

in2code GmbH
Kunstmühlstraße 12a
83026 Rosenheim

t +49 8031 8873983
f +49 8031 8873985

info@in2code.de
www.in2code.de

Entwicklung einer TYPO3- Erweiterung mit Hilfe der Extensions Extbase und Fluid

1. Inhalt

2.	Einführung	7
2.1.	Grundlegendes.....	7
2.2.	Begriffserklärung.....	8
2.2.1.	Extbase	8
2.2.2.	Fluid	9
2.2.3.	Objektorientierte Programmierung.....	10
2.2.3.1.	Prozedurale Programmierung (früher).....	10
2.2.3.2.	Objektorientierte Programmierung (heute).....	10
2.2.3.3.	Das Auto-Beispiel.....	11
	Aus der realen Welt.....	11
	Kurzerklärung	12
	Zugriffsregelungen	13
2.2.4.	Das MVC-Konzept	14
2.2.4.1.	So nicht.....	15
2.2.4.2.	Die Trennung	16
2.2.5.	Domain Driven Design	17

2.2.5.1.	Das Problem in der Softwareentwicklung	17
2.2.5.2.	Begriffe	18
	Domain	18
	Modell	18
2.2.5.3.	Lösungsansatz	19
2.2.6.	Convention over Configuration	20
2.2.7.	Namespaces	20
2.2.8.	Vendor Name	21
2.2.9.	piBase Extension / Abstract Plugin	21
2.2.10.	Composer	21
2.2.11.	PSR-2	22
2.2.12.	CamelCase Schreibweise	23
2.3.	Vorbereitung	25
2.3.1.	Testumgebung	25
2.3.1.1.	Voraussetzungen	25
2.3.1.2.	Installation von TYPO3	26
	Aufruf im Browser	27
	TYPO3 Install-Tool	30
	TYPO3 TypoScript Konfiguration	32

2.3.1.3. Best practice	33
AdditionConfiguration	33
DevelopmentConfiguration	33
2.3.2. Einrichtung der IDE (PhpStorm)	36
2.3.2.1. Aufnahme der Core-Dateien	37
2.3.2.2. Debugging Live Templates	38
2.3.2.3. PhpStorm Plugin TypoScript	39
2.3.2.4. PhpStorm Plugin Fluid	39
2.3.2.5. PhpStorm Plugin TYPO3 CMS	40
2.3.2.6. PhpStorm Plugin .ignore	40
2.3.2.7. TYPO3 Coding Guidelines	42
2.3.2.8. Diverses	42
3. Grundlegende Informationen zu einer Extbase-Erweiterung	43
3.1. Dateistruktur	43
3.2. Datenstruktur	47
4. Die Erste Extension	48
4.1. Kickstart mit dem Extension Builder	48
4.2. Die erste Ausgabe	50
4.2.1. Einfügen neuer Seiten	50

4.2.2.	Einfügen von Testdatensätzen.....	50
4.2.3.	Einfügen eines Frontend-Plugins	51
4.2.4.	Ausgabe	52
5.	Blick unter die Haube	53
5.1.	Caching oder doch nicht?	53
5.2.	Blick in den Controller	54
5.2.1.	Die Methode listAction()	55
5.2.2.	Die Methode showAction().....	55
5.2.3.	Das Attribut \$personRepository.....	55
5.2.4.	Nützliche Methoden im Controller	56
5.2.5.	Attribute im Controller	58
5.3.	Der View	58
5.3.1.	Aufteilung im Template.....	59
5.3.2.	Übergabe von Parametern vom Controller an den View.....	60
5.3.3.	Der eigentliche Teil des Renderings	61
5.4.	Das Repository	63
5.4.1.	Vorhandene Methoden im Repository	64
5.4.1.1.	Vorhandene Methoden zum Lesen	64
5.4.1.2.	Vorhandene Methoden zum Ändern.....	65

5.4.2.	Vorhandene Methoden überschreiben.....	66
5.4.3.	Neue Methode implementieren.....	67
5.4.4.	Eigene Queries.....	69
5.4.5.	Eigene SQL-Abfragen	71
5.5.	Das Model.....	73
6.	Fluid.....	74
6.1.	Unterteilung in Templates/Layouts/Partials.....	76
6.2.	ViewHelper	77
6.2.1.	Verfügbare ViewHelper.....	78
6.2.2.	Inline- und Outline-Schreibweise	81
6.2.3.	Eigene ViewHelper erstellen.....	82
7.	Links und Hilfe	85
7.1.	Links	85
7.2.	Literatur	85
7.3.	Kontakt	86

2. Einführung

2.1. Grundlegendes

Nachfolgende Beispiele und Screenshots beziehen sich ausschließlich auf **TYPO3 8 LTS**. Da die Entwicklung von Extbase und Fluid in großen Schritten vorangetrieben wird, ergeben sich deutliche Änderungen im Laufe der Zeit und nur bedingte Rückwärtskompatibilität.

So ist z.B. der Einsatz von Namespaces erst ab TYPO3 6.0 möglich.

Das bedeutet auch, dass eine Erweiterung, die unter TYPO3 6.2 erstellt wurde, sehr wahrscheinlich nicht oder nur teilweise unter TYPO3 8.7 lauffähig und umgekehrt.

Die hauptsächlich in TYPO3 verwendete Scriptsprache ist **PHP**. Dies zieht sich natürlich auch bis in die Extbase-Programmierung durch.

Upd.	A/D	Extension	Key	Version	State	Actions
		Extbase Framework for Extensions	extbase	8.7.0	stable	

Tipp: Die Extension Extbase ist bereits standardmäßig installiert und aktiviert

Andere Erweiterungen können die mitgebrachten Funktionen von Extbase nutzen. Hierdurch ergeben sich einige positive Effekte:

- Schnellere Erstellung von Erweiterungen durch **Convention over Configuration** und durch Nutzung vorhandener Methoden, die sich häufig nutzen lassen (Schnellere time-to-market TTM)
- Saubere Gliederung des Codes in dafür vorgesehene Bereiche
- Sehr flexibel und erweiterbar
- Nutzung der ursprünglichen „Brückentechnologie“ erleichtert die Arbeit in anderen Systemen (Flow, Neos, Symfony, Zend, etc...)

2.2. Begriffserklärung

2.2.1. Extbase

Sehr viele Benutzer verbinden den Begriff **Extbase** irgendwie im Zusammenhang mit „moderner Programmierweise in TYPO3“. Dies ist natürlich nicht falsch, dennoch ist Extbase im ersten Schritt nichts Anderes als eine zusätzliche TYPO3-Erweiterung, die weitere Funktionalität ins System mitbringt.

Entstanden ist Extbase ursprünglich aus einem kleinen Backport aus TYPO3 Flow. Im Vergleich mit anderen PHP-Frameworks kann Extbase natürlich nicht mithalten, bringt aber dennoch bereits eine Menge bekannter Funktionen mit.

Upd.	AID	Extension	Key	Version	State	Actions
		Fluid Templating Engine	fluid	8.7.0	stable	

Tipp: Die Extension fluid ist bereits standardmäßig installiert und aktiviert

2.2.2. Fluid

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

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

Fluid kann auch direkt in der TYPO3-Integration verwendet werden. Dies geht mit dem **cObject FLUIDTEMPLATE** (siehe TSref oder Administrationsschulungsunterlagen von in2code).

2.2.3. Objektorientierte Programmierung

Mit Extbase hält die objektorientierte Programmierung (OOP) Einzug in TYPO3. Bei OOP handelt es sich um ein Programmierparadigma mit der Grundidee, Daten und Funktionen möglichst eng in einem sogenannten Objekt zusammenzufassen, so dass Methoden fremder Objekte diese Daten nicht versehentlich manipulieren können.

2.2.3.1. Prozedurale Programmierung (früher)

```
<?php  
  
    // Anfang des Programmes  
    ...  
    // Ende des Programmes  
?>
```

Merkmale der Prozeduralen Programmierung:

- Das Programm wird vom Anfang bis zum Ende durchlaufen
- Mehrfach verwendete Codestücke sind in Funktionen ausgelagert, die verstreut im Code liegen können

2.2.3.2. Objektorientierte Programmierung (heute)

Als Grundlage gilt es die Wirklichkeit möglichst einfach, verständlich und originalgetreu in Objekten nachzubauen. Hierbei gilt:

- Alles wird als Objekt gesehen
- Objekte stehen in Verbindung zueinander
- Jedes Objekt hat Attribute und Methoden
- Objekte können voneinander erben
- Jedes Objekt kann bestimmen, wie von außen auf die Attribute und Methoden zugegriffen werden kann

Klassen sind dabei der Bauplan für ein Objekt. Die Klasse definiert, welche Eigenschaften und Möglichkeiten ein Objekt hat. Das Objekt ist die individuelle Ausprägung dieser Klasse.

2.2.3.3. Das Auto-Beispiel

Aus der realen Welt

Sie erhalten die Aufgabe, eine Software zu entwickeln, mit der man ein Fahrzeug steuern können soll. Auch wenn PHP nicht unbedingt die perfekte Wahl ist, lässt sich auch hier die Wirklichkeit vereinfacht in der Scriptssprache darstellen.



Der Fokus liegt im ersten Schritt bei der Fortbewegung und der Scheinwerfersteuerung

Eigenschaften (Attribute)	Was kann man machen (Methoden)
Scheinwerferstatus (an/aus)	Scheinwerfer aktivieren oder deaktivieren bei Knopfdruck
Geschwindigkeit	Beschleunigen oder Abbremsen bzw. zu voreingestellter Geschwindigkeit wechseln (Tempomat)

```
class Fahrzeug {  
  
    protected $scheinwerfer = false;  
    protected $geschwindigkeit = 0;  
  
    public function beschleunigen() {  
        $this->geschwindigkeit += 5;  
    }  
    public function abbremsen() {  
        $this->geschwindigkeit -= 10;  
        if ($this->geschwindigkeit < 0) {  
            $this->geschwindigkeit = 0;  
        }  
    }  
    public function aktiviereTempomat() {  
        $this->geschwindigkeit = 100;  
    }  
    public function getGeschwindigkeit() {  
        return $this->geschwindigkeit;  
    }  
  
    public function schalteLicht() {  
        if ($this->scheinwerferStatus === false) {  
            $this->scheinwerferStatus = true;  
        } else {  
            $this->scheinwerferStatus = false;  
        }  
    }  
}
```

Kurzerklärung

Die Klasse Fahrzeug hält zwei für uns wichtige Eigenschaften (Scheinwerfer mit der Grundeinstellung „false“ und Geschwindigkeit mit „0“). Diese beiden Eigenschaften sollen nicht von außen veränderbar sein, daher wurden diese als protected gekennzeichnet. Über Methoden in unserer Klasse lassen sich die Eigenschaften definiert manipulieren.

Über beschleunigen() erhöht sich die aktuelle Geschwindigkeit um den Wert 5. Mit abbremsen reduziert sie sich um 10. Bei aktiviereTempomat() wird die Geschwindigkeit sofort auf 100 gestellt. Mit dem Getter getGeschwindigkeit() erhalten wir die aktuelle Geschwindigkeit (z.B. für eine Ausgabe im Tachometer).

Daneben lässt sich das Licht durch einen Knopfdruck an- oder ausschalten. Hierzu muss lediglich die Methode schalteLicht() aufgerufen werden.

Alle diese Methoden sind als public ausgezeichnet und daher von außen aufrufbar.

Tipp: Weiterführende Informationen zu OOP

http://de.wikipedia.org/wiki/Objektorientierte_Programmierung

<http://www.sebadorn.de/2011/05/07/php-und-oop-teil-2-sichtbereiche-public-protected-private>

Zugriffsregelungen

Deklarationen von Attributen und Methoden als:

Public

Direkter Zugriff von außen möglich

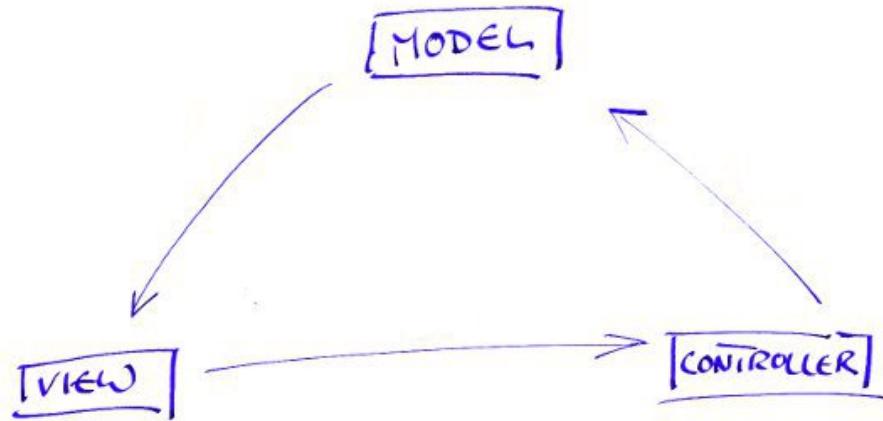
Protected

Interner Zugriff, Zugriff von
Tochterklassen aus möglich

Private

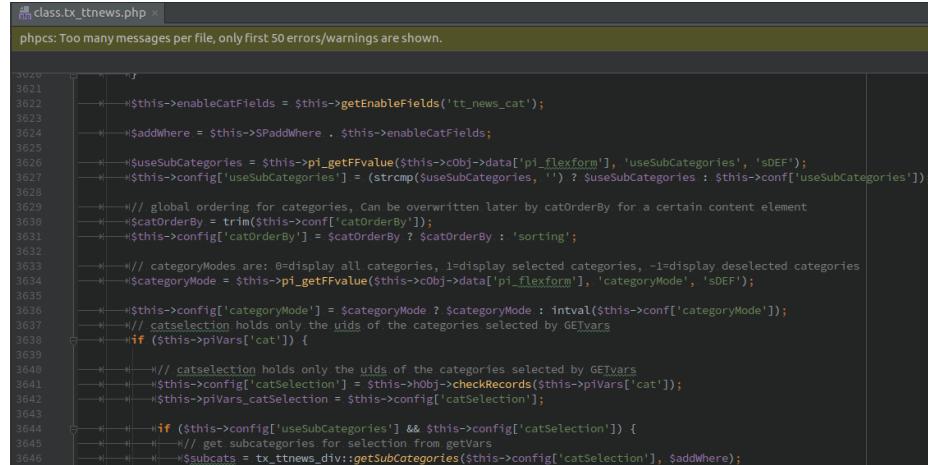
Nur interner Zugriff

In der OOP werden Attribute nicht als publik deklariert. Eine Änderung und das Auslesen erfolgen ausschließlich über Getter- und Setter-Methoden. Nur so können übergreifende Logiken eingebaut und erhalten werden.



2.2.4. Das MVC-Konzept

Model View Controller (MVC) ist ein Muster zur Strukturierung von Software-Entwicklung in drei Einheiten. Ziel ist hierbei ist ein flexibler Programmentwurf, der eine spätere Änderung oder Erweiterung möglichst erleichtert. Die Wiederverwendbarkeit der Programmierung steht also deutlich im Vordergrund. Extbase bringt auch das MVC-Konzept in die TYPO3-Extension -Entwicklung.



The screenshot shows a code editor with the file 'class.tx_ttnews.php' open. The status bar at the top says 'phpcs: Too many messages per file, only first 50 errors/warnings are shown.' The code itself is heavily annotated with numerous error and warning messages, indicating significant issues with the code's structure and logic.

```
3621 // $this->enableCatFields = $this->getEnableFields('tt_news_cat');  
3622 // $addWhere = $this->SPaddWhere . $this->enableCatFields;  
3623 // $useSubCategories = $this->pi_getFValue($this->cObj->data['pi_flexform'], 'useSubCategories', 'sDEF');  
3624 // $this->config['useSubCategories'] = ($useSubCategories == '') ? $useSubCategories : $this->conf['useSubCategories'];  
3625 // global ordering for categories, can be overwritten later by catOrderBy for a certain content element  
3626 // $catOrderBy = trim($this->conf['catorderBy']);  
3627 // $this->config['catorderBy'] = $catOrderBy ? $catOrderBy : 'sorting';  
3628 // categoryModes are: 0=display all categories, 1=display selected categories, -1=display deselected categories  
3629 // $categoryMode = $this->pi_getFValue($this->cObj->data['pi_flexform'], 'categoryMode', 'sDEF');  
3630 // $this->config['categoryMode'] = $categoryMode ? $categoryMode : intval($this->conf['categoryMode']);  
3631 // catselection holds only the uids of the categories selected by GETvars  
3632 if ($this->piVars['cat']) {  
3633 // catselection holds only the uids of the categories selected by GETvars  
3634 if ($this->config['catSelection'] = $this->hObj->checkRecords($this->piVars['cat']));  
3635 $this->piVars_catSelection = $this->config['catSelection'];  
3636 // if ($this->config['useSubCategories'] && $this->config['catSelection']) {  
3637 // get subcategories for selection from getVars  
3638 // $subcats = tx_ttnews_div::getSubCategories($this->config['catSelection'], $addWhere);  
3639 }  
3640 }  
3641 }  
3642 }  
3643 }  
3644 }  
3645 }
```

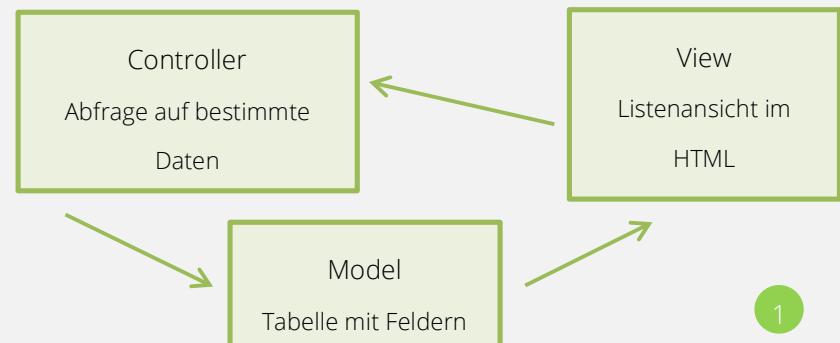
Beispieldatei *class.tx_ttnews.php* aus *tt_news* 7.6.3 mit über 4100 Zeilen Code (7.3.2017)

2.2.4.1. So nicht...

Wenn sich alle Merkmale einer Extension innerhalb einer Datei befindet, wird diese schnell unübersichtlich anwachsen. In einem schlechten Beispiel könnte sich neben der Logik auch noch HTML-Definition in der gleichen Datei oder sogar Funktion befinden.

Eine flexible Skalierung wird hierdurch erschwert oder teilweise sogar unmöglich gemacht.

Vorteil des MVC anhand einer TYPO3-Erweiterung



2.2.4.2. Die Trennung

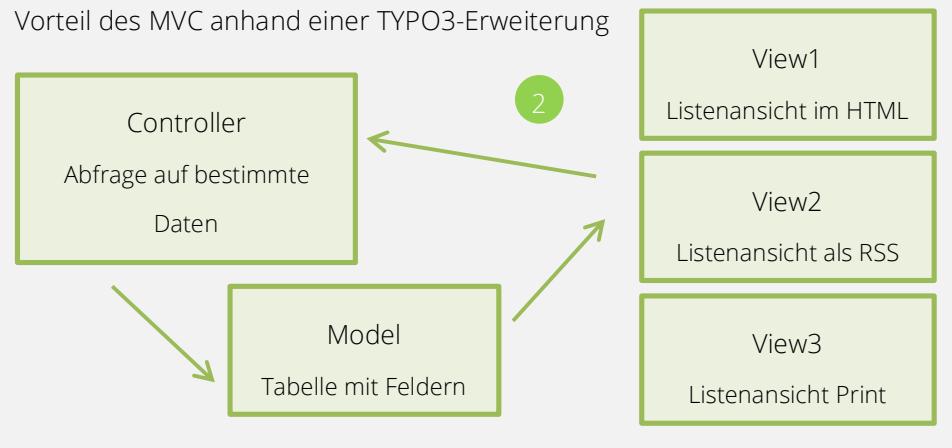
1

Hierbei handelt es sich um eine Extension mit einer klassischen Listenansicht von bestimmten Daten innerhalb eines bestimmten Zeitraums mit einer speziellen Ausgabe. Eine mögliche Anforderung wäre nun, diese Listenansicht auch als Druckversion und als RSS-Feed bereitzustellen.

2

Hier wird ersichtlich, dass sich die Programmierung im Controller und im Modell nicht ändern müssen – lediglich weitere Ausgaben waren gefordert. Eine Ergänzung ist hier schnell möglich.

Vorteil des MVC anhand einer TYPO3-Erweiterung



Durch die Trennung in verschiedene Bereiche (Logik, Ausgabe, Modell) ergeben sich nachfolgende **Vorteile**:

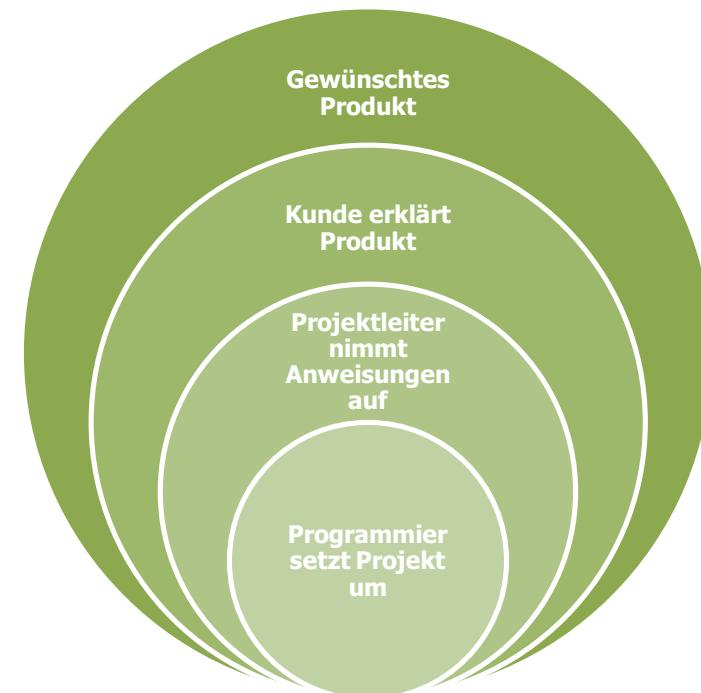
- Einfache Funktionsänderungen
- Einfache Erweiterung
- Spezialisierung für eine Schicht möglich (z.B. HTML-Profi für Ausgabeschicht)

2.2.5. Domain Driven Design

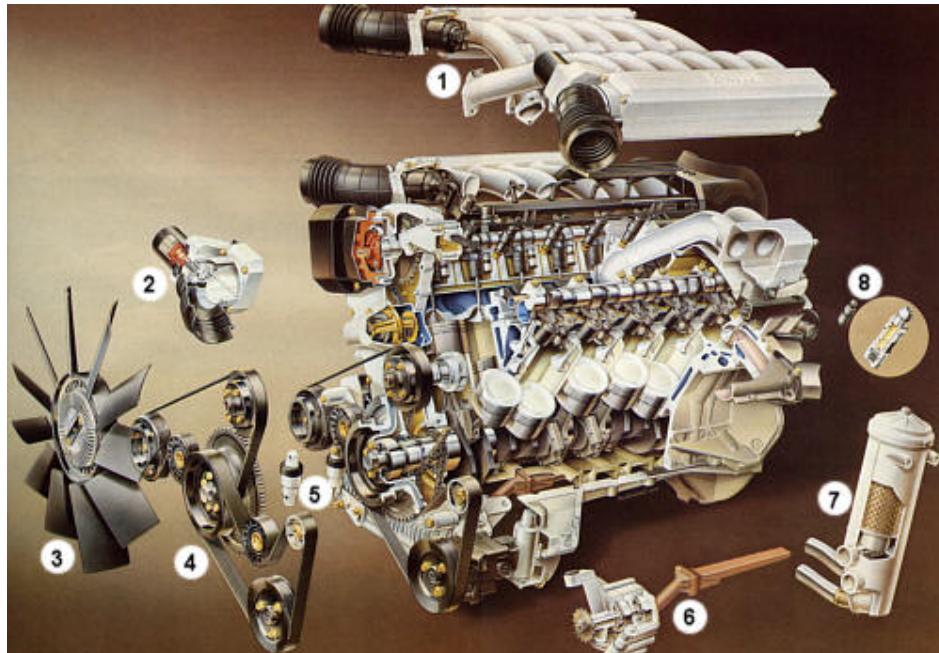
Domain-Driven Design (DDD) ist nicht nur eine Technik oder Methode. Es ist vielmehr eine Denkweise und Priorisierung zur Steigerung der Produktivität von Softwareprojekten im Umfeld komplexer fachlicher Zusammenhänge.

Domain-Driven Design ist eine Herangehensweise an die Modellierung komplexer Fachlichkeiten. Dabei wird eine Gliederung der Software durch eine Schichtenarchitektur vorausgesetzt. Das Hauptaugenmerk von Domain-Driven Design liegt dabei auf der Domänenenschicht.

2.2.5.1. Das Problem in der Softwareentwicklung



Konzepte werden unterschiedlich verstanden und interpretiert. Das finale Produkt weist zwangsläufig Differenzen auf, die es auszubessern gilt.



Modell als Abstraktion und Vereinfachung der Realität.

2.2.5.2. Begriffe

Domain

Als **Domain** (dt. Domäne) bezeichnet man einen Themenbereich, der inhaltlich spezialisiert ist. Beispiele:

- Architektur des Architekten
- Pädagogik der Lehrer
- Fahrzeugbau für KFZ-Mechaniker/-Mechatroniker

Komplikationen können vor allem dadurch auftreten, dass Programmierer in den seltensten Fällen Experten in der gleichen Domain wie ihr Kunde sind!

Modell

Als **Modell** versteht man die vereinfachte Beschreibung der Wirklichkeit. Ein Modell versteht sich als Vorbereitung zur technischen Umsetzung der Anforderung. Ein Modell wird gemeinsam zwischen Domänenexperten und Applikationsentwickler erstellt. Das Modell wird von allen Projektbeteiligten verstanden.

2.2.5.3. Lösungsansatz

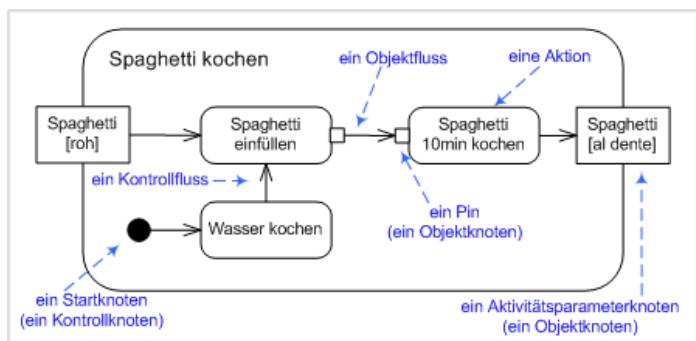


Keine gemeinsame Sprache

Lehrer: Ich möchte, dass bei der Benotung die individuelle Bezugs-normorientierung berücksichtigt wird.

Entwickler/Projektmanager: Eine Lösung mit JavaScript auf Basis von jQuery erscheint mir sinnvoll wobei die Sprachdateien in einem XML ausgelagert werden sollte.

Gemeinsames Modellieren einer Datenstruktur mit einer **gemeinsamen Sprache** (Ubiquitous Language) in einem (z.B.) UML



Umsetzung des UML in eine Programmstruktur mit Hilfe von Extbase.

2.2.6. Convention over Configuration

Der wohl verständlichste Vorteil von Extbase im Vergleich zu piBase ist die Reduzierung der Komplexität durch **vorgegebene Konventionen** (Konvention vor Konfiguration).

Ein einfaches Beispiel ist der Ort des HTML-Templates. Während es bei piBase dem Entwickler völlig frei gestellt war, wo und ob er eine HTML-Datei mitliefert, erwartet eine Action in einem Extbase-Controller bereits eine Datei mit einer bestimmten Bezeichnung an einem bestimmten Ort. Der Entwickler muss sich also nicht auch noch um die Einbindung und Verarbeitung der Template-Dateien kümmern.

Selbstverständlich lässt sich dieses Verhalten auch aufbrechen und weiter individualisieren, ergibt aber nur in den wenigsten Fällen wirklich Sinn (Edge Case).

2.2.7. Namespaces

TYPO3 wurde in der Version 6.0 komplett (also auch Extbase) auf PHP Namespaces umgestellt.

Um die Gefahr von gleich benannten Klassennamen auszuschließen, muss man lange Klassenname oder Namensräume benutzen. Da pro Datei nur eine Klasse zulässig ist, ergibt sich per Konvention aus Pfad + Dateiname der entsprechende Klassenname. (Vendor\ExtensionName\Verzeichnis\Datei => typo3conf/ext/extkey/Classes/Verzeichnis/Datei.php)

Klassenname	Pfad/Datei
tx_extension_domain_model_car	EXT:extension/Classes/Domain/Model/Car.php
\Vendor\Extension\Domain\Model\Car	EXT:extension/Classes/Domain/Model/Car.php

2.2.8. Vendor Name

Mit der Einführung von Namespaces wurden auch **Vendors** eingeführt. Der Vendor Name ist der kleinste gemeinsame Nenner zwischen Extensions aus einer Hand. Üblicherweise nutzen einzelne Entwickler oder Gruppierungen einen fixen Vendor (z.B. „In2code“ für Packages/Extensions von in2code) Somit kann das Package mit dem Namen „News“ von mehreren Anbietern entwickelt werden – z.B. UmbrellaCorporation/News

2.2.9. piBase Extension / Abstract Plugin

Unter einer piBase-Extension versteht man umgangssprachlich eine TYPO3-Erweiterung, die nicht auf die Methoden und Classen einer Extbase-Datei zurückgreift, um zu funktionieren. Bei einer Vielzahl der verfügbaren, freien Erweiterungen im TYPO3 Extension Repository (TER) handelt es sich noch um piBase-Extensions.

Solche Extensions lassen sich nur noch mit Umwegen in einer aktuellen TYPO3-Umgebung nutzen.

2.2.10. Composer

Zur Installation von TYPO3 empfiehlt sich der Einsatz des PHP-Package-Managers Composer. Über eine JSON-Konfiguration wird TYPO3 mit allen Abhängigkeiten per Befehl auf der Konsole heruntergeladen und zusammengebaut.

```
{  
    "repositories": [  
        {  
            "type": "composer",  
            "url": "https://composer.typo3.org/"  
        }  
    ],  
    "require": {  
        "typo3/cms": "8.*",  
        "typo3-ter/news": "*"  
    }  
}
```

Beispiel composer.json Datei um TYPO3 8 LTS mit der aktuellsten Version der Extension news zu installieren

2.2.11. PSR-2

Eine PHP Standard Recommendation (**PSR**) ist eine PHP-Spezifikation, welche durch die PHP Framework Interop Group veröffentlicht wird. Ziel ist es die Interoperabilität von Komponenten zu ermöglichen und eine gemeinsame technische Basis zu schaffen oder bewährte Konzepte für einen guten Programmierstil sowie eine gute Testbarkeit von Komponenten umzusetzen. Verschiedene Frameworks wie z. B. die der TYPO3-Community, Symfony oder Zend implementieren hierbei PSR-Spezifikationen in einem selbst gewählten Umfang.

Die **PSR-2 Spezifikation** beinhaltet Vorgaben zur Formatierung einer PHP-Datei (siehe <https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md>). Die IDE hilft einem sich an die Vorgaben zu halten. Über die Installation eines PHP-Sniffers lässt sich dies noch weiter vertiefen.

Als wichtigste Merkmale in PSR-2 gelten: Einrückung mit 4 Leerzeichen, Öffnende Klammern in Funktionen und Klassen auf einer neuen Zeile,

Klassenname ist identisch zum Dateinamen (ohne Endung) und keine schließenden PHP-Tags

```
<?php
declare(strict_types=1);
namespace In2code\Powermail\Domain\Service;

/**
 * Class DoSomethingService
 */
class DoSomethingService
{
    /**
     * @param string $string
     * @return string
     */
    public function trimString(string $string): string
    {
        return trim($string);
    }
}
```

Beispiel PHP-Datei Ext:powermail/Classes/Domain/Service/DoSomethingService.php im PSR-2 Format

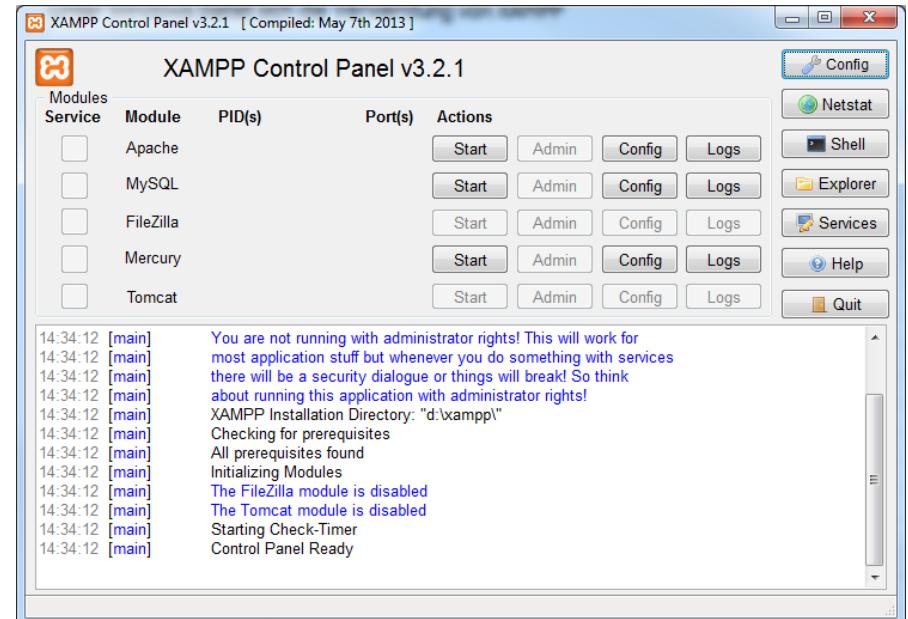
2.2.12. CamelCase Schreibweise

Innerhalb von TYPO3 lassen sich verschiedene Schreibweisen von Dateien, Methoden, Classen, etc... beobachten. Extbase bringt CamelCase auf den Plan – also eine Schreibweise, bei der mehrere, zusammengesetzte Wörter, innerhalb eines Begriffs durch einen Großbuchstaben optisch voneinander abgegrenzt werden. Mittlerweile ist dieser Ansatz in PSR-2 übernommen worden.

Historisch und technisch bedingt gibt es also nun eine Vielzahl von Möglichkeiten, die teilweise auch etwas verwirrend erscheinen. So gibt es z.B. in TypoScript und in der Datenbank derzeit leider alle drei Schreibweisen (z.B. „ATagParams“, „bodyTagCObject“ oder „index_enable“ / bzw. in tt_content „ CType“, „colPos“ und „fe_group“) um Rückwärtskompatibilität einzuhalten.

Hinweis: Beachten Sie in jedem Fall die Groß- und Kleinschreibung nachfolgender Beispiele

Schreibweisen	„Umbrella Corporation“	Vorkommen
Lower Underscored	umbrella_corporation	Tabellennamen, alte Methodenamen (z.B. Hooks)
Upper CamelCase	UmbrellaCorporation	Moderne Datei- und Verzeichnisnamen sowie PHP-Klassennamen.
Lower CamelCase	umbrellaCorporation	PHP Variablen- oder Methodenamen



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

```

Beispiel: PHP 7 unter Ubuntu

2.3. Vorbereitung

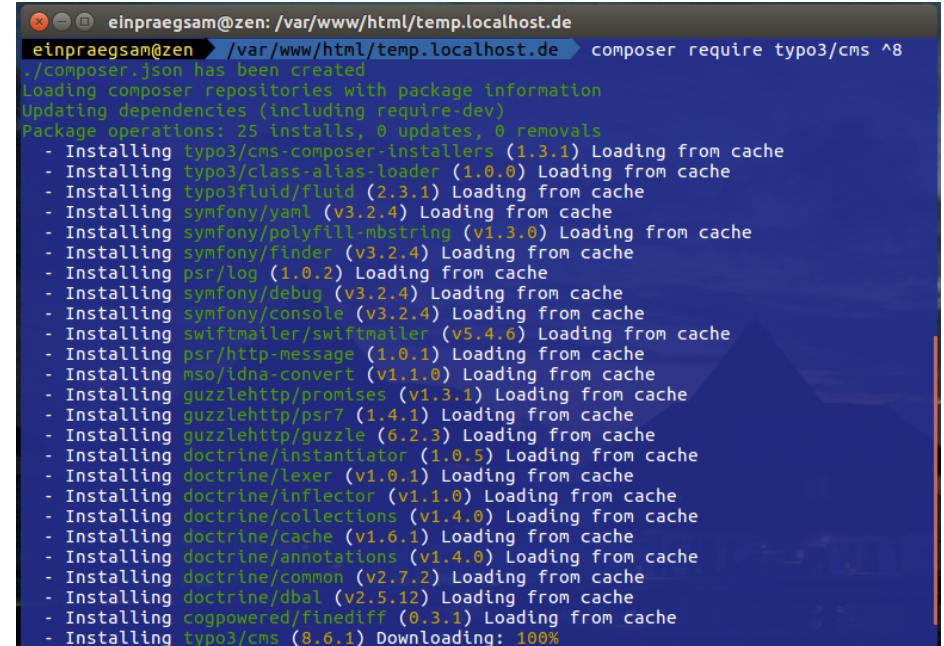
2.3.1. Testumgebung

Eine lokale Testumgebung ermöglicht die Entwicklung einer TYPO3-Erweiterung möglichst ohne großen Zeitverlust – auch offline.

2.3.1.1. 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.

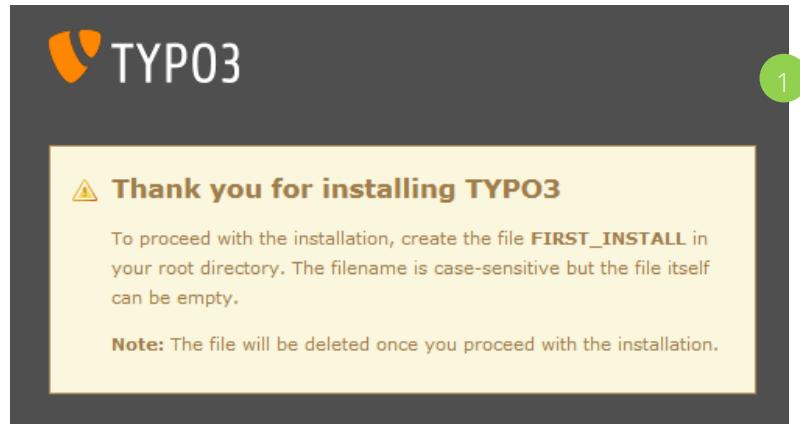
Entsprechende VHost und hosts Einträge müssen gemacht werden, damit die Seite(n) später lokal unter einer bestimmten Domain verfügbar ist (Z.B. soll beim Aufruf von extension.localhost.de im Browser der lokale Apache angesprochen werden – siehe <http://wiki.schmidtmarcel.de/xampp-vhosts-einrichten/>).



```
einpraegsam@zen: /var/www/html/temp.localhost.de
einpraegsam@zen ➤ /var/www/html/temp.localhost.de ➤ composer require typo3/cms ^8
./composer.json has been created
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 25 installs, 0 updates, 0 removals
- Installing typo3/cms-composer-installers (1.3.1) Loading from cache
- Installing typo3/class-alias-loader (1.0.0) Loading from cache
- Installing typo3fluid/fluid (2.3.1) Loading from cache
- Installing symfony/yaml (v3.2.4) Loading from cache
- Installing symfony/polyfill-mbstring (v1.3.0) Loading from cache
- Installing symfony/finder (v3.2.4) Loading from cache
- Installing psr/log (1.0.2) Loading from cache
- Installing symfony/debug (v3.2.4) Loading from cache
- Installing symfony/console (v3.2.4) Loading from cache
- Installing swiftmailer/swiftmailer (v5.4.6) Loading from cache
- Installing psr/http-message (1.0.1) Loading from cache
- Installing mso/idna-convert (v1.1.0) Loading from cache
- Installing guzzlehttp/promises (v1.3.1) Loading from cache
- Installing guzzlehttp/psr7 (1.4.1) Loading from cache
- Installing guzzlehttp/guzzle (6.2.3) Loading from cache
- Installing doctrine/instantiator (1.0.5) Loading from cache
- Installing doctrine/lexer (v1.0.1) Loading from cache
- Installing doctrine/inflector (v1.1.0) Loading from cache
- Installing doctrine/collections (v1.4.0) Loading from cache
- Installing doctrine/cache (v1.6.1) Loading from cache
- Installing doctrine/annotations (v1.4.0) Loading from cache
- Installing doctrine/common (v2.7.2) Loading from cache
- Installing doctrine/dbal (v2.5.12) Loading from cache
- Installing cogpowered/finediff (0.3.1) Loading from cache
- Installing typo3/cms (8.6.1) Downloading: 100%
```

2.3.1.2. 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).



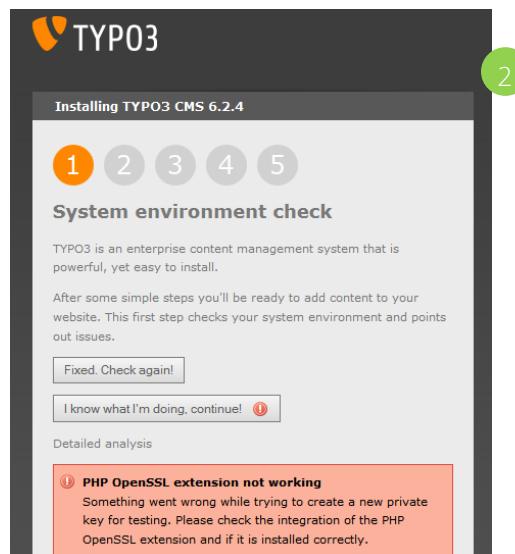
Aufruf im Browser

1

Wie in der Meldung dargestellt, muss eine Datei mit Namen „FIRST_INSTALL“ im Root-Verzeichnis (z.B. D:\xampp\htdocs\websites\testumgebung1) erstellt werden. Danach die Seite neu laden.

2

Der 1. Schritt der Installationsroutine sollte nun erscheinen.



Installing TYPO3 CMS 6.2.4

1 2 3 4 5

Database connection

If you have not already created a username and password to access the database, please do so now. This can be done using tools provided by your host.

Username

Password

Type

Host

Port

TYPO3 CMS native database implementation is based on MySQL. A database abstraction layer allows to run TYPO3 CMS on different database engines like postgres. This is used rather seldom and some core parts and extensions do not fully support this. Your TYPO3 CMS experience might suffer if you choose to install the system on anything different than MySQL.

I do not use MySQL

1

Durch Klick auf „I know what I'm doing, continue“ kommt man zu Schritt 2: Bei Username sollte der Benutzername „root“ eingetragen werden. Der Rest kann unverändert bleiben (Passwort leer ist so in Ordnung).

2

Der Schritt 3 erscheint. Wir wollen eine neue Datenbank erstellen. Diese nennen wir genauso wie unsere Testumgebung „testumgebung“ und gehen zu Schritt 4.

Installing TYPO3 CMS 6.2.4

1 2 3 4 5

Select database

You have two options:

Use an existing empty database:

OR create a new database:
Attention: The database user must have sufficient privileges to create the whole structure.
Enter a name for your TYPO3 database.

2

Installing TYPO3 CMS 6.2.4

1 2 3 4 5

Create user and import base data

Import basic database structure and create a backend administrator user. The password can be used to log in to the **install tool** and the **TYPO3 CMS backend** (default username is "admin").

Username: admin *

Password: *

Show password

⚠ This password gives an attacker full control over your instance if cracked. It should be strong (include lower and upper case characters, special characters and numbers) and must be at least eight characters long.

Site name: New TYPO3 site *

Continue

1

Hier muss ein Benutzername und Passwort für den Backend-Login eingegeben werden.

Achtung: Das Passwort wird zugleich für den Login zum Install-Tool verwendet.

Installing TYPO3 CMS 6.2.4

1 2 3 4 5

Installation done!

The only thing remaining is to set some configuration values based on your system environment, which happens automatically in this step. Then you will be redirected to your TYPO3 CMS backend, ready for you to log in with the user you just created.

Want a pre-configured site?

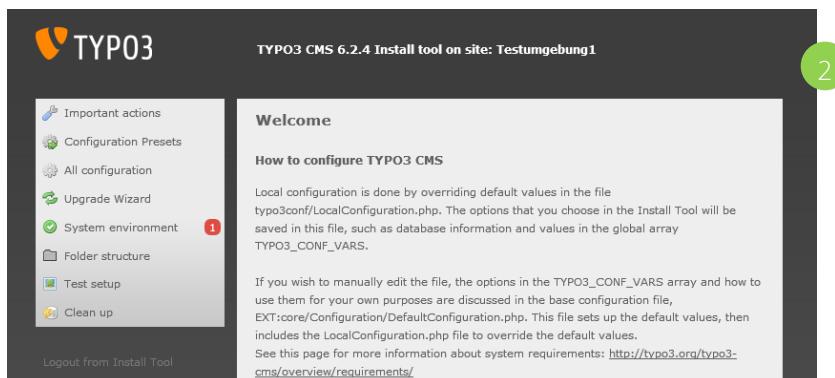
You now have an empty installation. If you want a pre-configured site, there are distributions on the web which can be installed via the Extension Manager. If you check the option below, the list of distributions will be fetched and you will be able to choose one directly. **Please note: This may take some time after login.**

Yes, download the list of distributions.

Open the backend

2

Den Download der Distribution-Liste kann man sich in der Regel ersparen. Durch Klick auf „Open the backend“ gelangt man nun zum Backend-Login.



1

TYPO3 Install-Tool

Durch Aufruf der URL /typo3/install gelangt man zum Install-Tool. Um dies nutzen zu können, muss noch eine Datei mit Namen „ENABLE_INSTALL_TOOL“ im typo3conf/ Verzeichnis erstellt werden.

2

Nach einem Reload kann man sich mit dem vergebenen Passwort ins Install-Tool einloggen.

Alle Änderungen die man im Install-Tool macht und speichert, werden als PHP-Array in der Datei typo3conf/LocalConfiguration.php eingetragen. Somit kann man diese Datei natürlich auch direkt bearbeiten.

Um mit der Umgebung ideal entwickeln zu können, sollte unter „Configuration Presets“ im Bereich „Development“ der Radio-Button „Development“ angehakt werden. Im Anschluss muss diese Einstellung durch Klick auf „Activate“ gespeichert werden.

Soll mit Bildern gearbeitet werden, müssen Image Magick oder Graphics Magick installiert werden. Die Pfade hierzu lassen sich im Install Tool (oder direkt in der typo3conf/LocalConfiguration.php) hinterlegen.

Hinweis: Der Login ins Backend funktioniert eventuell nicht korrekt. Das liegt üblicherweise an der Methode RSA. Entweder diese wird im Install-Tool auf „normal“ gestellt (alternativ direkt in der typo3conf/LocalConfiguration.php) oder der Pfad zu OpenSSL wird definiert.

```
page = PAGE
page.10 < styles.content.get

#####
page.20 = HMENU
page.20 {
    wrap = <hr /> |
    1 = TMENU
    1.expAll = 1
    1.wrap = <ul>|</ul>
    1.NO = 1
    1.NO.wrapItemAndSub = <li>|</li>

    2 < .1
    3 < .1
}

#####
config.linkVars = L

config.sys_language_uid = 0
config.language = de
config.locale_all = de

[globalVar = GP:L=1]
config.sys_language_uid = 1
config.language = en
config.locale_all = en_EN
[end]
```

TYPO3 TypoScript Konfiguration

Damit eine Frontend-Ausgabe klappt, muss in einer frischen TYPO3-Installation eine Minimalkonfiguration vorgenommen werden.

Hierzu empfehlen wir zu allererst das Anlegen von 2-3 Testseiten mit etwas Testinhalt.

Ein Main TypoScript erlaubt die Ausgabe von Seiteninhalt im Frontend (wichtig ist zudem die Einbindung des Statischen TypoScript der Erweiterung fluid_styled_content im Tab Includes).

2.3.1.3. Best practice

AdditionConfiguration

Es empfiehlt sich ein Verzeichnis typo3conf/ext/AdditionalConfiguration/ anzulegen. Danach dieser 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 AdditionalConfiguration/ mit dem Aufbau *Configuration.php inkluded.

DevelopmentConfiguration

Über eine typo3conf/ext/AdditionalConfiguration/DevelopmentConfiguration.php können Caches deaktiviert und Fehlermeldungen aktiviert werden:

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

// MISC
$GLOBALS['TYPO3_CONF_VARS']['BE']['sessionTimeout'] =
9999999999;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['enableDeprecationLo
g'] = '1';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['curlUse'] = 1;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['curlTimeout'] = 10;
```

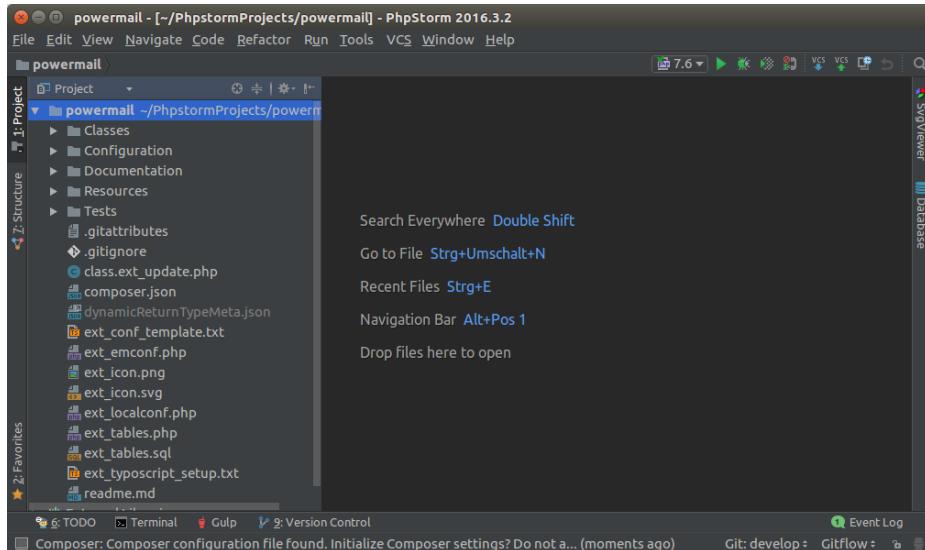
```
// DEBUG
$GLOBALS['TYPO3_CONF_VARS']['BE']['debug'] = 1;
$GLOBALS['TYPO3_CONF_VARS']['BE']['compressionLevel'] =
0;
$GLOBALS['TYPO3_CONF_VARS']['FE']['debug'] = 1;
// $GLOBALS['TYPO3_CONF_VARS']['FE']['pageNotFound_handling'] = '';
$GLOBALS['TYPO3_CONF_VARS']['FE']['compressionLevel'] =
0;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['displayErrors'] =
'1';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['devIPmask'] = '*';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['sqlDebug'] = '1';
\TYPO3\CMS\Core\Utility\ExtensionManagementUtility::add
TypoScript(
    'developmentConfiguration',
    'setup',
    'config.contentObjectExceptionHandler = 0'
);

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

```
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheCon
figurations']['cache_pagesection']['backend'] =
'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheCon
figurations']['cache_phpcode']['backend'] =
'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheCon
figurations']['cache_rootline']['backend'] =
'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheCon
figurations']['extbase_datamapfactory_datamap']['backen
d'] = 'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheCon
figurations']['extbase_object']['backend'] =
'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheCon
figurations']['extbase_reflection']['backend'] =
'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheCon
figurations']['extbase_typo3dbbackend_queries']['backen
d'] = 'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheCon
figurations']['extbase_typo3dbbackend_tablecolumns']['b
ackend'] = 'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheCon
figurations']['l10n']['backend'] =
'TYPO3\CMS\Core\Cache\Backend\NullBackend';
$GLOBALS['TYPO3_CONF_VARS']['EXT']['extCache'] = '0';
```

```
// MAIL
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['defaultMailFromAddress'] = 'alex@in2code.de';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['defaultMailFromName'] = 'defaultmailfromaddress';
$GLOBALS['TYPO3_CONF_VARS']['EXT']['powermailDevelopContextEmail'] = 'alexander.kellner@in2code.de';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport'] =
'smtp';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_password'] = 'password';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_server'] = 'smtp.einpraegsam.net:25';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_useUsername'] = 'lokal@einpraegsam.net';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_port'] = '25';
```

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



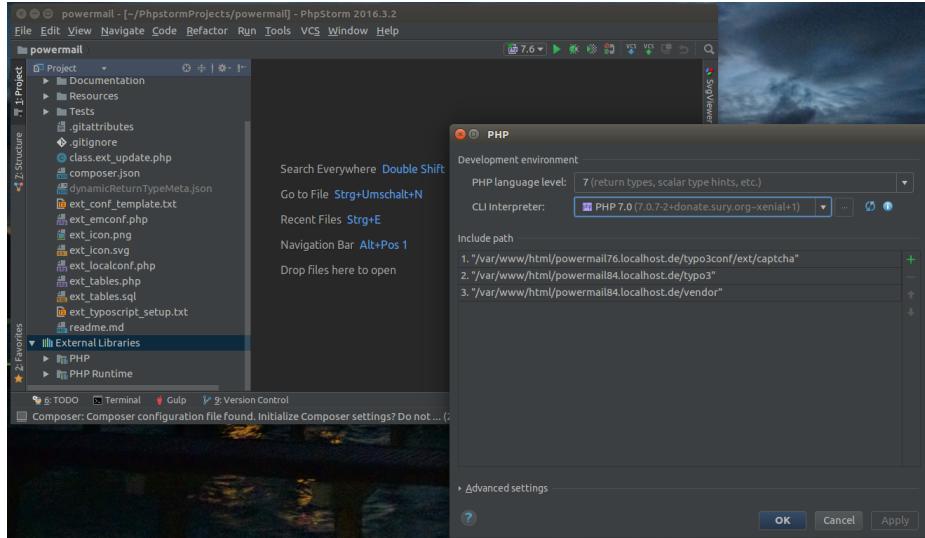
Beispiel eines PhpStorm-Projektes am Beispiel der Extension powermail

2.3.2. Einrichtung der IDE (PhpStorm)

Beim Entwickeln werden in erster Linie PHP-, HTML-, JavaScript- und XML-Dateien erzeugt und bearbeitet. Dies lässt sich mit einem einfachen Editor bewerkstelligen.

Entscheidet man sich jedoch für eine IDE (Integrated Development Environment), stehen einem eine große Auswahl an Hilfsfunktionen zur Verfügung, die die Programmierdauer drastisch senken und die Codequalität erhöhen können.

Bekannte IDE-Software sind Netbeans, Eclipse oder PhpStorm. Eine Vielzahl von TYPO3-Core- und Extension-Entwickler haben sich für PhpStorm entschieden. Nachfolgende Beispiele setzen daher auf PhpStorm (<http://www.jetbrains.com/phpstorm/>) auf.

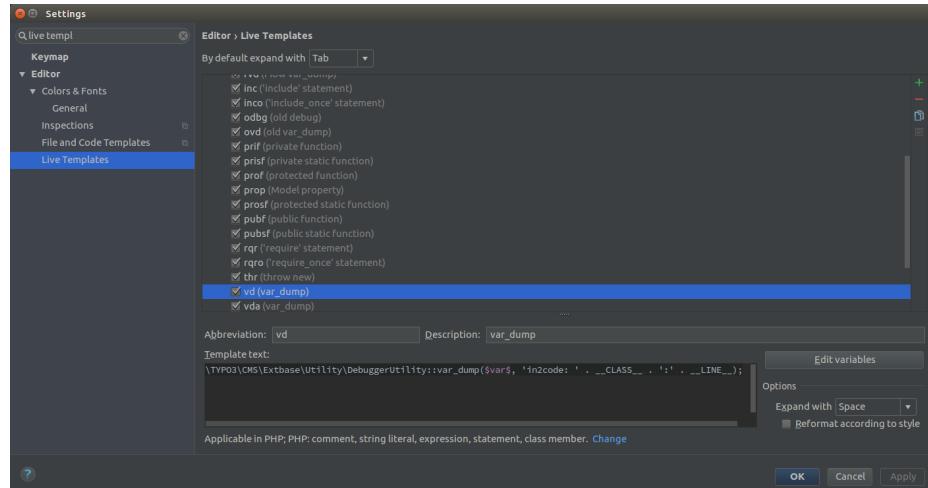


2.3.2.1. Aufnahme der Core-Dateien

Damit eine Code-Auto vervollständigung funktioniert, muss die IDE alle Dateien und Klassen von TYPO3 kennen. Haben Sie also den direkten Pfad zu einer Extension gewählt an Stelle den Pfad zum Webroot, fehlten der IDE die Dateien von TYPO3 (mit Extbase und Fluid)

Als letztes Element im Seitenbaum gibt es „External Libraries“. Rechtsklick darauf erlaubt die Auswahl eines oder mehrerer Ordner mit den TYPO3 Core Dateien oder weiteren Extensions.

Hinweis: Für die Auto-Vervollständigung spielt es übrigens keine Rolle, wo der ausgewählte Code liegt und ob dieser Verwendung beim Rendering über den Apache findet.



Hinweis: Nach Hinzufügen nicht vergessen bei „No applicable context yet.“ Auf „Define“ zu klicken und PHP auszuwählen.

2.3.2.2. Debugging Live Templates

TYPO3 unterstützt verschiedene Debugging-Methoden. Diese lassen sich durch ein Tastenkürzel in PhpStorm einbinden. Um dies einzurichten, muss man File/Settings/Live Templates und dann PHP bzw. HTML auswählen.

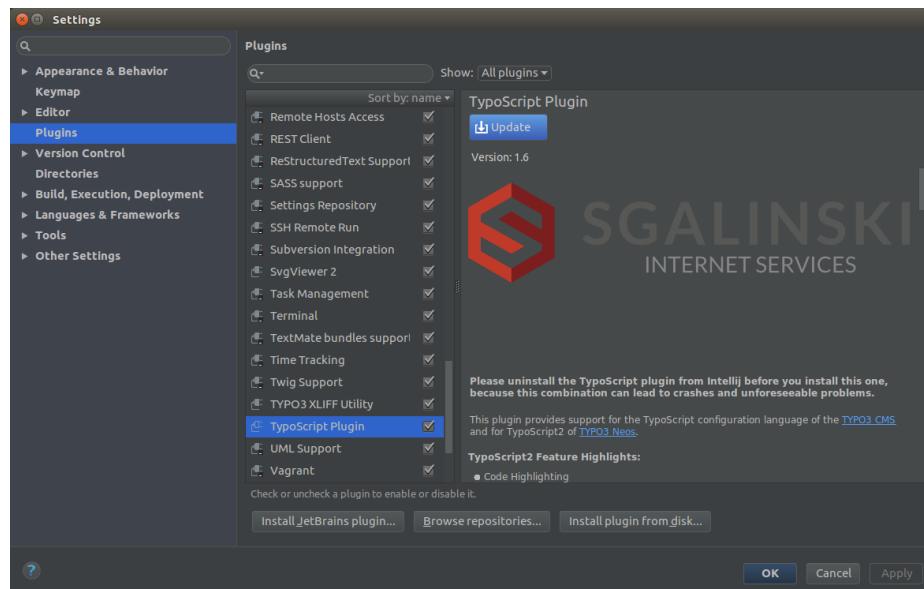
Empfohlene Live Templates:

Kürzel	Template Text	Dateityp
vd	\TYPO3\CMS\Extbase\Utility\DebuggerUtility::var_dump(\$var\$, 'in2code: ' . __CLASS__ . ':' . __LINE__);	PHP
vd	<f:debug>\$var\$</f:debug>	HTML

2.3.2.3. PhpStorm Plugin TypoScript

Wird viel TypoScript bei der Extension Entwicklung oder bei der Integration verwendet, so ist das „TypoScript Plugin“ von sgalinski Internet Services sehr nützlich.

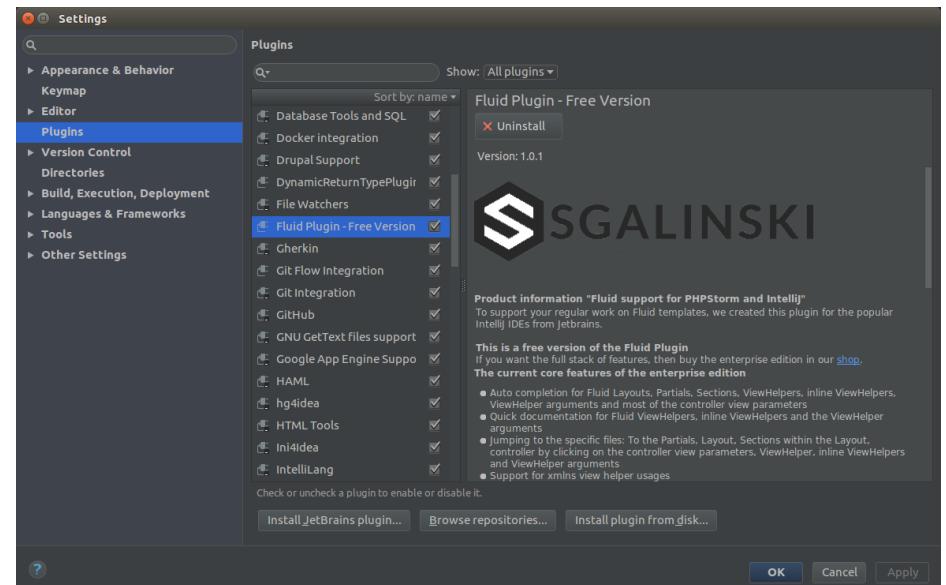
Nach Einbindung gibt es Auto vervollständigung und Codeanalyse für TypoScript (und TypoScript2) in TypoScript-Dateien.



2.3.2.4. PhpStorm Plugin Fluid

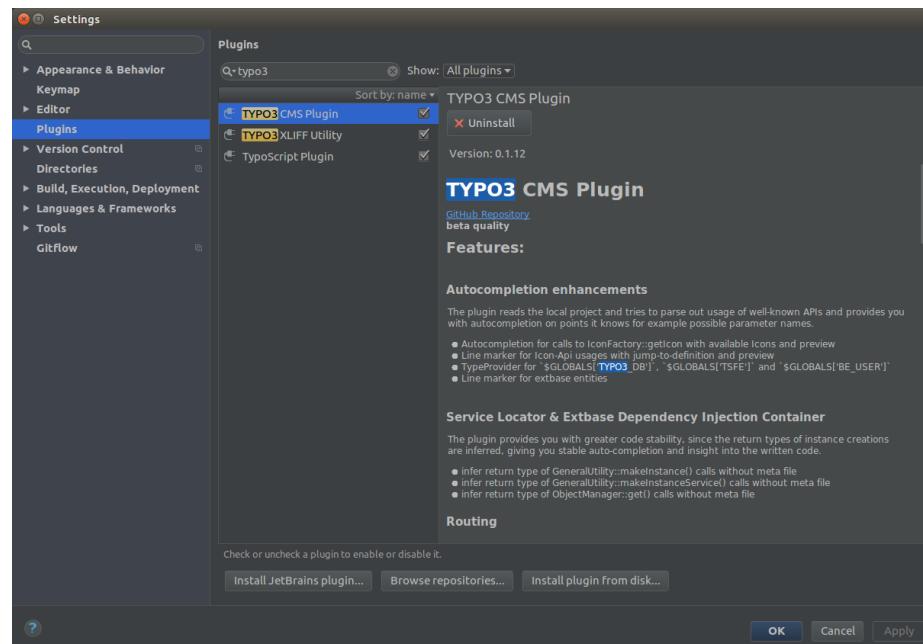
Um Auto vervollständigung in FLUID-Templates genießen zu können, empfiehlt sich das Plugin „Fluid Plugin – Free Version“.

Nach einer Installation ist eventuell ein Neustart von PhpStorm notwendig.



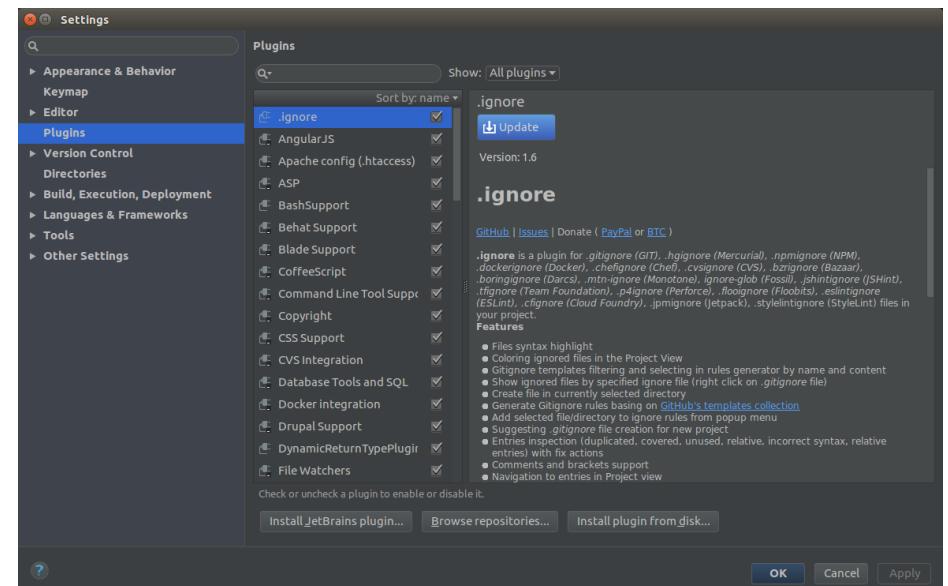
2.3.2.5. PhpStorm Plugin TYPO3 CMS

Damit die IDE erkennt, dass es sich beim Rückgabewert von GeneralUtility::makeInstance() oder ObjectManager->get() um eine Objektinstanz handelt, empfiehlt sich das TYPO3 CMS Plugin. Darüber hinaus gibt einem das Plugin im Controller ein paar nützliche Hinweise.



2.3.2.6. PhpStorm Plugin .ignore

Bei der Arbeit mit GIT und mit Composer möchte man eventuell Unterstützung bei der Bearbeitung der Datei .gitignore haben. Darüber hinaus werden über Composer hinzugeladene Extensions und Verzeichnisse grau dargestellt.



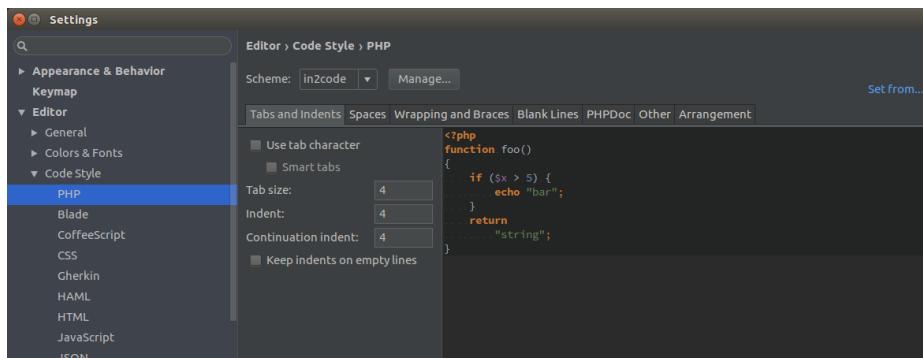
in2code GmbH
Kunstmühlstraße 12a
83026 Rosenheim

t +49 8031 8873983
f +49 8031 8873985

info@in2code.de
www.in2code.de

2.3.2.7. TYPO3 Coding Guidelines

TYPO3 empfiehlt sich an entsprechende Coding-Vorgaben zu halten (siehe <http://docs.typo3.org/typo3cms/CodingGuidelinesReference/>). Dank PSR-2 ist es sehr einfach, dies in PhpStorm voreinzustellen. Unter Settings/Editor/Code Style/PHP kann man „Set from“ und „PSR1/PSR2“ auswählen.

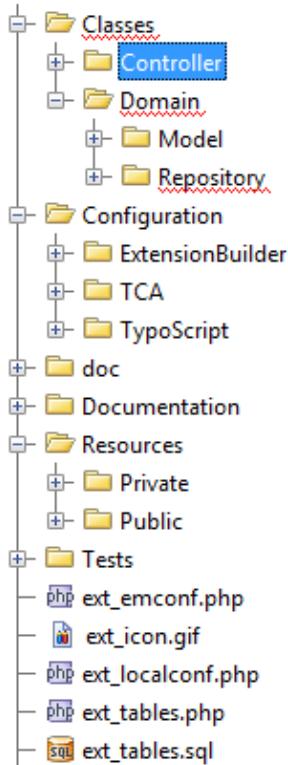


Hinweis: Wenn man sich für den PHP Code Sniffer in PhpStorm entscheidet (siehe <https://blog.jetbrains.com/phpstorm/2014/06/using-php-code-sniffer-in-phpstorm/>) bekommt man bereits beim Schreiben entsprechende Warnhinweise.

2.3.2.8. Diverses

PhpStorm kann noch deutlich mehr – hier ein kleiner Auszug hilfreicher Funktionen:

- Ausführung von Unitests
- Unterstützung von XDEBUG
- Unterstützung von Versionierungssoftware wie GIT oder SVN
- Ausführung von NPM- oder Gulp-Tasks
- Ausführung von composer Befehlen
- Zugriff auf die lokale Datenbank
- PhpStorm Plugins erweitern die Funktionalität noch mehr (z.B. GIT Flow Plugin)



Beispielerweiterung mit `extension_builder` erstellt

3. Grundlegende Informationen zu einer Extbase-Erweiterung

3.1. Dateistruktur

Eine Extbase-Extension lässt sich sofort an der markanten Dateistruktur erkennen. Diese Datei- und Ordnerstruktur ist teilweise zwingend nötig. Neu ist die Schreibweise von Dateien innerhalb einer solchen Erweiterung. Dateinamen werden in UpperCamelCase geschrieben (siehe Dateien und deren Bedeutung)

Im Root-Verzeichnis einer Extbase-Extension befinden sich einige Dateien, die es auch in piBase Extensions gibt. Alleine die Schreibweise (durchgehend mit lowerunderscored) deutet bereits darauf hin, dass es sich hierbei nicht um Extbase-spezifische Files handelt

Dateien und deren Bedeutung

Dateiname	Erklärung
ext_emconf.php	<p>„emconf“ steht für Extension Manager Configuration</p> <p>Beim Öffnen fällt auf, dass hier lediglich ein Array mit Aussagen zur Extension zu finden ist. Diese Art der Aussagen werden vom Extension Manager für dessen Auflistung benötigt.</p>
ext_icon.gif	<p>Hierbei handelt es sich um das Icon der Extension. Alternativ kann mittlerweile auch eine ext_icon.png oder ext_icon.svg verwendet werden.</p>
ext_localconf.php	<p>Ähnlich wie die LocalConfiguration.php (ehemals localconf.php) befindet sich hier die Grundkonfiguration der Extension. Unter anderen werden hier Extbase und piBase Frontend-Plugins ins TYPO3 eingebunden.</p>

ext_tables.php

Einsatzzwecke:

- Einbindung von statischem TypoScript
- Einbindung von TSconfig
- Backend-Module in TYPO3 einbinden
- Flexform Konfiguration einbinden

ext_tables.sql

Diese Datei besteht aus einer Reihe von SQL-Anweisungen, die bei der Erstellung eigener Tabellen in der vorhandenen Datenbank wichtig sind.

Hinweis: Obwohl über jeder Tabelle ein „CREATE TABLE...“ steht, werden Tabellen für TYPO3 nicht erneut angelegt, wenn diese bereits vorhanden sind. Updates sind natürlich weiterhin möglich.

Verzeichnisname	Erklärung
Classes	Im Verzeichnis Classes befindet sich alle PHP-Dateien wie Controller, Model, sowie Repository (und unter Umständen weitere Funktionen wie Validatoren, eigene Classen und ViewHelper)
Controller	Hier befindet sich das Herzstück der Extension. Es bündelt die Action Methoden und initialisiert die Ausgabe. Es können beliebig viele Controller angelegt werden. Dateinamenkonvention: [ControllerName]Controller.php

Domain	Innerhalb von Domain sollte sich die Domänenlogik befinden. Standardmäßig umfasst dies alle Modelle und Repositories.
Model	Im Model werden die Model-Dateien hinterlegt, die das Aussehen von Objekten beschreiben. Dateinamenkonvention: [ModelName].php
Repository	Hier befinden sich alle Zugriffsfunktionen auf die Datenbank. Dateinamenkonvention: [RepositoryName]Repository.php
Configuration	In Configuration befindet sich das TypoScript, TCA- und Flexform-Einstellungen

doc	Die Schreibweise deutet auf eine historisch-bedingte Funktion hin. Hier befindet sich die Dokumentation in Form einer manual.sxw (OpenOffice) Datei. Falls die Extension im TER bereitgestellt wird, wird die doc/manual.sxw geparsst und online angezeigt.	Public	Im Ordner Public befinden sich alle zusätzlichen Dateien zur Extension, die man direkt über den Browser öffnen können soll (z.B. CSS, JavaScript, etc...).
Documentation	Hier befindet sich die neue Dokumentation im RST-Format (damit dies bei einem Upload ins TER übernommen werden kann).	Tests	Dieser Ordner wird vom extension_builder (wenn benutzt) bereits angelegt und bildet den ersten Baustein beim Test-Driven-Development.
Resources	Unter Resources befinden sich alle zusätzlichen Dateien – üblicherweise gibt es lediglich zwei Ordner (Private/Public) – siehe nachfolgende Beschreibung.		
Private	Im Ordner Private befinden sich zusätzliche Dateien zur Extension, die nicht von außen (Browser) zugänglich sind. In der Regel sind das HTML-Templates und Sprachdateien.		

3.2. Datenstruktur

Alle, von der Extension angelegten Tabellen, folgen auch einer strikten

Namenskonvention:

tx_[extKey]_domain_model_[Objektnname]

 **tx_address_domain_model_address** InnoDB utf8_general_ci 16 384 49 152 0 3 ~ 2

The screenshot shows the TYPO3 Extension Builder interface. On the left, the 'Extension properties' panel is open, showing the extension key 'person' and a description: 'This is a training extension - to show persons'. Below this are sections for 'Persons', 'Frontend plugins', and 'Backend modules'. In the center, a 'New Model Object' dialog is open for creating a 'Person' model. The 'Domain object settings' section includes 'Object type: Entity', 'Is aggregate root: checked', and 'Enable sorting: checked'. The 'Default actions' section lists 'list', 'show', 'new/create', 'edit/update', and 'delete'. The 'Properties' section contains two properties: 'name' (String, required) and 'email' (String, excluded). A 'Relations' section is also present.

Beispiel Erweiterung mit dem Extension Key „person“

4. Die Erste Extension

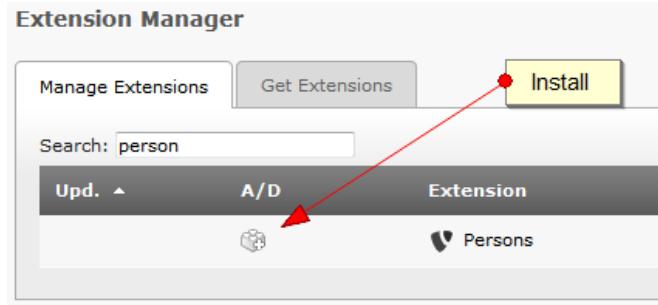
4.1. Kickstart mit dem Extension Builder

Mit Hilfe der Erweiterung „extension_builder“ lässt sich sehr schnell eine Extension erstellen. Dies ist nicht zwingend nötig, erspart aber eine Menge Tipparbeit.

Hierzu bitte die Extension „extension_builder“ aus dem TER laden und installieren. Nach einem Reload des Backends steht den Administratoren ein gleichnamiges Modul zur Verfügung.

Bitte erstellen Sie nun eine Erweiterung mit nebenstehenden Eigenschaften und klicken unten in der Mitte auf Save.

In der linken Spalte beschreiben wir die Extension im Allgemeinen und geben des Weiteren an, dass wir ein Frontend-Plugin benötigen. In der rechten Spalte lässt sich über Drag'n Drop ein Neues Objekt-Modell erzeugen.

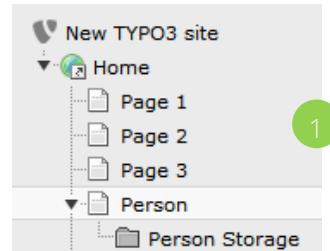


Im Anschluss muss die neue Extension noch installiert werden. Dies funktioniert im Extension Manager.

Wichtig wenn TYPO3 im Composer-Mode läuft, ist das richtige Autoloading der PHP-Dateien. Wenn Extensions nicht über Composer hinzugeladen werden, wie es in unserem Fall geschehen ist, muss in der composer.json der Pfad zu den PHP-Dateien hinterlegt werden:

```
{  
    ...  
    "autoload": {  
        "psr-4": {  
            "Test\\Persons\\": "typo3conf/ext/persons/Classes"  
        }  
    }  
}
```

Im Anschluss den Befehl „composer dump-autoload“ auf der Kommandozeile ausführen. Danach sollten unsere Dateien automatisch eingebunden werden.



Create new Person on page "Person Storage"

General	Access
Language:	Default
Hide:	<input type="checkbox"/>
Name	Alex Kellner
Email	alexander.kellner@in2code.de

4.2. Die erste Ausgabe

Im Prinzip steht die neue Erweiterung bereits in den Startlöchern und kann auch schon Datensätze aus der Datenbank anzeigen.

Hierzu müssen wir jedoch noch im Backend ein-zwei Datensätze sowie ein Plugin einfügen.

4.2.1. Einfügen neuer Seiten

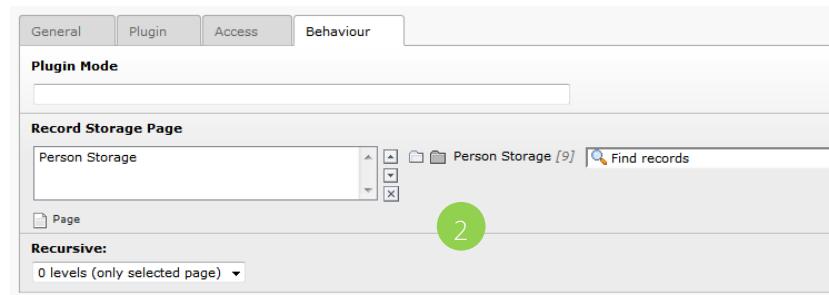
1

Bitte fügen Sie eine Seite für das FE-Plugin (z.B. „Person“) sowie einen Folder Unterhalb (z.B. „Person Storage“) hinzu.

4.2.2. Einfügen von Testdatensätzen

2

Über das Listenmodul können Sie im Folder „Person Storage“ nun ein paar Testdatensätze anlegen.



4.2.3. Einfügen eines Frontend-Plugins

1

Auf der Seite „Person“ können Sie nun einen neuen Seiteninhalt vom Typ Plugin einfügen. Im Anschluss wählen Sie das Plugin Persons aus.

2

Ganz wichtig ist außerdem den Ausgangsort mit den Datensätzen zu selektieren, sonst erhalten wir später keine Ausgabe.

in2code.
Wir leben TYPO3 Person

- [Page 1](#)
- [Page 2](#)
- [Page 3](#)
- [Person](#)
- [Newslist](#)

Listing for Person

First and Lastname	email
Alex Kellner	alexander.kellner@in2code.de Edit Delete
Tina Gasteiger	tina.gasteiger@in2code.de Edit Delete
Stefan Busemann	stefan.busemann@in2code.de Edit Delete

[New Person](#)

Copyright © 2013 Lala GmbH Deutschland

4.2.4. Ausgabe

1

Eventuell muss noch der Cache geleert werden, aber danach sollte bereits eine Listenansicht der Datensätze zur Verfügung stehen.

2

Beim Klick auf den Namen oder die E-Mail ist die Detailansicht sichtbar.

Single View for Person

First and Lastname Alex Kellner

email alexander.kellner@in2code.de

[Back to list](#)

[New Person](#)

5. Blick unter die Haube

Alle nachfolgenden Codebeispiele oder Screenshots beziehen sich auf die vorherige Extension.

```
\TYPO3\CMS\Extbase\Utility\ExtensionUtility::configure
Plugin(
    'TYPO3.' . $_EXTKEY,
    'Pi1',
    array(
        'Person' => 'list, show',
    ),
    // non-cacheable actions
    array(
        'Person' => '',
    )
);
```

5.1. Caching oder doch nicht?

Ein weiterer Vorteil bei der Benutzung eines Plugins mit Extbase ist die genaue Einstellung, welche Ansicht gecacht werden soll und welche nicht. In der Regel gilt, dass man jede Ansicht aus Performancegründen cachen sollte, in bestimmten Fällen oder in der Entwicklung kann es sinnvoll sein einzelne Views hiervon auszunehmen.

Beim Öffnen der Datei ext_localconf.php fällt uns die Einbindung des FE-Plugins auf.

Der Methode configurePlugin() werden 4 Parameter übergeben:

1. Der Extension Key
2. Der Name des Plugins (in der Regel Pi1, Pi2, Pi3 – Pi steht für Plugin)
3. Ein Array bestehend aus allen Controllern mit dessen Actions (Views)
4. Ein Array bestehend aus allen Controllern mit dessen Actions (Views) genau wie unter Punkt 3 mit dem Unterschied, dass jede angegebene Action nicht gecacht wird.

In der Regel wird die erste Action im ersten Controller aufgerufen, sollte im Frontend nichts anderes angegeben werden.

```
class PersonController extends \TYPO3\CMS\Extbase\Mvc\Controller\ActionController
{
    /**
     * @var \TYPO3\Person\Domain\Repository\PersonRepository
     * @inject
     */
    protected $personRepository;

    /**
     * @return void
     */
    public function listAction()
    {
        $persons = $this->personRepository->findAll();
        $this->view->assign('persons', $persons);
    }

    /**
     * @param \TYPO3\Person\Domain\Model\Person $person
     * @return void
     */
    public function showAction(\TYPO3\Person\Domain\Model\Person $person)
    {
        $this->view->assign('person', $person);
    }
}
```

5.2. Blick in den Controller

Der Controller besteht aus zwei Methoden (zwei Views oder auch zwei Actions) sowie einem Attribut. Jede beschriebene Action in der ext_localconf.php findet sich im entsprechenden Controller unter [actionName]Action() wieder.

Somit gibt es eine Listen- (listAction) und eine Detailansicht (showAction).

Hinweis: Standardmäßig wird immer der erste definierte Controller mit der ersten definierten Action (siehe ext_localconf.php) aufgerufen. Ein Umschalten lässt sich über GET-Parameter beim Aufruf der Seite erzwingen:
`index.php?id=1&tx_extkey_pi1[action]=actionName&tx_extkey_pi1[controller]=Controllername`

5.2.1. Die Methode listAction()

In der Methode listAction() wird in Zeile 50 auf alle vorhandenen Datensätze abgefragt (mit Hilfe von findAll() aus dem Repository). In der Zeile 51 werden diese Datensätze an den View übergeben.

Da es sich bei der listAction um den ersten View in der ext_localconf.php handelt, wird diese auch im Frontend angezeigt (sofern keine weiteren Parameter übergeben werden)

5.2.2. Die Methode showAction()

Die Methode unterscheidet sich zur listAction vor allem durch die Übergabe eines Parameters \$person.

In Zeile 61 wird dieser Personendatensatz wieder an den View übergeben.

Möchte man den Detailview im Frontend aufrufen, muss man das Extbase entsprechend mit &tx_person_pi1[action]=show mitteilen

Hinweis: Bei Aufruf der URL mit index.php?id=1&tx_person_pi1[action]=show erscheint ein entsprechender Fehler „An error occurred while trying to call TYPO3\Person\Controller\PersonController->showAction(). Error: Required property 'person' does not exist.“ Das liegt daran, dass die Methode showAction() den Parameter \$person erwartet. Der korrekte Aufruf lautet also index.php?id=8&tx_person_pi1[action]=show&tx_person_pi1[person]=1

5.2.3. Das Attribut \$personRepository

Mit Hilfe von Dependency Injection (siehe http://de.wikipedia.org/wiki/Dependency_Injection) und dem Zusatz @inject im Kommentar oberhalb wird das Object \$this->personRepository in der kompletten Klasse zur Verfügung gestellt. Somit funktioniert auch die Datenabfrage in der listAction()

5.2.4. Nützliche Methoden im Controller

Nachfolgend ein paar Beispiele wichtiger Methoden, die man immer wieder im Controller benötigt. Einige Methoden lassen sich auch im Model, Repository oder in einem ViewHelper nutzen.

Beispiel	Erklärung
<pre>\$this->view->assign ('name', 'wert')</pre>	Übergabe eines Wertes (String, Array oder Objekt) an Fluid
<pre>\$array = ['name' => 'wert', 'name2' => 'wert2']; \$this->view->assignMultiple (\$array);</pre>	Mehrere Werte an Fluid übergeben. Aufruf mit {name} oder {name2}

<pre>\$this->view-> setTemplatePathAndFilename()</pre>	Anpassen des Templatepfades (in der Regel nicht nötig, da über TypoScript möglich) für außergewöhnliche Fälle (z.B. E-Mail Template)
<pre>\$string = \$this->view->render()</pre>	Soll der Rückgabewert aus dem Fluidtemplate nicht direkt an TYPO3 übergeben werden, kann man auch in eine Variable rendern.
<pre>\$this->request->getArguments()</pre>	Abfrage auf alle Pluginvariablen (GET oder POST Parameter innerhalb von tx_ext_pi1)
<pre>\$this->request- >getArgument('action')</pre>	Abfrage auf ein bestimmtes Argument (GET oder POST Parameter innerhalb von tx_ext_pi1)

\$this->
>addFlashmessage('gespeichert!') Hiermit können Flashmessages für die Ausgabe im Fronend gespeichert werden.

\$this->forward('actionName') Weiterleitung auf eine andere Action (ohne Browserreload). Parameter:

1. Actionname
2. Controllername (leer: Aktueller Controller)
3. Extensionname (leer: Aktuelle Extension)
4. Weiter Argumente als array

\$this->redirect('actionName') Weiterleitung auf eine andere Action (mit Browserreload). Gleiche Parameter wie \$this->forward()

\$this->configurationManager->getContentObject();

\TYPO3\CMS\Extbase\Utility\LocalizationUtility::translate('key',
\$this->extensionName)

initializeAction()

\$this->objectManager->get('ClassName')

Hiermit erhält man das zum Plugin passende ContentObject. Hilfreich um an Werte des aktuellen tt_content Datensatz zu gelangen.

Aufruf eines Wertes aus locallang.xlf

Bei Bedarf kann neben den normalen Actions eine initializeAction (ähnlich einem Constructor) eingebaut werden.

Ein vorhandenes Objekt instanzieren

5.2.5. Attribute im Controller

Einige Attribute werden von Extbase definiert und stehen zur Verwendung zur Verfügung.

Beispiel	Erklärung
\$this->settings	Werte aus dem TypoScript oder aus dem Flexform als array
\$this->extensionName	Extension Key (mit beginnendem Großbuchstaben)
\$this->actionMethodName	Aktuelle Action

5.3. Der View

Per Konvention ist festgelegt, dass bei Aufruf der listAction() im PersonController ein entsprechendes HTML-Template geladen wird. Extbase sucht die Datei automatisch entsprechend:
Resources/Private/Templates/Person>List.html

Wird die Detailansicht im Frontend aufgerufen, zieht die showAction() entsprechend ein Template nach sich:
Resources/Private/Templates/Person>Show.html

Weitere Informationen zum Templating in Fluid (siehe Fluid)

5.3.1. Aufteilung im Template

Vereinfachte Darstellung:

```
<f:layout name="Default" />  
Interne Notizen  
<f:section name="main">  
    HTML für Ausgabe  
</f:section>
```

Der Extension Builder hat das HTML-Template bereits so angelegt, dass ein Layout aufgerufen werden soll. Der Name Default weist auf folgende Pfad/Datei Kombination: Resources/Private/Layouts/Default.html

Die Layout Datei beinhaltet drei Zeilen und weist Extbase an, wieder zum Template zurückzuspringen und lediglich den Teil zwischen <f:section name="main"> und </f:section> zu rendern.

```
<div class="tx-person">  
    <f:render section="main" />  
</div>
```

5.3.2. Übergabe von Parametern vom Controller an den View

3 verschiedene Möglichkeiten stehen zur Verfügung, Werte mit der Methode assign() zu übergeben:

Über- gabe- wert	Beispiel	Aufruf in Fluid	Erklärung
String	\$this->view->assign ('value', 'der Wert');	{value}	Bei der Übergabe eines Strings kann man diesen direkt wieder in Fluid aufgreifen
Array	\$this->view->assign ('value', [0 => 'der Wert']);	{value.0}	Bei der Übergabe eines Arrays, kann man durch Trennung mit einem Punkt den entsprechenden Key nutzen

Objekt	\$persons = \$this->personRepository->findAll(); \$this->view->assign('persons', \$persons);	So bald ein Objekt mit Getter und Setter bereitsteht, lässt sich dieses ebenfalls an den View übergeben. Wenn hier also mit .name ein Wert aufgerufen wird, wird der getter getName() erwartet
--------	--	---

```
<table class="tx_person" >  
    <tr>  
        <th><f:translate  
key="tx_person_domain_model_person.name" /></th>  
        <th><f:translate  
key="tx_person_domain_model_person.email" /></th>  
        <th> </th>  
        <th> </th>  
    </tr>  
  
    <f:for each="{persons}" as="person">  
        <tr>  
            <td><f:link action="show"  
arguments="{person : person}"> {person.name}</f:link.action></td>  
            <td><f:link action="show"  
arguments="{person : person}"> {person.email}</f:link.action></td>  
            <td><f:link action="edit"  
arguments="{person : person}">Edit</f:link.action></td>  
            <td><f:link action="delete"  
arguments="{person : person}">Delete</f:link.action></td>  
        </tr>  
    </f:for>  
</table>  
  
<f:link.action action="new">New Person</f:link.action>  
</f:section>
```

5.3.3. Der eigentliche Teil des Renderings

Wenn man sich nun den Bereich zwischen den Section-Tags ansieht, erkennt man einen Mix aus HTML-Templates und Fluid-Anweisungen, die wie HTML-Tags aussehen – hierbei handelt es sich um ViewHelper.

Ein ViewHelper ist, wie der Name schon sagt, eine Unterstützung bei der Ausgabe von HTML-Code.

In den Zeilen 21 und 22 erkennt man beispielsweise einen Translate-ViewHelper. Dieser gibt einen lokalisierten (übersetzten) Text aus einer Übersetzungsdaten (locallang.xlf oder locallang.xml) je nach Frontend-Sprache aus.

Bei dem ForEach-ViewHelper in Zeile 27 handelt es sich um eine Schleife – das bedeutet, dass sämtlicher HTML-Code zwischen <f:for> und </f:for> so lange wiederholt wird, wie es Objekte in der Variable {persons} gibt. Zugleich stellt der ViewHelper eine neue Variable in der Schleife zur Verfügung {person}. Die Funktionsweise verhält sich also ähnlich zur PHP-Funktion foreach.

Innerhalb der Schleife lassen sich die Felder aus der Personentabelle mit
{objekt.feldname} wieder ausgeben (z.B. {person.email})

Einen weiteren ViewHelper kann man um jedes Feld in der Schleife erkennen:

Der Extension Builder hat jeden Zelleninhalt mit einem Link versehen

<f:link.action></f:link.action>. Hierbei lässt sich auf einen anderen View
verlinken und bereits Argumente mitgeben.

Weitere Informationen über ViewHelper (siehe ViewHelper)

```
namespace TYPO3\Person\Domain\Repository;  
  
/*  
 * Einige Kommentare  
 */  
  
class PersonRepository extends  
\TYPO3\CMS\Extbase\Persistence\Repository {  
  
}
```

5.4. Das Repository

Über das Repository lassen sich Datenbank-Abfragen gestalten.

Der Aufruf \$this->personRepository->findAll() im Controller bemüht das PersonRepository um eine Methode um alle Objekte zu finden.

Etwas befremdlich ist der erste Anblick der Datei PersonRepository.php, die vollkommen leer zu sein scheint.

Vermutlich wird der eine oder andere hier eine Funktion mit Namen findAll() erwartet haben, diese ist jedoch nicht zu sehen. Lediglich eine Klasse die wiederum erweitert wird ist sichtbar.

Die Extbase Klasse \TYPO3\CMS\Extbase\Persistence\Repository beinhaltet bereits die wichtigsten Repository-Abfragen, die wir nicht noch einmal aufnehmen müssen.

5.4.1. Vorhandene Methoden im Repository

5.4.1.1. Vorhandene Methoden zum Lesen

Methode	Erklärung
findAll()	Sucht alle Objekte innerhalb der Datenbank. Hinweis: Funktioniert nur wenn der Ausgangspunkt im Plugin oder per TypoScript gesetzt wurde
findByUid(\$uid)	Findet einen Datensatz an Hand der Uid
findBy[Feldname](\$string)	Es stehen automatisch Methoden zu jedem Feld bereit. Gibt es beispielsweise ein Feld namens „email“, gibt es auch die Methode findByEmail()

findOneBy[Feldname](\$string)

Die Funktionsweise ist ähnlich wie bei findBy[Feldname](), jedoch wird hier nur ein Objekt zurückgegeben (vergleichbar mit limit=1)

countAll()

Anzahl aller Objekte

countBy[Feldname](\$string)

Anzahl aller Objekte, die in einem bestimmten Feld einen Wert enthalten

5.4.1.2. Vorhandene Methoden zum Ändern

Methode	Erklärung
add(\$object)	Ein Objekt wird dem Repository hinzugefügt (ein neuer Datensatz wird erstellt)
remove(\$object)	Ein Objekt wird aus dem Repository entfernt (ein Datensatz wird aus der Tabelle gelöscht)
removeAll()	Löscht alle Datensätze
replace(\$object1, \$object2)	Ersetzt ein Objekt durch ein anderes

update(\$object)

Die Änderungen an einem Datensatz werden gespeichert

```
/**  
 * Find all persons  
 *  
 * @return query object  
 */  
  
public function findAll() {  
    $query = $this->createQuery();  
    return $query->execute();  
}
```

1

```
/**  
 * Find all persons  
 *  
 * @return query object  
 */  
  
public function findAll() {  
    $query = $this->createQuery();  
    $query->setOrderings(array('name' =>  
        \TYPO3\CMS\Extbase\Persistence\QueryInterface::ORDER_DESCENDING));  
    return $query->execute();  
}
```

2

5.4.2. Vorhandene Methoden überschreiben

In PHP5 kann man geerbte Methoden überschreiben – dies bezeichnet man als „überladen“. Dies kann in diesem Fall nützlich sein, wenn eine bereits vorhandene Methode nicht hundertprozentig das macht, was man benötigt.

1

Die Originalmethode lässt sich so nachbilden.

2

Wir wollen nun die Original-Methode um eine Zeile erweitern, damit die Ausgabe im Frontend nach dem Feld „name“ absteigend sortiert wird. Hierzu fügen wir eine weitere Zeile ein.

Hinweis: Beim Überladen von Methoden kann man die Anzahl der übergebenen Parameter nicht beeinflussen. Hat also die Original-Methode findAll() keine Parameter, so kann die neue Methode auch keine Parameter beinhalten

```
/**  
 * Find all persons  
 *  
 * @return query object  
 */  
  
public function getAllPersons() {  
    $query = $this->createQuery();  
  
    $query->setOrderings(array('name' =>  
        \TYPO3\CMS\Extbase\Persistence\QueryInterface::ORDER_DESCENDING));  
  
    return $query->execute();  
}
```

1

5.4.3. Neue Methode implementieren

1

Will man vorhandene Methoden nicht überschreiben oder braucht man noch weitere Repository Methoden, lassen diese sich natürlich mit einem beliebigen Methodennamen implementieren

2

Diese Methode lässt sich nun im Controller aufrufen.

```
public function listAction() {  
    $persons = $this->personRepository-  
        >getAllPersons();  
  
    $this->view->assign('persons', $persons);  
}
```

2

```
/**  
 * Find all persons and sort by given fieldname  
 *  
 * @param \string $field  
 * @return query object  
 */  
  
public function getAllPersons($field) {  
    $query = $this->createQuery();  
    $query->setOrderings(array($field =>  
        \TYPO3\CMS\Extbase\Persistence\QueryInterface::ORDER_DESCENDING));  
    return $query->execute();  
}
```

1

1

Natürlich lassen sich auch Parameter übergeben.

```
public function listAction() {  
    $persons = $this->personRepository->getAllPersons('email');  
    $this->view->assign('persons', $persons);  
}
```

```
/**  
 * Find all persons with given searchparam and email  
 *  
 * @param \string $searchparam  
 * @return query object  
 */  
  
public function getAllPersons($searchparam) {  
    $query = $this->createQuery();  
  
    $and = array(  
        $query->like('email', '%@%'),  
        $query->like('name', '%' . $searchparam .  
        '%')  
    );  
  
    $constraint = $query->logicalAnd($and);  
    $query->matching($constraint);  
  
    $result = $query->execute();  
    return $result;  
}
```

5.4.4. Eigene Queries

Natürlich lassen sich auch kompliziertere Abfragen erstellen. Im nachfolgenden Beispiel wird nach einem Suchbegriff (z.B. „alex“) gefiltert. Zusätzlich sollen nur Personen ausgegeben werden, die ein „@“ Zeichen im E-Mail-Feld enthalten.

Methoden im Überblick:

Methode	Erklärung
\$this->createQuery()	Erzeugt eine neue Query. Steht in der Regel am Anfang einer Repository-Funktion.
logicalAnd(\$constraints)	Verbindet einzelne Abfrage mit der logischen Funktion UND
logicalOr (\$constraints)	Verbindet einzelne Abfrage mit der logischen Funktion ODER
logicalNot(\$constraints)	Logische Funktion NOT zur Negierung
setOrderings()	Fügt Sortierung hinzu

setLimit()	Setzt ein Limit für die Abfrage (ein Integer-Wert wird erwartet)
setOffset()	Setzt einen Versatz – sinnvoll z.B. für einen eigenen Pagebrowser
count()	Zählt die Objekte
getFirst()	Gibt den ersten Datensatz zurück
execute()	Führt den Query aus. Steht in der Regel am Ende jeder Repository-Funktion.

```
public function getPageTitleByUid($uid) {  
    $query = $this->createQuery();  
  
    $sql = 'select title';  
    $sql .= ' from pages';  
    $sql .= ' where uid = ' . intval($uid);  
    $sql .= ' limit 1';  
  
    $query->getQuerySettings() ->setReturnRawQueryResult(true);  
    $result = $query->statement($sql)->execute();  
    return $result[0]['title'];  
}
```

5.4.5. Eigene SQL-Abfragen

Sollten die vorhandenen Methoden nicht ausreichen, ein bestimmtes SQL-Query zu erzeugen, lässt sich das auch selber machen.

Die Gründe warum man ein SQL-Query nicht selber ausführen sollte, liegen vor allem in der Flexibilität der Anwendung. Sollte in Zukunft auf eine SQL-Datenbank (nosql oder textdatei, etc...) verzichtet werden, funktioniert die eigene Abfrage nicht mehr.

```
namespace TYPO3\Person\Domain\Model;

class Person extends \TYPO3\CMS\Extbase\DomainObject\AbstractEntity {

    /**
     * First and Lastname
     *
     * @var \string
     * @validate NotEmpty
     */
    protected $name;

    /**
     * email
     *
     * @var \string
     */
    protected $email;

    /**
     * Returns the name
     *
     * @return \string $name
     */
    public function getName() {
        return $this->name;
    }
}
```

```
/**
 * Sets the name
 *
 * @param \string $name
 * @return void
 */
public function setName($name) {
    $this->name = $name;
}

/**
 * Returns the email
 *
 * @return \string $email
 */
public function getEmail() {
    return $this->email;
}

/**
 * Sets the email
 *
 * @param \string $email
 * @return void
 */
public function setEmail($email) {
    $this->email = $email;
}
}
```

5.5. Das Model

Im Model wird der Datensatz mit allen Eigenschaften beschrieben.

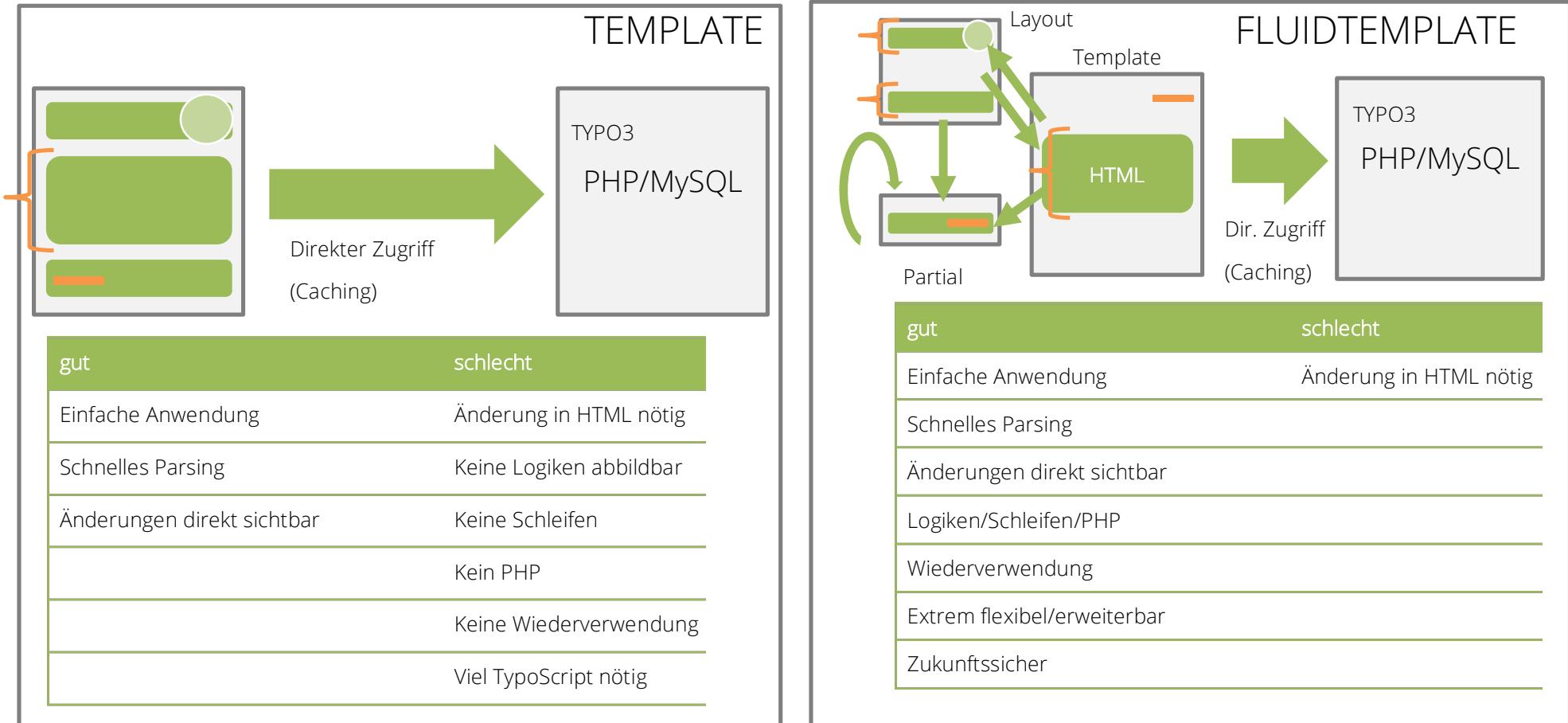
Wie im Absatz OOP erklärt, besteht ein Model nicht ausschließlich aus Attributen, über die man von außen zugreifen kann, sondern aus einem (oder mehreren) Attribut(en) mit eigenen Methoden, diese auszulesen oder zu verändern (Getter und Setter).

Sobald eine Methode getName() im Model verfügbar ist, lässt sich über Fluid mit {object.name} auf das gleichnamige Attribut zugreifen.

Hinweis: Über die Dokumentation oberhalb der Attribute und Methoden wird auch die Validierung der Werte bestimmt. Sollte also im Modell ein Attribut mit NotEmpty gekennzeichnet sein, ist es nicht möglich im Frontend Datensätze mit leeren Feldern zu zeigen. Dies führt unweigerlich zu Fehlermeldungen.

6. Fluid

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



Vergleich zwischen TEMPLATE und FLUIDTEMPLATE

6.1. Unterteilung in Templates/Layouts/Partials

Mit Fluid ist es möglich, Templates in drei Kategorien zu unterteilen:

Templates

Template ist das Haupt-HTML-Template. Dieses wird immer zuerst aufgerufen und beinhaltet in der Regel den Hauptteil der Ausgabe

Partials

Ein Partial ist eine Art Snippet. Hierbei handelt es sich also um einen ausgelagerten Teilbereich. Der Vorteil des Auslagerns ist die Wiederverwendbarkeit.
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

Layouts

Bei einem Layout handelt es sich um das Gerüst der Extension. Im Frontend beinhaltet das Layout meist nur einen umschließenden DIV-Container mit einer Extensionklasse.

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...)

6.2. 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.

6.2.1. Verfügbare ViewHelper

ViewHelper	Erklärung
<pre><f:count subject="{myarray}" /></pre>	Gibt die Anzahl der Objekte oder Inhalte aus einem Array wieder
<pre><f:if condition="{var} > 3"> // do something </f:if> <f:if condition="{var}"> <f:then>A</f:then> <f:else>B</f:else> </f:if></pre>	<p>Ein Bereich wird im Fluid nur geparsst, wenn die Condition zutrifft.</p> <p>Hinweis: Ein Stringvergleich ist nicht möglich – ein Workaround ist der Vergleich zweier Arrays</p>

```

<f:for each="{objects}"
as="object">

{object.title}

</f:for>

<f:for each="{objects}"
as="object" key="number"
iteration="itemIteration">

...

</f:for>

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

Foreach Schleife generiert eine neue Variable für den innenliegenden Bereich.

var_dump Debug auf String, Array oder Objekt

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

Beschneidet einen String auf einen bestimmten Wert. Hierbei werden vorhandene HTML-Tags nicht zerstört und bestehen weiterhin.

```
<f:format.currency  
currencySign="€"  
decimalSeparator=", "  
thousandsSeparator=".">"  
  
1234.56</f:format.currency>
```

Formatierte Ausgabe mit Währung
– Output 1.234,56 €

```
<f:format.number  
decimals="1"  
decimalSeparator=", "  
thousandsSeparator=".">"  
  
2345.678</f:format.number>
```

Formatierte Ausgabe einer Zahl –
Output 2.345,6

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

Formatiertes Datum

```
<f:image  
src="fileadmin/image.png"  
width="200 alt="My Image"  
/>
```

Ausgabe eines Bildes ähnlich der
Ausgabe mit TypoScript

```
<f:image  
src="{f:uri.resource  
(path: 'Images/Image.png') }  
" width="200 alt="My Image"  
/>
```

Unteres Beispiel mit Bild aus
Verzeichnis
Resources/Public/Images/Image.pn
g

```
<f:link.action  
action="edit">  
Bearbeiten</f:link.action>  
  
<f:uri.action action="edit"  
/>
```

Link auf Action
URL zu einer Action

```
<f:link.email email="foo@example.com" />
```

Link auf E-Mail

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

Externer Link

```
<f:link.page pageUid="23"> Interner Link
```

```
Contact</f:link.page>
```

```
<f:uri.page pageUid="23" />
```

URL zu einem internen Link

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

Label aus der Sprachdatei mit Berücksichtigung der aktuell eingestellten FE-Sprache (locallang.xlf oder locallang.xml)

Alle ViewHelper sind in der FluidWiki-Dokumentation von TYPO3 verfügbar:
<http://wiki.typo3.org/Fluid>

```
Outline:  
  
<f:translate key="idFromXml" />  
  
Inline:  
  
{f:translate(key:'idFromXml')}
```

ViewHelper mit Beginnendem und schließendem Tag

```
Outline:  
  
<f:format.date format="d.m.Y">{date}</f:format.date>  
  
Inline:  
  
{date -> f:format.date(format:'d.m.Y')}
```

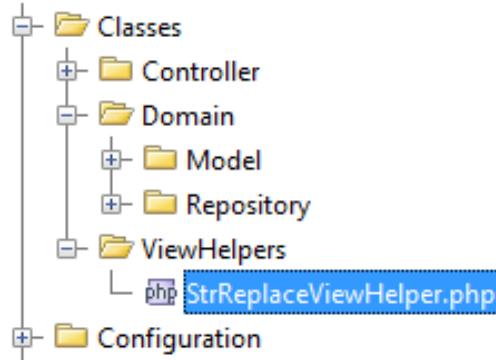
Beispiel für Verschachtelung

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

6.2.2. Inline- und Outline-Schreibweise

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

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



6.2.3. Eigene ViewHelper erstellen

In einigen Fällen benötigt man mehr Funktionen im View als es bereits ViewHelper von Fluid gibt. In so einem Fall kann man etwas PHP nutzen um einen eigenen ViewHelper zu erstellen.

In diesem Beispiel wollen wir innerhalb eines String ein Zeichen durch ein anderes ersetzen. In PHP einfach möglich mit str_replace().

Hierzu benötigen wir eine Datei mit dem Namen StrReplaceViewHelper.php im Verzeichnis Classes/ViewHelpers/

```
<?php
namespace TYPO3\Person\ViewHelpers;

/**
 * StrReplace Viewhelper
 *
 * @package TYPO3
 * @subpackage Fluid
 * @version
 */

class StrReplaceViewHelper extends \TYPO3\CMS\Fluid\Core\ViewHelper\AbstractViewHelper {

    /**
     * StrReplace
     *
     * @param string $value          Any String
     * @param string $search         String to be
     * @param string $replace        String to be
     * @return string $value
     */
    public function render($value, $search = ',', $replace = '.') {
        return str_replace($search, $replace, $value);
    }
}
?>
```

Der Inhalt der Datei kann z.B. so aussehen.

Die Methode render() wird in jedem ViewHelper erwartet. Der Klassenname ergibt sich aus dem Dateinamen.

```
{namespace vh=TYPO3\Person\ViewHelpers}
<f:layout name="Default" />

Kommentar

<f:section name="main">
    <vh:StrReplace value="a,b,c,d,e" />
    <vh:StrReplace value="a,b,c,d,e" search="," replace="." />
</f:section>
```

Wichtig ist beim Einsatz den neuen Namespace im Template zu deklarieren (siehe erste Zeile). Damit lassen sich ViewHelper auch extensionübergreifend verwenden.

Hinweis: Wenn die Parameter \$search und \$replace nicht explizit im Template angegeben werden, werden diese mit Defaultwerten befüllt.

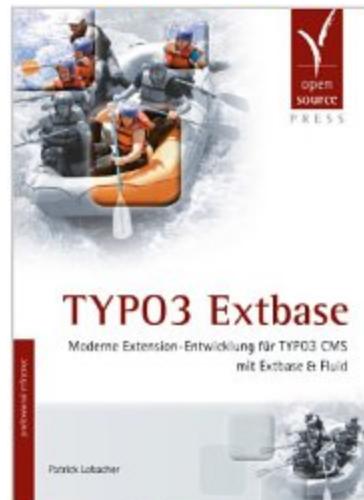
7. Links und Hilfe

7.1. Links

Titel	Links
Coding Guidelines	http://docs.typo3.org/typo3cms/CodingGuidelinesReference
TCA Reference	http://docs.typo3.org/typo3cms/TCAReference
Extbase und Fluid Cheatsheet	http://www.typovision.de/fileadmin/slides/ExtbaseFluidCheatSheetTypovision.pdf
Unittesting	http://docs.typo3.org/typo3cms/extensions/phpunit/
Fluid Wiki	http://wiki.typo3.org/Fluid

7.2. Literatur

Wir empfehlen das Buch TYPO3 Extbase von Patrick Lobacher (ISBN-10: 3955390705, ISBN-13: 978-3955390709)



7.3. Kontakt

in2code GmbH
Kunstmühlstraße 12a
83026 Rosenheim

0049 80 31 / 88 73 983
service@in2code.de