```
<f:switch expression="{person.gender}">
    <f:case value="male">Mr.</f:case>
    <f:case value="female">Mrs.</f:case>
    <f:defaultCase>Mr. / Mrs.</f:defaultCase>
</f:switch>
```

### 3.4.2.13 Form – ViewHelper (TYPO3 CMS)

Dieser ViewHelper stellt das HTML – Tag „form“ zur Verfügung. Für jedes Element eines Formulars gibt es einen eigenen Viewhelper, wie z.B. „format.input“, „format.textarea“, „format.button“ etc.

```
<f:form action="..." name="customer" object="{customer}">
    <f:form.hidden property="id" />
    <f:form.textfield property="name" />
    <f:form.select name="name_for_field">
        <f:form.select.option value="1">Mein Label</f:form.select.option>
        <f:form.select.optgroup>
            <f:form.select.option value="2">Mein
Label 2 </f:form.select.option>
            <f:form.select.option value="3">Mein
Label 3 </f:form.select.option>
        </f:form.select.optgroup>
        <f:form.submit value="Send Mail" />
    </f:form>
```

Folgende ViewHelper stehen für Formular-Elemente zur Verfügung:

- f:form.button
- f:form.submit
- f:form.checkbox
- f:form.hidden
- f:form.textarea
- f:form.textfield
- f:form.password
- f:form.radio
- f:form.upload
- f:form.select
- f:form.select.option
- f:form.select.optgroup

### 3.4.2.14 Variable – ViewHelper (TYPO3 Fluid)

Dieser ViewHelper weist einer Template-Variable einen Wert zu, der auch nach dem Rendern des ViewHelpers zur Verfügung steht.

```
{f:variable(name: 'myvariable', value: 'some value')}  
<f:variable name="myvariable">some value</f:variable>
```

### 3.4.2.15 Alias – ViewHelper (TYPO3 Fluid)

Dieser ViewHelper erzeugt ein Alias von bestehenden Variablen. Nach dem Schließen des ViewHelpers stehen die „neuen“ Variablen nicht mehr zur Verfügung.

```
<f:alias map="{x: 'foo'}">{x}</f:alias>
```

*Hinweis: Im Verzeichnis „/typo3/sysext/fluid\_styled\_content/Resources/Private“ befinden sich die Templates und Partials für die Standard-Content-Elemente von TYPO3. Dort findet man viele Beispiele für die Anwendung von Viewhelpers.*

## 4 Fluid-Styled-Content

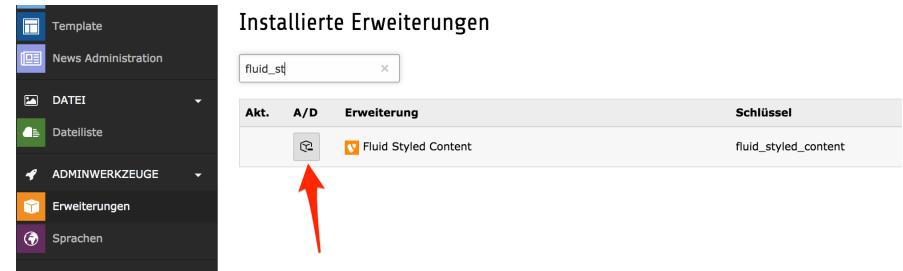
Wenn man im Backend nun Content-Elemente angelegt und sich diese im Frontend ansehen will, bekommt man solche Fehlermeldungen:

```
"ERROR: Content Element with uid "7" and type "text" has no rendering definition!"  
  
"ERROR: Content Element with uid "11" and type "image" has no rendering definition!"
```

Sie besagen, dass TYPO3 zwar Inhaltselemente gefunden hat, aber nicht weiß, wie sie im Frontend auszugeben sind.

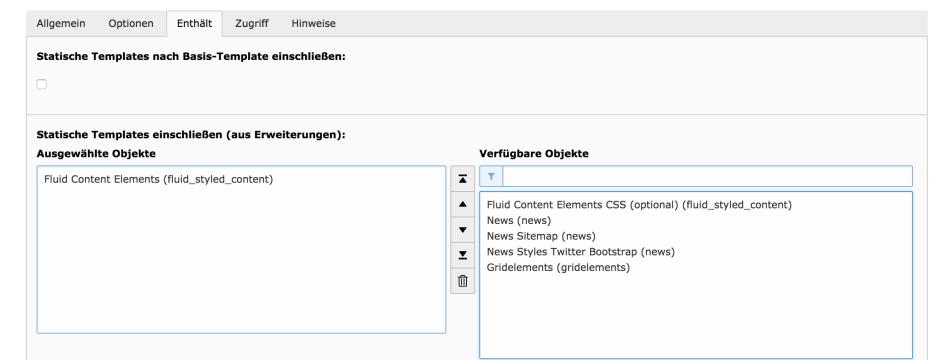
### 4.1 Installation der Extension „fluid\_styled\_content“

Abhilfe schafft hier die Installation der System-Extension „fluid\_styled\_template“ im Extension-Manager.



In einem zweiten Schritt muss das statische TypoScript eingebunden werden. Das passiert im Modul „Template“ in der Ansicht „Vollständigen Template-Datensatz bearbeiten“.

### Template "NEW SITE" auf Seite "Home" bearbeiten



Durch einen Klick im Bereich „Verfügbare Objekte“ auf „Fluid Styled Content (fluid\_styled\_content)“ wird das Template hinzugefügt. Nach dem Speichern des Templates werden nun die Inhalte im Frontend ausgegeben.

#### Mein erstes Element

Lore ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

#### Meine Bilder



Dieser Inhalt ist noch komplett ungestyled, aber bringt das vollständiges HTML-Markup mit CSS Klassen mit. In der Regel wird das aber nicht genügen, man wird also das Markup und CSS-Definitionen anpassen wollen.

## 4.2 Content-Elemente anpassen

Wie man CSS Dateien einbindet, ist im Abschnitt über TypoScript erklärt. Das Markup wird von Templates und Partials geliefert. Für jedes mögliche Inhaltselement bringt TYPO3 bereits eine vollständige Definition mit.

```
"/typo3/sysext/fluid_styled_content/Resources/Private"
```

Je nach TYPO3-Version können das Verzeichnis-Layout und die vorhandenen Dateien abweichen, da in TYPO3 8 LTS Fluid Styled Content sehr stark umgebaut werden.

## 4.2.1 Verzeichnisstruktur

### 4.2.1.1 Das Verzeichnis „Layout“

Im Verzeichnis „Layout“ befindet sich nur noch die Datei „Default.html“, die für alle Content-Elemente verwendet wird. Das Layout verfügt über fünf Abschnitte (Sections):

- Before
- Header
- Main
- Footer
- After

Die Section „Header“ renderet alles, was mit Überschriften zu tun hat. Die Section „Footer“ enthält standardmäßig nur den „To-Top-Link“.

Die Section „Main“ ist in dem Template für das jeweilige Content-Element definiert.

Die Sections „Before“ und „After“ sind in TYPO3 8LTS neu hinzugekommen. Alles, was darüber hinzugefügt wird, wird am Anfang bzw. am Ende des umschließenden Containers hinzugefügt.

### 4.2.1.2 Das Verzeichnis „Templates“

Das Verzeichnis „Templates“ ist der Einstiegspunkt für jedes Content-Element. Für jedes Content-Element gibt es ein eigenes Template.

### 4.2.1.3 Das Verzeichnis „Partials“+

Hier sind die gesamten Partials hinterlegt, die in den Templates und im Layout Verwendung finden. Die Pfadangaben entsprechen den Aufrufen ausgehend von dem Eltern-Verzeichnis

## 4.2.2 Layouts, Templates und Partials überschreiben

Hier greift der bereits bekannte Mechanismus zum Überschreiben der Pfade. In dem TypoScript Template müssen dafür folgende Zeilen hinzugefügt werden:

```
lib.fluidContent {  
    templateRootPaths {  
        100 = fileadmin/templates/  
        200 = EXT:yourextensionkey/Resources/Private/Templates/  
    }  
    partialRootPaths {  
        100 = fileadmin/partials/  
        200 = EXT:yourextensionkey/Resources/Private/Partials/  
    }  
    layoutRootPaths {  
        100 = fileadmin/layouts/  
        200 = EXT:yourextensionkey/Resources/Private/Layouts/  
    }  
}
```

*Hinweis:* Die Dateien dürfen nicht innerhalb im TYPO3-Code geändert werden. Die eigenen Änderungen gehen dann beim nächsten TYPO3 Update wieder verloren.

Mit dieser Konfiguration sucht TYPO3 zuerst in der Extension "yourextensionkey" und danach im „fileadmin“. Wenn dort kein entsprechendes Template oder Partial gefunden wurde, dann wird auf die von TYPO3 mitgelieferten Dateien zurückgegriffen.

*Hinweis:*  
Wenn man nur eine Kleinigkeit an dem Template oder Partial ändern möchte, dann empfiehlt es sich, sich eine Kopie des Originals in die RootPaths zu legen und dort dann die Änderungen vornehmen.

#### 4.2.3 Aufbau eines Templates oder Partials

Grundsätzlich ist ein Template oder Partial, wie eine normale HTML-Datei anzusehen und zu verwenden. Deswegen enden die Dateinamen auch alle auf .html.

In der Datei selbst befindet sich nur der Teil, der sich innerhalb des Body-Tags befinden würde.

Wenn Viewhelper verwendet werden sollen, müssen am Anfang der Datei die verwendeten Namespaces angeben werden. Die Namespaces sind notwendig, da nicht nur der TYPO3-Core, sondern jede Extension eigene Viewhelper mitbringen kann. Sie verhindern den Konflikt bei gleich benannten Viewhelpern.

Die Namespaces müssen am Anfang jedes Templates oder Partials in einem HTML – Tag definiert werden. Der Standard-Namespace für TYPO3 Fluid ist das „f.“.

```
<html
    xmlns:f="http://typo3.org/ns/TYPO3/CMS/Fluid/ViewHelpers"
    xmlns:n="http://typo3.org/ns/GeorgRinger/News/ViewHelpers"
    data-namespace-typo3-fluid="true">
```

Nun als komplettes Beispiel an Hand des Content-Elements „Text und Media“:

```
<html xmlns:f="http://typo3.org/ns/TYPO3/CMS/Fluid/ViewHelpers" data-namespace-typo3-
fluid="true">
<f:layout name="Default" />

<f:section name="Header">

    <f:if condition="{gallery.position.noWrap} != 1">
        <f:render partial="Header/All" arguments="{'_all'}" />
    </f:if>

</f:section>
<f:section name="Main">
    <div class="ce-textpic ce-{gallery.position.horizontal} ce-
{gallery.position.vertical}{f:if(condition: gallery.position.noWrap, then: ' ce-
nowrap')}">
        <f:if condition="{gallery.position.vertical} != 'below'">
            <f:render partial="Media/Gallery" arguments="{'_all'}" />
        </f:if>
        <f:if condition="{data.bodytext}">
            <f:then>
                <div class="ce-bodytext">
                    <f:if condition="{gallery.position.noWrap}">
                        <f:render partial="Header/All" arguments="{'_all'}" />
                    </f:if>
                    <f:format.html>{data.bodytext}</f:format.html>
                </div>
            </f:then>
            <f:else>
                <f:if condition="{gallery.position.noWrap}">
                    <f:if condition="{data.header}">
                        <div class="ce-bodytext">
                            <f:render partial="Header/All"
arguments="{'_all'}" />
                        </div>
                    </f:if>
                </f:else>
            </f:if>
        </f:if>
        <f:if condition="{gallery.position.vertical} == 'below'">
            <f:render partial="Media/Gallery" arguments="{'_all'}" />
        </f:if>
    </div>
</f:section>
</html>
```

## 5 Links und Hilfe

### 5.1 Links

Titel	Links
TypeScript Reference	<a href="https://docs.typo3.org/typo3cms/TyposcriptReference/Index.html">https://docs.typo3.org/typo3cms/TyposcriptReference/Index.html</a>
Fluid Guide	<a href="https://docs.typo3.org/typo3cms/ExtbaseGuide/Fluid/Index.html">https://docs.typo3.org/typo3cms/ExtbaseGuide/Fluid/Index.html</a>
Fluid Styled Content	<a href="https://docs.typo3.org/typo3cms/extensions/fluid_styled_content/8.7/">https://docs.typo3.org/typo3cms/extensions/fluid_styled_content/8.7/</a>
TYPO3 Slack	<a href="https://typo3.slack.com">https://typo3.slack.com</a>

## 5.2 Kontakt

in2code GmbH  
Kunstmühlstraße 12a  
83026 Rosenheim  
  
+49 (0) 80 31 / 88 73 983  
[service@in2code.de](mailto:service@in2code.de)