Der Einstieg in die neue Extension-Entwicklung

Zukunftssichere TYPO3-Extensions *mit* Extbase & Fluid

O'REILLY®

Jochen Rau & Sebastian Kurfürst

Inhalt

Einführung IX

1 Installation 1
Den Server einrichten 1
Die Entwicklungsumgebung einrichten 4
Weitere hilfreiche Extensions 8

2 Grundlagen 11
Objektorientierte Programmierung mit PHP 12
Domain-Driven Design 26
Model-View-Controller in Extbase 37
Test-Driven Development 42
Zusammenfassung 51

3 Reise durch das Blog-Beispiel 53
Erste Orientierung 54
Die Stationen der Reise 56
Die Extension aufrufen 57
Und Action! 59
Blogs aus dem Repository abholen 61
Ein Ausflug zur Datenbank 62
Pfade auf der Data-Map 62
Zurück im Controller 65
Die Ausgabe durch Fluid rendern 66
Das Ergebnis an TYPO3 zurückgeben 69
Alternative Reiseroute: Einen neuen Post anlegen 69
Automatische Speicherung der Domäne 74
Hinweise für Umsteiger 75

4 **Eine erste Extension anlegen** . . . 79
Die Beispiel-Extension . . . 79
Ordnerstruktur und Konfigurationsdateien anlegen . . . 80
Das Domänenmodell anlegen . . . 82
Produkte haltbar machen . . . 84
Den Ablauf steuern . . . 87
Das Template anlegen . . . 88
Das Plugin konfigurieren . . . 89

5 **Die Domäne modellieren** . . . 93
Die Anwendungsdomäne . . . 94
Das Domänenmodell implementieren . . . 98

6 **Die Persistenzschicht einrichten** . . . 117
Die Datenbank vorbereiten . . . 118
Eingabemasken des Backends konfigurieren . . . 129
Individuelle Abfragen implementieren . . . 148
Fremde Datenquellen nutzen . . . 158
Klassenhierarchien abbilden . . . 159

7 **Den Ablauf mit Controllern steuern** . . . 165
Controller und Actions anlegen . . . 166
Frontend-Plugins konfigurieren und einbinden . . . 180
Das Verhalten der Extension konfigurieren . . . 182

8 **Die Ausgabe mit Fluid gestalten** . . . 183
Basiskonzepte . . . 183
Verschiedene Ausgabeformate verwenden . . . 191
Wiederkehrende Snippets in Partials auslagern . . . 191
Die Darstellung mit Layouts vereinheitlichen . . . 193
TypoScript zur Ausgabe nutzen: der cObject-ViewHelper . . . 194
Zusätzliche Tag-Attribute mit additionalAttributes einfügen . . . 197
Boolesche Bedingungen zur Steuerung der Ausgabe verwenden . . . 198
Einen eigenen ViewHelper entwickeln . . . 200
PHP-basierte Views einsetzen . . . 207
Template-Erstellung am Beispiel . . . 209
Zusammenfassung . . . 215

9 Mehrsprachigkeit, Validierung und Sicherheit 217
Eine Extension lokalisieren und mehrsprachig auslegen 217
Domänenobjekte validieren 227
Sichere Extensions programmieren 241
Zusammenfassung 245

10 Ausblick 247
Eine Extension mit dem Kickstarter anlegen 247
Backend-Module 251
Migration auf FLOW3 und TYPO3 v5 253

A Coding Guidelines 255

B Referenz für Extbase 259

C Referenz für Fluid 275

Index 313

KAPITEL 4

Eine erste Extension anlegen

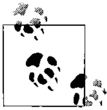
Dies ist ein Auszug aus dem Buch "Zukunftssichere TYPO3-Extensions mit Extbase und Fluid", ISBN 978-3-89721-965-6
http://www.oreilly.de/catalog/typo3extger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2010

In diesem Kapitel lernen Sie die Grundzüge einer auf Extbase und Fluid basierenden Extension kennen. Sie legen eine minimalistische Extension an, die auf die absolut notwendigen Strukturen beschränkt ist. Damit bleibt der Blick für das Ganze frei, ohne sich in den Details zu verlieren. In den folgenden Kapiteln wenden wir uns dann einem komplexeren Beispiel zu, um alle wesentlichen Merkmale von Extbase und Fluid erschöpfend zu behandeln.

Die Beispiel-Extension

Unsere erste Extension soll eine Liste eines Lagerbestands an Produkten ausgeben können, die wir zuvor im List-Modul des Backends angelegt haben. Jedes Produkt ist durch einen Titel, eine kurze Beschreibung sowie die Anzahl der am Lager befindlichen Stücke gekennzeichnet. Folgende Schritte sind für die Umsetzung der Extension notwendig:

1. Ordnerstruktur und die minimal notwendigen Konfigurationsdateien anlegen
2. Problemdomäne in ein abstrahiertes Domänenmodell (*Model*) übersetzen
3. Konfiguration der Persistenzschicht einrichten
    - Definition der Datenbank-Tabellen anlegen
    - Anzeige der Formulare für das Backend konfigurieren
    - Repositories für Produkt-Objekte anlegen
4. Ablauf innerhalb der Extension festlegen (*Controller* und *Action*-Methoden anlegen)
5. Design in HTML-Templates umsetzen
6. Plugin zur Anzeige der Listen konfigurieren
7. Die Extension installieren und testen



Wir haben die Reihenfolge der Schritte innerhalb der Beispiel-Extensions so gewählt, dass die Zusammenhänge sichtbar bleiben und sich ein »natürliches« Wachstum der Extension und Ihres Wissens ergibt. Nachdem Sie die ersten Erfahrungen in der Programmierung mit Extbase gesammelt haben, werden Sie diese Schritte wahrscheinlich in einer etwas anderen Reihenfolge und schneller durchlaufen wollen. Wir weisen an den entsprechenden Stellen darauf hin. Außerdem steht Ihnen in Zukunft mit dem Kickstarter, der in Kapitel 10 umrissen wird, ein komfortables Werkzeug zur Erzeugung der Grundstruktur einer Extension zur Verfügung.

Ordnerstruktur und Konfigurationsdateien anlegen

Bevor wir die erste Zeile Code schreiben, müssen wir die Infrastruktur der Extension einrichten. Dazu zählen neben der Ordnerhierarchie auch die minimal nötigen Konfigurationsdateien. Den eindeutigen Bezeichner unserer Extension (Extension-Key) legen wir auf `inventory`, und damit legen wir zugleich den Namen der Extension auf *Inventory* fest.

Der Name einer Extension wird immer in *UpperCamelCase* (beginnend mit einem Großbuchstaben, dann Groß- und Kleinbuchstaben; kein Unterstrich) geschrieben, während der Extension-Key nur Kleinbuchstaben und Unterstriche enthalten darf (lower_underscore). Eine Übersicht über die Namenskonventionen finden Sie in Anhang A, *Coding Guidelines*.

Extensions können in TYPO3 an verschiedenen Orten abgelegt werden. Lokal installierte Extensions sind der Regelfall. Diese befinden sich im Ordner *typo3conf/ext/*. Global installierte Extensions stehen allen Seiten zur Verfügung, die auf dieselbe Installation zurückgreifen. Sie werden in *typo3/ext/* abgelegt. System-Extensions werden mit der TYPO3-Distribution ausgeliefert und befinden sich im Ordner *typo3/sysext/*. Extbase oder Fluid sind Beispiele für System-Extensions. Alle drei Pfade befinden sich unterhalb des Installationsverzeichnisses von TYPO3, in dem auch die Datei *index.php* liegt.

Wir legen zunächst im Verzeichnis für lokale Extensions *typo3conf/ext/* den Ordner *inventory* an. Der Name dieses Ordners muss gleich dem Extension-Key und damit in Kleinbuchstaben und ggf. mit Unterstrichen geschrieben sein. Auf der obersten Ebene liegen die Ordner *Classes* und *Resources*. Der Ordner *Classes* enthält alle PHP-Klassen, mit Ausnahme von externen Bibliotheken, wie z.B. JavaScript-Bibliotheken. Der Ordner *Resources* enthält alle sonstigen Dateien, die noch von unserer Extension verarbeitet werden müssen (z.B. HTML-Templates) oder direkt an das Frontend ausgeliefert werden (z.B. Icons). Innerhalb des Ordners *Classes* befinden sich die Ordner *Controller* und *Domain*. Der Ordner *Controller* enthält in unserem Beispiel nur eine Klasse, die den gesamten Prozess der

Dies ist ein Auszug aus dem Buch "Zukunftssichere TYPO3-Extensions mit Extbase und Fluid", ISBN 978-3-89721-965-6
http://www.oreilly.de/catalog/typo3extger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2010

Listenerzeugung später steuern wird. Der Ordner *Domain* enthält wiederum die beiden Ordner *Model* und *Repository*. Daraus ergibt sich nun innerhalb des Extension-Ordners *inventory/* die Ordnerstruktur, die Sie in Abbildung 4-1 sehen.

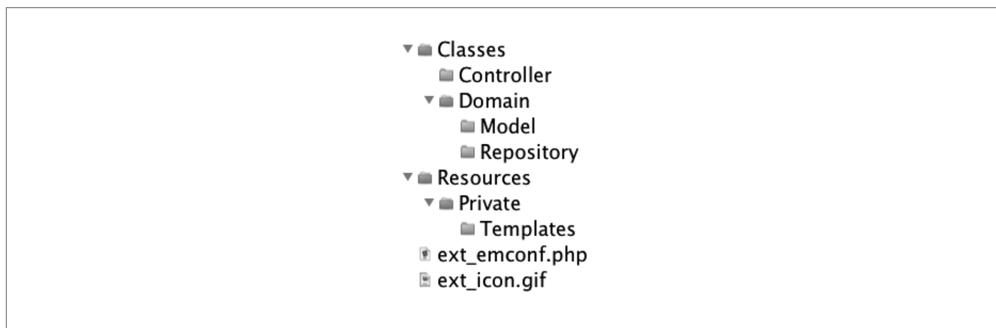


Abbildung 4-1: Die einfachste Ordnerstruktur mit den grundlegenden Dateien

Dies ist ein Auszug aus dem Buch "Zukunftssichere TYPO3-Extensions mit Extbase und Fluid", ISBN 978-3-89721-965-6
http://www.oreilly.de/catalog/typo3extger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2010

Damit die Extension von TYPO3 geladen werden kann, benötigen wir zwei Konfigurationsdateien. Diese werden in den Extension-Ordner *inventory/* auf der obersten Ebene abgelegt. Sie können diese Dateien von einer existierenden Extension kopieren und anpassen. Später werden Sie diese durch den Kickstarter anlegen lassen.

Die Datei *ext_emconf.php* enthält die Meta-Informationen zur Extension, wie z.B. den Titel, eine Beschreibung, den Status, den Namen des Autors etc. Sie unterscheidet sich nicht von herkömmlichen Extensions. Allerdings empfiehlt es sich, die Abhängigkeit zu Extbase und Fluid (ggf. auch zu einer bestimmten Version) anzugeben.

```
<?php
$EM_CONF[$_EXTKEY] = array(
   'title' => 'Inventory List',
   'description' => 'An extension to manage a stock.',
   'category' => 'plugin',
   'author' => 'Jochen Rau',
   'author_company' => '',
   'author_email' => '',
   'dependencies' => 'extbase,fluid',
   'state' => 'alpha',
   'clearCacheOnLoad' => '1',
   'version' => '0.0.0',
   'constraints' => array(
        'depends' => array(
              'typo3' => '4.3.0-4.3.99',
         'extbase' => '1.0.0-0.0.0',
         'fluid' => '1.0.0-0.0.0',
        )
   )
);
?>
```

Die Datei *ext_icon.gif* enthält das Icon der Extension. Hierfür können Sie jede im GIF-Format gespeicherte Grafik verwenden. Sie sollte eine Breite von 18 Pixel und eine Höhe von 16 Pixel nicht überschreiten. Das Icon erscheint im Extension Manager und im Extension-Repository (TER).

Nachdem die grundlegende Struktur aufgebaut wurde, kann die Extension im Extension-Manager bereits angezeigt und installiert werden. Wir wenden uns aber zunächst unserer Domäne zu.

Das Domänenmodell anlegen

Die Domäne unserer ersten Extension ist sehr schlicht. Der wesentliche Begriff unserer Domäne ist das »Produkt«. Alle für uns wichtigen Eigenschaften eines Produkts und dessen »Verhalten« werden in einer Klasse mit dem Namen `Tx_Inventory_Domain_Model_Product` definiert. Der Code dieser Klasse wird in einer Datei mit dem Namen *Product.php* abgelegt. Der Name der Datei ergibt sich durch Anhängen von *.php* an den letzten, durch Unterstrich abgetrennten Teil des Klassennamens. Diese Klassendatei wird im Ordner *EXT:inventory/Classes/Domain/Model/* abgelegt.

Die Bezeichnungen der Klassen müssen in jedem Fall die Ordnerstruktur widerspiegeln. Extbase erwartet z.B. die Klasse `Tx_MyExtension_ErsterOrdner_ZweiterOrder_File` im Ordner *my_extension/Classes/ErsterOrdner/ZweiterOrdner/File.php*. Achten Sie auch auf die entsprechende Großschreibung der Ordnernamen.

Werfen wir nun einen Blick in diese Datei. Beachten Sie, dass die Klasse `Tx_Inventory_Domain_Model_Product` von der Extbase-Klasse `Tx_Extbase_DomainObject_AbstractEntity` abgeleitet werden muss.

```
<?php
class Tx_Inventory_Domain_Model_Product extends Tx_Extbase_DomainObject_AbstractEntity {

    /**
     * @var string
     **/
    protected $name = '';

    /**
     * @var string
     **/
    protected $description = '';

    /**
     * @var int
     **/
    protected $quantity = 0;
```


Dies ist ein Auszug aus dem Buch "Zukunftssichere TYPO3-Extensions mit Extbase und Fluid", ISBN 978-3-89721-965-6
http://www.oreilly.de/catalog/typo3extger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2010

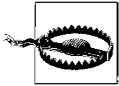
```
    public function __construct($name = '', $description = '', $quantity = 0) {
        $this->setName($name);
        $this->setDescription($description);
        $this->setQuantity($quantity);
    }

    public function setName($name) {
        $this->name = (string)$name;
    }

    public function getName() {
        return $this->name;
    }

    public function setDescription($description) {
        $this->description = (string)$description;
    }

    public function getDescription() {
        return $this->description;
    }

    public function setQuantity($quantity) {
        $this->quantity = (int)$quantity;
    }

    public function getQuantity() {
        return $this->quantity;
    }

}
?>
```

Die Eigenschaften (*Properties*) sind als Klassenvariablen `$name`, `$description` und `$quantity` angelegt und durch das Schlüsselwort `protected` vor direkten Zugriffen von außen geschützt (*gekapselt*). Die Eigenschaftswerte können nur über die als `public` deklarierten Methoden `set`*`Property`*`()` und `get`*`Property`*`()` gesetzt bzw. ausgelesen werden. Methoden in dieser Form werden sehr häufig verwendet und daher kurz *Getter* und *Setter* genannt.

Auf den ersten Blick erscheinen Methoden für den Zugriff auf Klassenvariablen umständlich zu sein. Sie haben aber mehrere Vorteile: Die Interna der Verarbeitung können zu einem späteren Zeitpunkt hinzugefügt oder geändert werden, ohne dass dazu Änderungen am aufrufenden Objekt vorgenommen werden müssten. Es kann auch z.B. das Auslesen erlaubt werden, ohne dass man gleichzeitig schreibenden Zugriff erlaubt. Die etwas lästige Arbeit, diese Methoden zu schreiben, nimmt Ihnen später der Kickstarter ab. Außerdem bieten die meisten Entwicklungsumgebungen Makros oder Snippets für diesen Zweck an. Beachten Sie auch, dass Extbase zu verschiedenen Zeitpunkten intern versucht, eine Eigenschaft `$name` über eine Methode `setName()` zu befüllen.

Dies ist ein Auszug aus dem Buch "Zukunftssichere TYPO3-Extensions mit Extbase und Fluid", ISBN 978-3-89721-965-6
http://www.oreilly.de/catalog/typo3extger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2010

Die Methode `__construct()` dient dazu, einen wohldefinierten Zustand am Anfang des Lebenszyklus des Objekts sicherzustellen. Hier werden die Eigenschaften des Produkts gesetzt bzw. mit Werten vorbelegt.



In der Deklaration des Konstruktors wird das Argument `$name` mit einem Standardwert (leerer String) vorbelegt und damit optional. Das ist notwendig, damit Extbase die Klasse »leer« instanziieren kann, ohne einen Namen übergeben zu müssen. Damit verstößt Extbase gegen die reine Lehre, da der Konstruktor eigentlich die Minimalkonfiguration des Objekts *Organisation* sicherstellen soll. Bei Extbase wird dies aber besser über sogenannte Validatoren erledigt (siehe dazu den Abschnitt »Domänenobjekte validieren« in Kapitel 9).

Dies ist ein Auszug aus dem Buch "Zukunftssichere TYPO3-Extensions mit Extbase und Fluid", ISBN 978-3-89721-965-6
http://www.oreilly.de/catalog/typo3extger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2010

Produkte haltbar machen

Von der Klasse `Tx_Inventory_Domain_Model_Product` können wir jetzt bereits Instanzen – also konkrete Produkte mit individuellen Eigenschaften – zur Laufzeit des Skriptes erstellen. Diese sind aber nur in flüchtiger Form im Arbeitsspeicher vorhanden und werden, nachdem die Seite komplett von TYPO3 erzeugt wurde, von PHP wieder gelöscht. Damit die Produkte über eine längere Zeit zur Verfügung stehen, müssen wir sie »haltbar« machen. Üblicherweise geschieht dies, indem man sie in eine Datenbank ablegt. Daher legen wir zunächst die dafür notwendige Datenbanktabelle an.

Das Anlegen der Datenbanktabellen können Sie vom Kickstarter übernehmen lassen. In der TYPO3 v5 werden diese Schritte gänzlich entfallen.

TYPO3 erledigt dies für uns, wenn wir den entsprechenden SQL-Aufruf in der Datei *EXT:inventory/ext_tables.sql* eintragen:

```
CREATE TABLE tx_inventory_domain_model_product (
    uid int(11) unsigned DEFAULT '0' NOT NULL auto_increment,
    pid int(11) DEFAULT '0' NOT NULL,

    name varchar(255) DEFAULT '' NOT NULL,
    description text NOT NULL,
    quantity int(11) DEFAULT '0' NOT NULL,

    PRIMARY KEY (uid),
    KEY parent (pid),
);
```

Dieser SQL-Befehl legt eine neue Tabelle mit den entsprechenden Spalten an. Die Spalten `uid` und `pid` dienen zur internen Verwaltung. Unsere Produkteigenschaften tauchen als Spalten `name`, `description` und `quantity` wieder auf.

Auf die Einträge in der Datenbank kann dann über das Backend von TYPO3 zugegriffen werden. Die Formulare des Backends werden anhand einer Konfiguration erzeugt, die in einem PHP-Array abgelegt ist, dem sogenannten *Table-Configuration-Array* (kurz *TCA*).

Innerhalb der Extension wird dann über Repositories transparent auf diese Daten zugegriffen. »Transparent« bedeutet, dass man sich beim Zugriff auf Repositories um die *Art* der Speicherung der Daten keine Gedanken machen muss.

Damit das Backend nun weiß, wie es die Produktdaten in einem Formular anzeigen soll, müssen wir dies für die Tabelle in der Datei *EXT:inventory/ext_tables.php* konfigurieren. Dort wird im Array `$TCA` unter dem Tabellennamen als Schlüssel die Konfiguration abgelegt. Diese umfasst mehrere Sektionen. In der Sektion `ctrl` befinden sich grundlegende Eigenschaften, wie der Tabellenname oder die Angabe, welcher Tabellenspalte das Label für die Einträge entnommen werden soll. In der Sektion `columns` wird für jede Tabellenspalte beschrieben, wie diese im Backend angezeigt werden soll. Die Sektion `types` definiert, in welcher Reihenfolge die Tabellenspalten angezeigt werden und wie diese ggf. gruppiert werden.

Die Möglichkeiten, mit dem `TCA` die Ausgabe im Backend zu beeinflussen, sind immens. Im Rahmen diese Buches können wir diese nur anreißen. Eine vollständige Auflistung aller Optionen finden Sie online unter *http://typo3.org/documentation/document-library/core-documentation/doc_core_api/4.3.0/view/4/1/*. In umfangreicheren Extensions würde man die Sektionen `columns` und `types` aus Performance-Gründen auch in eine eigene Datei *tca.php* auslagern. In unserem Beispiel soll diese Minimalkonfiguration aber genügen.

```
<?php
if (!defined ('TYPO3_MODE')) die ('Access denied.');

$TCA['tx_inventory_domain_model_product'] = array (
    'ctrl' => array (
        'title' => 'Lagerbestände',
        'label' => 'name',
    ),
    'columns' => array(
        'name' => array(
            'label' => 'Produktbezeichnung',
            'config' => array(
                'type' => 'input',
                'size' => '20',
                'eval' => 'trim,required'
            )
        ),
        'description' => array(
            'label' => 'Produktbeschreibung',
            'config' => array(
                'type' => 'text',
                'eval' => 'trim'
            )
        ),
        'quantity' => array(
            'label' => 'Lagerbestand',
            'config' => array(
```

Dies ist ein Auszug aus dem Buch "Zukunftssichere TYPO3-Extensions mit Extbase und Fluid", ISBN 978-3-89721-965-6
http://www.oreilly.de/catalog/typo3extger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2010

```
                    'type' => 'input',
                    'size' => '4',
                    'eval'=> 'int'
                )
            ),
        ),
        'types' => array(
            '0' => array('showitem' => 'name, description, quantity')
        )
    );
?>
```

Nachdem wir die Extension installiert haben, können wir im Backend unsere ersten Produkte anlegen. Wie in Abbildung 4-2 gezeigt, erzeugen wir dazu einen Systemordner, der die Produkte aufnehmen wird ❶. In diesem legen wir einige wenige neue Bestandsdaten an ❷.

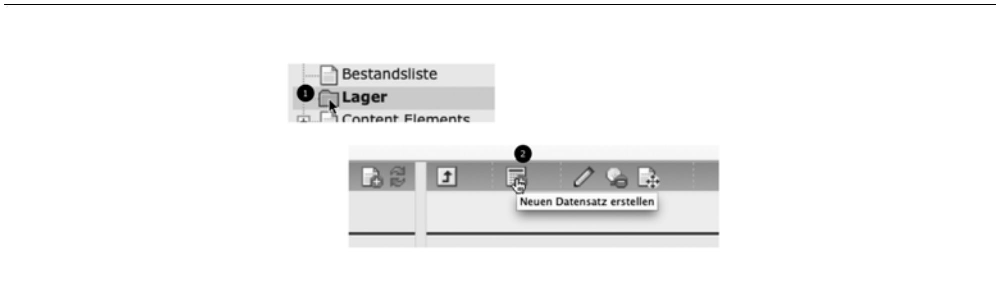


Abbildung 4-2: Anlegen eines neuen Produkts

Wir haben in diesem Abschnitt ein Abbild (oder ein Modell) der Realität geschaffen, indem wir nur einen Ausschnitt an Eigenschaften der realen Produkte in Software übersetzt haben, die in unserer Domäne eine Rolle spielen. Dieses von der realen Welt abstrahierte Modell ist damit vollständig angelegt.

Um auf die im Backend angelegten Objekte zugreifen zu können, legen wir ein Repository für Produkte an. Das `Tx_Inventory_Domain_Repository_ProductRepository` ist ein Objekt, in dem die Produkte »abgelegt« sind. Wir können ein Repository auffordern, alle (oder bestimmte) Produkte zu finden und an uns zu übergeben. Die Repository-Klasse ist in unserem Fall sehr kurz:

```
<?php
class Tx_Inventory_Domain_Repository_ProductRepository
      extends Tx_Extbase_Persistence_Repository {}
?>
```

Unser `ProductRepository` muss von `Tx_Extbase_Persistence_Repository` abgeleitet werden und erbt von diesem alle Methoden. Es kann daher in unserem einfachen Beispiel leer bleiben. Die Klassendatei *ProductRepository.php* legen wir in den Ordner *EXT:inventory/Classes/Domain/Repository/* ab.

Dies ist ein Auszug aus dem Buch "Zukunftssichere TYPO3-Extensions mit Extbase und Fluid", ISBN 978-3-89721-965-6
http://www.oreilly.de/catalog/typo3extger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2010

Den Ablauf steuern

Der im Backend angelegte Lagerbestand soll nun im Frontend als Liste ausgegeben werden. Die Erzeugung des HTML-Codes aus den anzuzeigenden Produkt-Objekten übernimmt der sogenannte *View*. Extbase verwendet als View standardmäßig die Klasse `Tx_Fluid_View_TemplateView` der Extension Fluid.

Das Bindeglied zwischen dem *Model* und dem *View* ist der *Controller*. Er steuert die Abläufe innerhalb der Extension und ist in unserem Fall zuständig für die `list`-Aktion. Diese umfasst das Auffinden der Produkte, die angezeigt werden sollen, sowie die Weitergabe dieser ausgewählten Produkte an den zuständigen View.

Der Klassenname des Controllers muss auf `Controller` enden. Da unser Controller die Ausgabe des Lagerbestands steuert, nennen wir ihn `Tx_Inventory_Controller_InventoryController`.

Bei der Namensgebung des Controllers sind Sie im oben beschriebenen Rahmen frei. Wir empfehlen jedoch, einen Controller danach zu benennen, was er »kontrolliert«. Bei größeren Projekten sind dies insbesondere die Aggregate-Root-Objekte (siehe Abschnitt »Aggregates« in Kapitel 2). Wir hätten unseren Controller daher auch `Tx_Inventory_Controller_ProductController` nennen können.

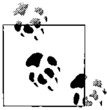
In unserem einfachen Beispiel sieht der Controller wie folgt aus:

```
<?php
class Tx_Inventory_Controller_InventoryController
      extends Tx_Extbase_MVC_Controller_ActionController {

     public function listAction() {
          $productRepository = t3lib_div::makeInstance('Tx_Inventory_Domain_Repository_
               ProductRepository');
          $products = $productRepository->findAll();
          $this->view->assign('products', $products);
     }

}
?>
```

Unser `Tx_Inventory_Controller_InventoryController` muss vom `Tx_Extbase_MVC_Controller_ ActionController` abgeleitet werden. Er enthält als einzige Methode die `listAction()`. Extbase erkennt alle Methoden, deren Name auf `Action` endet, als Aktionen – also als kleine Ablaufpläne.

In der ersten Zeile der `listAction()` wird das `ProductRepository` instanziiert. Die anzuzeigenden Produkte erhalten wir durch `findAll()` des Repositories. Diese Methode ist in der Klasse `Tx_Extbase_Persistence_Repository` implementiert. Welche Methoden Ihnen noch zur Verfügung stehen, können Sie in Kapitel 6 nachschlagen.

Dies ist ein Auszug aus dem Buch "Zukunftssichere TYPO3-Extensions mit Extbase und Fluid", ISBN 978-3-89721-965-6
http://www.oreilly.de/catalog/typo3extger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2010



Achten Sie darauf, dass Sie zum Erzeugen einer neuen Instanz des Repositories die Framework-Methode `t3lib_div::makeInstance()` von TYPO3 aufrufen und nicht das Schlüsselwort `new` verwenden. Hintergrund für Ortskundige: Das Repository ist ein sogenanntes *Singleton* und ist als solches entsprechend gekennzeichnet. Die Methode `makeInstance()` erkennt das Singleton anhand dieser Kennzeichnung und liefert – nach der erstmaligen Erstellung – immer dasselbe Objekt zurück, unabhängig davon, an welcher Stelle in Ihrem Code Sie es anfordern. Die Erzeugung mit »new« liefert dagegen immer ein neues und damit leeres Repository-Objekt.

Als Ergebnis erhalten wir ein PHP-Array mit den Produkt-Objekten. Diese geben wir abschließend durch `$this->view->assign(...)` an den View weiter. Ohne unser weiteres Zutun wird am Ende der Aktion der View aufgefordert, den an ihn übergebenen Inhalt auf Basis eines HTML-Templates zu rendern und an TYPO3 zurückzugeben.

```
return $this->view->render();
```

Diese Zeile nimmt uns aber Extbase ab, falls wir nicht zuvor selbst den Rendering-Prozess anstoßen. Sie kann in unserem Fall also weggelassen werden.

Das Template anlegen

In Extbase werden Templates für das Frontend – falls es nicht anders konfiguriert wird – in einem Unterordner des Ordners *EXT:inventory/Resources/Private/Templates/* angelegt. Der Name des Unterordners ergibt sich aus dem letzten Abschnitt des Klassennamens des Controllers, indem das Suffix `Controller` weggelassen wird. Aus dem Klassennamen `Tx_Inventory_Controller_InventoryController` wird also der Ordnername *Inventory*.

Innerhalb des Ordners *Inventory* legen wir die Datei mit dem HTML-Template an. Der Name der Datei ergibt sich aus dem Namen der aufrufenden Action, ergänzt um das Suffix *.html*. In unserem Fall lautet der Dateiname also *list.html*.

Beachten Sie, dass die Datei *list.html* und nicht *listAction.html* heißt. *list* ist der Name der Action. `listAction()` ist der Name der zugehörigen Methode im Controller. Ohne weitere Angabe erwartet Extbase die Endung *.html*. Es können jedoch auch Templates für andere Formate, wie z.B. JSON oder XML, hinterlegt werden. Wie Sie diese ansprechen, können Sie in Kapitel 8, Abschnitt »Verschiedene Ausgabeformate verwenden« nachschlagen.

Das HTML-Template in der Datei *EXT:inventory/Resources/Private/Templates/Inventory/list.html* sieht wie folgt aus:

```
<table border="1" cellspacing="1" cellpadding="5">
    <tr>
        <th>Produktbezeichnung</th>
```

Dies ist ein Auszug aus dem Buch "Zukunftssichere TYPO3-Extensions mit Extbase und Fluid", ISBN 978-3-89721-965-6
http://www.oreilly.de/catalog/typo3extger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2010

```
            <th>Produktbeschreibung</th>
            <th>Anzahl</th>
        </tr>
        <f:for each="{products}" as="product">
        <tr>
            <td valign="top">{product.name}</td>
            <td valign="top"><f:format.crop maxCharacters="100">{product.description}
                </f:format.crop>
            </td>
            <td valign="top">{product.quantity}</td>
        </tr>
        </f:for>
    </table>
```

Wir geben den Lagerbestand als Tabelle aus. Auf das Array mit den Produkt-Objekten, die wir im Controller durch `$this->view->assign('products', $products)` dem View übergeben haben, können wir nun durch `{products}` zugreifen. Die mit `<f:` beginnenden Tags sind Fluid-Tags. Der Code innerhalb des `for`-Tags wird für jedes Produkt-Objekt in `products` wiederholt. Das `crop`-Tag kürzt den enthaltenen Text auf die angegebene Länge. Eine ausführliche Einführung in die Verwendung von Fluid-Tags finden Sie in Kapitel 8, *Die Ausgabe mit Fluid gestalten*, und in der Referenz in Anhang C.

Noch können wir das Ergebnis nicht im Frontend aufrufen. Dazu müssen wir zuerst ein Plugin definieren.

Das Plugin konfigurieren

Eine Extension stellt für die Ausgabe ihrer Daten üblicherweise ein sogenanntes *Plugin* zur Verfügung. Ein Plugin ist ein Inhaltselement, das wie ein Textelement oder ein Bild auf einer Seite platziert werden kann. Es ist eine »virtuelle« Zusammenstellung von einer oder mehreren Actions. Diese Actions können durchaus in verschiedenen Controllern liegen. In unserem Beispiel gibt es nur eine Controller-Action-Kombination, nämlich `Inventory->list`. Diese Kombination wird in der Datei *ext_localconf.php* eingetragen, die wir auf der obersten Ebene des Extension-Ordners anlegen.

```
<?php
if (!defined ('TYPO3_MODE')) die ('Access denied.');

Tx_Extbase_Utility_Extension::configurePlugin(
    $_EXTKEY,
    'List',
    array('Inventory' => 'list')
);
?>
```

Dies ist ein Auszug aus dem Buch "Zukunftssichere TYPO3-Extensions mit Extbase und Fluid", ISBN 978-3-89721-965-6
http://www.oreilly.de/catalog/typo3extger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2010

Mit der ersten Zeile wird – wie auch in der Datei *ext_tables.php* – aus Sicherheitsgründen verhindert, dass der PHP-Code außerhalb von TYPO3 direkt ausgeführt werden kann. Die statische Methode `configurePlugin()` der Klasse besitzt mehrere Argumente. Mit dem ersten übergeben wir den Extension-Key, der in der globalen Variablen `$_EXTKEY` bereits zur Verfügung steht (ergibt sich aus dem Namen des Extension-Ordners). Mit dem zweiten Argument geben wir dem Plugin einen eindeutigen Namen (in UpperCamelCase-Schreibweise). Dieser dient später dazu, das Plugin unter mehreren Plugins auf der Seite eindeutig zu identifizieren. Das dritte Argument ist ein Array mit allen Controller-Action-Kombinationen, die das Plugin ausführen darf. Der Array-Key ist dabei der Name des Controllers (ohne das Suffix `Controller`), und das Array-Value ist eine kommaseparierte Liste aller durch das Plugin ausführbaren Actions des Controllers. In unserem Fall ist dies die `list`-Action (wiederum ohne das Suffix `Action`). Das Array `array('Inventory' -> 'list')` erlaubt also, über das Plugin die Methode `listAction()` im `Tx_Inventory_Controller_InventoryController` auszuführen.

Standardmäßig wird das Ergebnis aller Actions im Cache abgelegt. Falls dies für einzelne Actions nicht erwünscht ist, kann man dies mit einem vierten, optionalen Argument angeben. Es ist ein Array, das gleich aufgebaut ist wie das vorherige. Nur werden jetzt alle Actions aufgelistet, deren Ergebnis *nicht* im Cache abgelegt werden soll.

Technisch wird dies dadurch gelöst, dass im automatisch generierten TypoScript-Code eine Abfrage (*Condition*) hinzugefügt wird, die je nach Bedarf Extbase entweder als Content-Objekt vom Typ USER (gecacht) oder vom Typ USER_INT (ungecacht) aufruft. Falls Sie also auf der Suche nach Problemen mit dem Caching sein sollten, lohnt ein Blick in dieses generierte TypoScript.

Nun folgt die Registrierung des Plugins, so dass es in der Auswahlbox des Content-Elements *Plugin* auftaucht. Dazu fügen wir die folgende Zeile in die bereits bestehende Datei *ext_tables.php* ein:

```
Tx_Extbase_Utility_Extension::registerPlugin(
    $_EXTKEY,
    'List',
    'The Inventory List'
    );
```

Der erste Parameter ist wie bei der Methode `configurePlugin()` wieder der Extension-Key und der zweite der Name des Plugins. Das dritte Argument ist ein beliebiger, nicht zu langer Titel des Plugins für die Auswahlbox des Content-Elements. Nach der Installation der Extension können wir das Plugin auf einer Seite einfügen. Vergessen Sie nicht, den Systemordner, der die Produkte enthält, als Ausgangspunkt (in unserem Beispiel »Lager«) im Plugin einzutragen. Ihre Produkte werden ansonsten nicht gefunden (siehe Abbildung 4-3).

Dies ist ein Auszug aus dem Buch "Zukunftssichere TYPO3-Extensions mit Extbase und Fluid", ISBN 978-3-89721-965-6
http://www.oreilly.de/catalog/typo3extger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2010

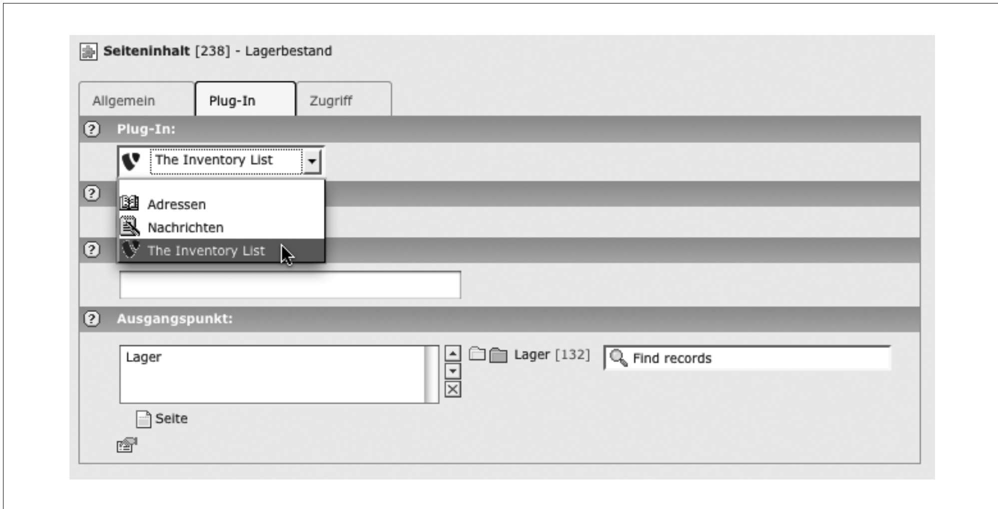


Abbildung 4-3: Unser Plugin erscheint in der Auswahlbox des Content-Elements.

Der nächste Aufruf der Seite, auf der das Plugin liegt, zeigt den Lagerbestand als Tabelle (Abbildung 4-4).

Lagerbestand

Produktbezeichnung	Produktbeschreibung	Anzahl
Gartenzwerg	Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore...	212
Rasenmäher	Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla...	5

Abbildung 4-4: Die Ausgabe des Lagerbestands im Frontend

Damit ist die erste kleine Extbase-Extension fertiggestellt. Das Beispiel war bewusst einfach gehalten. Es verdeutlicht damit die wichtigsten Arbeitsschritte und einzuhaltenden Konventionen. Zu einer ausgewachsenen Extension fehlen uns aber noch einige Zutaten:

- Reale Domänenmodelle weisen eine hohe Komplexität auf. (Produkte haben z.B. verschiedene Preise und sind Produktkategorien zugeordnet.)
- Mehrere unterschiedliche Ansichten müssen generiert werden (Einzelansicht, Listenansicht mit Suche etc.).

Dies ist ein Auszug aus dem Buch "Zukunftssichere TYPO3-Extensions mit Extbase und Fluid", ISBN 978-3-89721-965-6
http://www.oreilly.de/catalog/typo3extger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2010

- Der Webseitenbenutzer soll auf verschiedene Arten mit den Daten interagieren können (bearbeiten, neu anlegen, sortieren etc.).
- Eingaben des Webseitenbenutzers müssen auf Konsistenz geprüft (validiert) werden.

Die Beispiel-Extension, die wir Ihnen ab Kapitel 5 vorstellen, ist daher wesentlich facettenreicher.

Dies ist ein Auszug aus dem Buch "Zukunftssichere TYPO3-Extensions mit Extbase und Fluid", ISBN 978-3-89721-965-6
http://www.oreilly.de/catalog/typo3extger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2010