SpeedPartner

Extbase/Fluid

Tipps, Tricks und Stolperfallen

(eine Einführung)

Inhalte

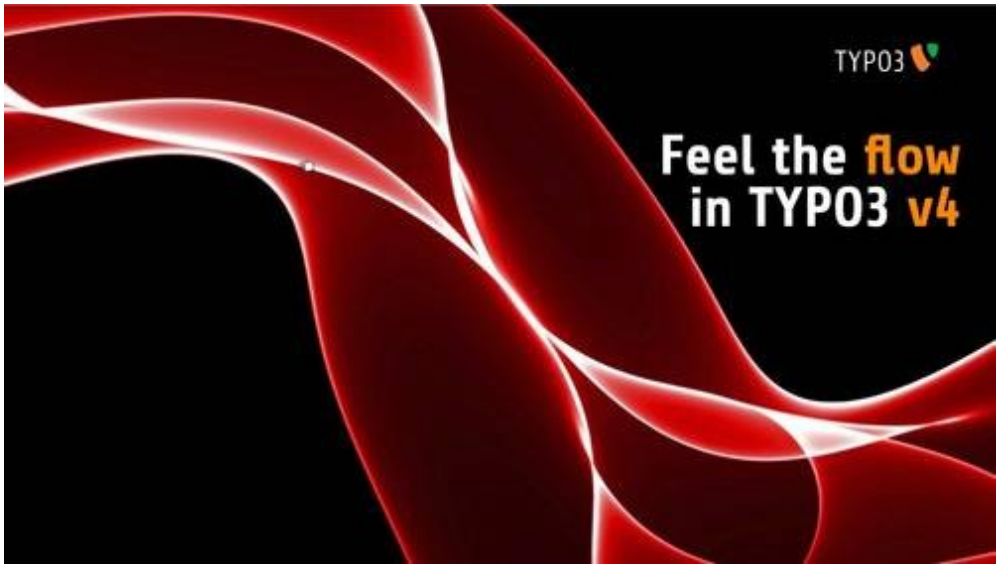
- Einführung/Überblick
- Fluid ohne Extbase
- Extbase
  - Konventionen (Naming, phpdoc, ...)
- Vom Pfad abweichen
- Eigene Erweiterungen
- Stolperfallen
- Links / Hilfen

Über mich

- Stefan Neufeind
- Aus Neuss
- Tätig für SpeedPartner GmbH, ein Internet-Service-Provider (ISP)
  (Consulting, Entwicklung, Administration)
  - Individuelle TYPO3-Entwicklungen
  - Hosting, Housing, Managed Services
  - Domains / Domain-Services
  - xDSL, IP-Upstream, IPv6
- Aktive Mitarbeit im Community-Umfeld (PHP/PEAR, TYPO3, Linux)
- Autor für z.B. t3n, iX, Internet World, ...

Ziele des Vortrags:

- Einstieg geben
  - Grober Überblick
  - Aufbauend auf Standard-Beispiel „blog_example“
- Wichtigsten Konventionen erläutern
- Standard-Fehler vermeiden
- Berührungsängste nehmen
- Spaß an Arbeit mit Extbase/Fluid vermitteln
- Vorbereitet sein für Umstieg auf FLOW3



Hintergrund zur Entwicklung:

- „Klassische“ TYPO3-Architektur weit verbreitet
  - Gemeinsame Basis für Extensions („tslib_pibase“)
  - Teilweise sehr unterschiedliche Ansätze für Aufbau von Extensions
  - Templating in Extensions überwiegend Marker-basiert
    - Zusammengesetzt aus Platzhaltern (Markern) und Blöcken (Subparts)

- In 2006 Entscheidung TYPO3 „neu zu schreiben“
  - Aufteilung in Framework (FLOW3) und das eigentliche CMS (TYPO3 v5)
  - Moderne Architektur und Software-Entwicklungs-Paradigmen
  - Nutzung aktueller Möglichkeiten (PHP 5.2+)
  - Domain-driven design, MVC (Model-View-Controller)

Zusammenhang und Zusammenarbeit TYPO3v4 und TYPO3v5:

- Okt. 2008: „Transition Days“ in Berlin
  - Verabschiedung des „The Berlin Manifestio“
    - Beschluss TYPO3 v4 weiterzuentwickeln (auch nach TYPO3 v5-Release)
    - Übergang TYPO3 v4 zu TYPO3 v5 ermöglichen/vereinfachen
  - War für TYPO3 v4 Grundlage zur Entscheidung für
    - Eine moderne, neue Basisklasse als Ersatz für tslib_pibase: Extbase
    - Neue, flexible und erweiterbare Template-Engine: Fluid

Neuer Ansatz für TYPO3 v4-Extensions:

- Extbase-Framework:
  - Konzepte und Lösungen von FLOW3 für TYPO3 v4.3+ verfügbar machen
  - Struktur der Extensions, Konventionen, Schnittstellen (APIs) weitgehend gleich
  - Leichten Übergang zu FLOW3/TYPO3v5 ermöglichen
    (Anpassungen erforderlich, jedoch Architektur und Schnittstellen ähnlich)

- Fluid für FLOW3 Basis ist Basis von Fluid für TYPO3v4
  - Fluid-Team entwickelte in wenigen Tagen ein „Backporter“-Tool
  - Ermöglicht Fluid-Code von TYPO3v4 und FLOW3 synchron zu halten

- Beta von Extbase und Fluid im Herbst 2009
- Aufnahme in TYPO3 Core mit Veröffentlichung von TYPO3 4.3.0 (Nov. 2009)

Wechseln von Markern zu Fluid?

- Marker-Konzept: einfach und „bekannt“
  - Jedoch fast zu „minimalistisch“ und wenig
  - Umfangreiche Templates erfordern teils sehr kreative Lösungen (Schleifen, ...)
  - Unterschiedliche Lösungsansätze mit Stärken/Schwächen
  - Umsetzungen in manchen Fällen fehleranfällig

- Fluid: strukturiert, flexibel, erweiterbar
  - Übersichtlicher Aufbau
  - Kontrollstrukturen: Verzweigungen / Schleifen
  - Objektorientierung
  - Erweiterbarkeit (ViewHelper, ...)

Einfaches Beispiel mit Markern:

- HTML-Basis / Ziel:

```
<ul>
  <li style=“background:red“>Name1</li>
  <li style=“background:green“>Name2</li>
  <li style=“background:red“>Name3</li>
</ul>
```

- Im Template:

```
<!-- ###NAMELIST### begin -->
<ul>
  <!-- ###ROW_ODD### begin -->
  <li style="background:red">###NAME###</li>
  <!-- ###ROW_ODD### begin -->
  <!-- ###ROW_EVEN### begin -->
  <li style="background:green">###NAME###</li>
  <!-- ###ROW_EVEN### begin -->
</ul>
<!-- ###NAMELIST### end -->
```

Einfaches Beispiel mit Markern:

- Extension-Code (Auszug):

```
$templateCode = $this->cObj->fileResource('EXT:' . $this->extKey . '/res/marker.htm');

$template = array();
$template['namelist']   = $this->cObj->getSubpart($templateCode,'###NAMELIST###');
$template['row_odd']    = $this->cObj->getSubpart($template['namelist'],'###ROW_ODD###');
$template['row_even']   = $this->cObj->getSubpart($template['namelist'],'###ROW_EVEN###');

$contentList = '';
$odd = true;
foreach($names as $name) {
 if($odd) {
  $contentList .= $this->cObj->substituteMarker($template['row_odd'], '###NAME###', $name);
 } else {
  $contentList .= $this->cObj->substituteMarker($template['row_even'],'###NAME###',$name);
 }
 $odd = ($odd == false);
}

$content = $template['namelist'];
$content = $this->cObj->substituteSubpart($content, '###ROW_ODD###', '');
$content = $this->cObj->substituteSubpart($content, '###ROW_EVEN###', $contentList);
```

Einfaches Beispiel mit Fluid:

- HTML-Basis / Ziel:

```
<ul>
  <li style=“background:red“>Name1</li>
  <li style=“background:green“>Name2</li>
  <li style=“background:red“>Name3</li>
</ul>
```

- Im Template:

```
<ul>
<f:for each="{names}" as="name">
<f:cycle values="{0: 'red', 1: 'green'}" as="alternate">
<li style="background:{alternate}">{name}</li>
</f:cycle>
</f:for>
</ul>
```

Einfaches Beispiel mit Fluid:

- Extension-Code (Auszug):

```
$view = t3lib_div::makeInstance('Tx_Fluid_View_TemplateView');

$controllerContext =
  t3lib_div::makeInstance('Tx_Extbase_MVC_Controller_ControllerContext');
$controllerContext->setRequest(t3lib_div::makeInstance('Tx_Extbase_MVC_Request'));
$view->setControllerContext($controllerContext);

$template = t3lib_extMgm::extPath($this->extKey) . 'res/marker_fluid.htm';
$view->setTemplatePathAndFilename($template);
$view->assign('names', $names);
$content = $view->render();
```

Konzept:

- Domain-driven design:
  - Umfeld der Problemstellung
  - Gemeinsame Sprache (Begrifflichkeiten, ...)
  - Objekt-Modell mit Eigenschaften, Funktionalitäten und Beziehungen

- Model-View-Controller (MVC):
  - Model: kapselt Anwendungslogik und -daten sowie Zugriffs- und Speicherlogik
  - Controller: stellt nach außen sichtbare, aufrufbare Funktionalitäten bereit, koordiniert Funktionalität welche im Model implementiert wird
  - View: Ausgabelogik, Darstellung der Daten (bei Extbase in der Regel mit Fluid)

Konventionen:

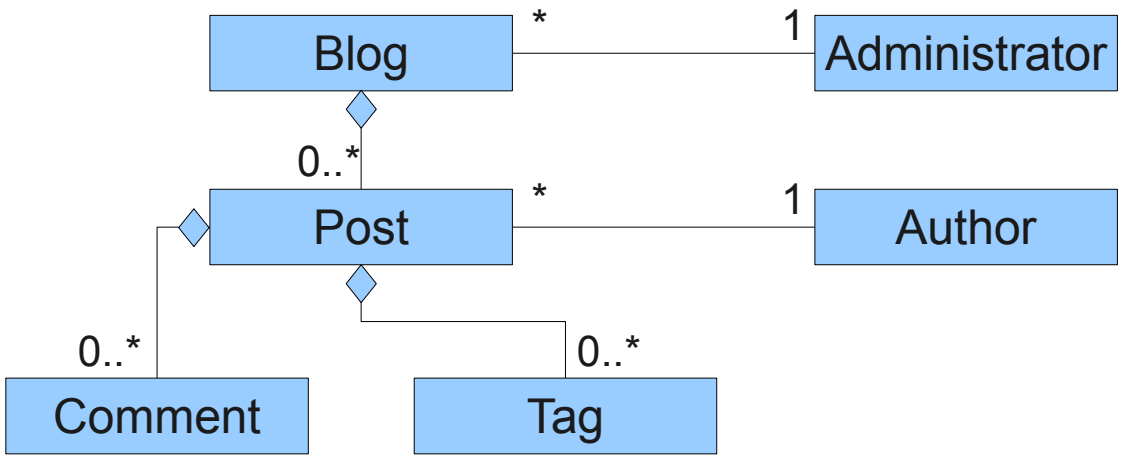
- Klassen: *UpperCamelCase*
- Methoden & Variablen: *lowerCamelCase*
- Bildung von Namen
  - Für Klassennamen: Tx_*[ExtensionName]*_*[Ebene1]*_*[Ebene2]*...
    z.B. Tx_BlogExample_Controller_BlogController, Tx_BlogExample_Domain_Model_Blog
  - Verzeichnisse/Dateien: einzelne Bestandteile durch Verzeichnisse abgebildet
    z.B. Classes/Controller/BlogController.php
    oder Classes/Domain/Model/Blog.php
  - Datenbank: lower-case mit Unterstrichen, z.B. tx_blogexample_domain_model_blog

Struktur:

- Objekte sind in einem „Repository“ gesammelt, welches die Verwaltung übernimmt
- Persistence-Manager übernimmt Speicherung / Rekonstruktion von Objekten
- Optional: Validatoren übernehmen die Prüfung von Objekten und Werten
- Ausgabe aus Rahmen (Layout), Template und Blöcke (Partials) zusammengesetzt

Überblick über Standard-Beispiel „blog_example“:

- Besteht aus:



- Ansatz: Blogs enthalten Posts, welche mit Kommentaren/Tags versehen sein können

- Funktionalitäten z.B. eines Blog-Post (siehe PostController):
  Listenansicht (Index), Einzelansicht, Erzeugen (+ Speichern), Editieren (+ Updaten), Löschen

... Live-Demo„blog_example“ ...

To-Do:

- Installation
- Einbindung in eine Seite
- Anlegen von Storage Folder und Einträgen
- Funktionstests

Stolperfallen bei ersten Schritten:

- Statische Konfiguration inkludieren
- Storage Folder angeben
- Anlagen eines „Administrator“ und einer „Person“ per Backend
  - Person muss einen „name“ (statt „username“) erhalten, sonst kein Dropdown-Eintrag sichtbar

Verzeichnis-/ Datei-Struktur:

Konfigurationen / Abhängigkeiten:

- Üblicherweise Nutzung von Konventionen statt expliziten Angaben („Convention over configuration“)
- Alternativ explizite Konfiguration (Tabellennamen, Spaltennamen, Abhängigkeiten)
- Methodenparameter („Type Hinting“)
- TypoScript
  - Standardmäßig unter „Configuration/TypoScript“, siehe constants.txt / setup.txt
  - Für Extension: unterhalb „plugin.tx_ourextension“
  - Für alle Extensions: unterhalb „config.tx_extbase“

Konfigurationen / Abhängigkeiten:

- TCA
  - Standardmäßig unter „Configuration/TCA“ in „tca.php“
- phpdoc (!)
  - Angabe von Typen für Eigenschaften (Models)
    - in Verbindung mit TCA somit Angabe der Verbindungen zwischen Objekten
  - Validatoren für Objekte und Eigenschaften sowie „@dontvalidate“
  - Lade-/Verarbeitungsanweisungen („@lazy“, „@cascade remove“)

Konfigurationen / Abhängigkeiten:

```
class Tx_BlogExample_Domain_Model_Blog extends Tx_Extbase_DomainObject_AbstractEntity {
  /**
   * The blog's title.
   *
   * @var string
   * @validate StringLength(minimum = 1, maximum = 80)
   */
  protected $title = '';

  /**
   * The posts of this blog
   *
   * @var Tx_Extbase_Persistence_ObjectStorage<Tx_BlogExample_Domain_Model_Post>
   * @lazy
   * @cascade remove
   */
  protected $posts;

  /**
   * Adds a post to this blog
   *
   * @param Tx_BlogExample_Domain_Model_Post $post
   * @return void
  */
  public function addPost(Tx_BlogExample_Domain_Model_Post $post) {
    $this->posts->attach($post);
  }
```

Konfigurationen / Abhängigkeiten:

```
class Tx_BlogExample_Controller_PostController extends Tx_Extbase_MVC_Controller_ActionController {

  // ...

  /**
   * Displays a form for creating a new post
   *
   * @param Tx_BlogExample_Domain_Model_Blog $blog The blog the post belogs to
   * @param Tx_BlogExample_Domain_Model_Post $newPost Fresh post object as basis for
rendering
   * @return string An HTML form for creating a new post
   * @dontvalidate $newPost
  */
  public function newAction(Tx_BlogExample_Domain_Model_Blog $blog,
        Tx_BlogExample_Domain_Model_Post $newPost = NULL) {
    $this->view->assign('authors', $this->personRepository->findAll());
    $this->view->assign('blog', $blog);
    $this->view->assign('newPost', $newPost);
  }
```

Konfigurationen / Abhängigkeiten:

```
class Tx_BlogExample_Controller_PostController extends Tx_Extbase_MVC_Controller_ActionController {

  // ...

  /**
   * Creates a new post
   *
   * @param Tx_BlogExample_Domain_Model_Blog $blog The blog the post belogns to
   * @param Tx_BlogExample_Domain_Model_Post $newBlog Fresh post object, not yet
added to repo
   * @return void
  */
  public function createAction(Tx_BlogExample_Domain_Model_Blog $blog,
        Tx_BlogExample_Domain_Model_Post $newPost) {
    $blog->addPost($newPost);
    $newPost->setBlog($blog);
    $this->flashMessages->add('Your new post was created.');
    $this->redirect('index', NULL, NULL, array('blog' => $blog));
  }
```

Ausgaben:

- Per Fluid
- Verschiedene Ausgabetypen („format“) möglich
  - standardmäßig „HTML“ mit Endung „.html“
  - Alternativ z.B. „txt“, „json“ oder „pdf“ möglich
- Rahmen (Layout), z.B. Resources/Private/Layouts/default.html

```
<h1>Blog Example</h1>
<hr />

<f:renderFlashMessages class="tx-extbase-flash-message" />
<f:render section="content" />

<hr />
<p>&copy; TYPO3 Association</p
```

Ausgaben:

- Template, z.B. Resources/Private/Templates/Post/new.html

```
<f:layout name="default" />

<f:section name="content">
    <div class="csc-header csc-header-n1">
        <h1 class="csc-firstHeader">Create a new Post</h1></div>
    <f:render partial="formErrors" arguments="{for: 'newPost'}" />
    <f:form method="post" controller="Post" action="create"  name="newPost"
            object="{newPost}" arguments="{blog : blog}">
        <label for="author">Author <span class="required">(required)</span></label><br />
        <f:form.select property="author" options="{authors}" optionValueField="uid"
            optionLabelField="fullName"><select><option>dummy</option></select>
        </f:form.select><br />
        <label for="title">Title <span class="required">(required)</span></label><br />
        <f:form.textbox property="title" /><br />
        <label for="content">Content <span class="required">(required)</span></label><br />
            <f:form.textarea property="content" rows="8" cols="46" /><br />
            <f:form.submit class="submit" value="Submit"/>
    </f:form>
</f:section>
```

Ausgaben:

- Blöcke (Partials), z.B. Resources/Private/Partials/formErrors.html

```
<f:form.errors for="{for}">
	<div class="error">
		{error.message}
		<f:if condition="{error.propertyName}">
			<p>
				<strong>{error.propertyName}</strong>:
				<f:for each="{error.errors}" as="errorDetail">
					{errorDetail.message}
				</f:for>
			</p>
		</f:if>
	</div>
</f:form.errors>
```

... Live-Demo„blog_example“ ...

To-Do:

- Funktionstests
  - Linkerzeugung (Fluid-ViewHelper)
  - Validatoren
  - Fehlerausgaben
  - Redirects

Eigene ViewHelper (Fluid):

- Eigener Implode-ViewHelper,
  Implementierung z,B. unter Classes/ViewHelpers/ImplodeViewHelper.php

```
class Tx_OurOwn_ViewHelpers_ImplodeViewHelper extends
Tx_Fluid_Core_ViewHelper_AbstractViewHelper {
	/**
	 * @param array $array List of values
	 * @param string $glue Glue used to join values
	 *
	 * @return string
	 */
	public function render($array,$glue = ',') {
		return implode($glue,$array);
	}
}
```

Verwendung in Fluid-Template

```
{namespace ourown = Tx_OurOwn_ViewHelpers}

...

<ourown:implode array="{answers}" glue=", " />
```

Eigene ViewHelper (Fluid):

- Eigener For-ViewHelper mit festen von/bis-Werten (Ausgabe fester Anzahl von Elementen), Implementierung z,B. unter Classes/ViewHelpers/ForViewHelper.php

```
class Tx_OurOwn_ViewHelpers_ForViewHelper extends
Tx_Fluid_Core_ViewHelper_AbstractViewHelper {
// ...
	public function render($start,$until,$as = null,$array = null,$arrayValue = null) {
		$returnValue = '';
// ...
		for($i = (int)$start;$i<(int)$until;$i++) {
			if($array !== null && $arrayValue !== null && count($array) > $i)
				$this->templateVariableContainer->add($arrayValue, $array[$i]);
			elseif($arrayValue !== null)
				$this->templateVariableContainer->add($arrayValue, null);
			if($as !== null)
				$this->templateVariableContainer->add($as, $i);
			$returnValue .= $this->renderChildren();
			if($arrayValue !== null)
				$this->templateVariableContainer->remove($arrayValue);
			if($as !== null)
				$this->templateVariableContainer->remove($as);
		}
		return $returnValue;
	}
```

Eigene ViewHelper (Fluid):

- Eigener For-ViewHelper mit festen von/bis-Werten (Ausgabe fester Anzahl von Elementen), Verwendung in Fluid-Template

```
{namespace ourown = Tx_OurOwn_ViewHelpers}

...

<ourown:for start="0" until="6" array="{entries}" arrayValue="oneentry">
    <f:if condition="{oneentry.comment}">
        <f:then>
            <td>{oneentry.comment}</b></td>
        </f:then>
        <f:else>
            <td>- no comment -</td>
        </f:else>
    </f:if>
</ourown:for>
```

Eigene ViewHelper (Fluid):

- Eigener InArray-ViewHelper
  Implementierung z,B. unter Classes/ViewHelpers/InArrayViewHelper.php

```
class Tx_OurOwn_ViewHelpers_InArrayViewHelper extends
Tx_Fluid_Core_ViewHelper_AbstractViewHelper {
	/**
	* @param mixed $needle The searched value
	* @param mixed $array The array
	*/
	public function render($needle,$array) {
		if (is_object($array)) {
			if (!$array instanceof Traversable) {
				throw new Tx_Fluid_Core_ViewHelper_Exception('ForViewHelper only supports
				arrays and objects implementing Traversable interface' , 1248728393);
			}
			$array = $this->convertToArray($array);
		}
		return in_array($needle,$array);
	}

// ...
}
```

Eigene ViewHelper (Fluid):

- Eigener InArray-ViewHelper
  Verwendung in Fluid-Template; hier z.B. auch verschachtelt mit einem Standard if-ViewHelper

```
{namespace ourown = Tx_OurOwn_ViewHelpers}

...

<f:if condition="{ourown:inArray(needle:'{ourObj.name}' array:'{testarray}')}">
<f:then>
JA
</f:then>
<f:else>
NEIN
</f:else>
</f:if>
```

Eigene ViewHelper (Fluid):

- Eigener PageBrowser-ViewHelper als Adapter zu bestehender Pagebrowser-Extension
  Implementierung z,B. unter Classes/ViewHelpers/PageBrowserViewHelper.php

```
class Tx_OurOwn_ViewHelpers_PageBrowserViewHelper extends
Tx_Fluid_Core_ViewHelper_AbstractViewHelper {
// ...
	public function render($numberOfPages,$prefixId,$pageParameterName = 'page') {
		// Get default configuration
		$conf = $GLOBALS['TSFE']->tmpl->setup['plugin.']['tx_pagebrowse_pi1.'];

		// Modify this configuration
		$conf += array(
			'pageParameterName' => $prefixId . '|'.$pageParameterName,
			'numberOfPages' => intval($numberOfPages)
		);

		// Get page browser
		$cObj = t3lib_div::makeInstance('tslib_cObj');
		$cObj->start(array(), '');
		return $cObj->cObjGetSingle('USER', $conf);
	}
}
```

Verwendung in Fluid-Template

```
<nmrc:pageBrowser numberOfPages="5" prefixId="{prefixId}" pageParameterName="page" />
```

Abspeichern mehrerer Datensätze in einem Submit (Extbase):

- Problem: Extbase nimmt (derzeit) eigentlich nur einen Datensatz zur Speicherung entgegen
- Eleganter per $this->propertyMapper->mapAndValidate(...), jedoch derzeit noch Probleme: http://lists.typo3.org/pipermail/typo3-project-typo3v4mvc/2010-January/002618.html

```
class Tx_OurOwn_Controller_ValuesController extends Tx_OurOwn_Controller_BaseController {

// ...

    public function saveValuesAction(Tx_OurOwn_Domain_Model_Item $item, $valueArray) {
        foreach($item->getValues() as $value) {
            if(in_array($value->getUid(), array_keys($valueArray))) {
                $newValue = $ValueArray[$value->getUid()];
                // @todo Extbase validator
                if($newValue >= 0 && $newValue <= 100)
                    $item->setFinalRatingValue($newValue);
            }
        }
        $this->redirect('index', 'Item');
    }
}
```

Abspeichern mehrerer Datensätze in einem Submit (Extbase):

- Problem: Extbase nimmt (derzeit) eigentlich nur einen Datensatz zur Speicherung entgegen

```
<f:form method="post" controller="Item" action="saveValues" name="item"
        arguments="{item : item}" >
    <table cellspacing="0" cellpadding="0">
        <tr>
            <f:for each="{valueArray}" as="valueInfo">
                <td>
                        <f:form.textbox name="valueArray[{valueInfo.value.uid}]" size="3"
value="{valueInfo.value.value}" />
                </td>
            </f:for>
        </tr>
        <tr>
            <td colspan="6" align="right">
                <f:form.submit style="width:120px;margin-right:5px;" value="Submit"  />
            </td>
        </tr>
    </table>
</f:form>
```

Manuelle Abfragen in Repository (Extbase):

- Daten werden „manuell“ aus beliebiger Quelle geholt und per DataMapper in Objekt geladen
- Nur zur Verdeutlichung was möglich WÄRE!
- DON'T DO THIS!

```
class Tx_OurOwn_Domain_Repository_ProfilRepository extends
        Tx_Extbase_Persistence_Repository {
// ...
    public function getProfilData($id) {
        $profilDaten = t3lib_div::makeInstance('Tx_OurOwn_Domain_Model_ProfilDaten');

        $res = $GLOBALS["TYPO3_DB"]->exec_SELECTquery(
            '*', 'tx_ourown_profil_daten', 'profil_id='.$id');

        $found = $GLOBALS["TYPO3_DB"]->sql_num_rows($res);

        If ($found) {
            $profilDaten->loadRow(mysql_fetch_assoc($res));
        }
        return $profilDaten;
    }
```

Manuelle Abfragen in Repository (Extbase):

- Sauberer Ansatz: z.B. Konfiguration der Datenquelle und Verwendung des PersistenceManager

```
plugin.tx_myextension {
    persistence {
        classes {
            Tx_MyExtension_Domain_Model_Person {
                mapping {
                    tableName = tt_address
                    recordType = Tx_MyExtension_Domain_Model_Person
                    columns {
                        birthday.mapOnProperty = dateOfBirth
                        street.mapOnProperty = thoroughfare
                    }
                }
            }
        }
    }
}
```

(Quelle: Buch „TYPO3-Extensions mit Extbase & Fluid“, O'Reilly, Rau/Kurfürst)

Stolperfalle: Finale Speicherung erfolgt automatisch erst am Ende einer Anfrage (Extbase):

- ObjectStorage nimmt Daten entgegen. Diese werden jedoch erst nach einer Anfrage (automatisch) vom PersistenceManager gespeichert.
- Falls vorher Zugriff auf alle Werte benötigt, selbst die Speicherung auslösen

```
public function saveAndFindAction(Tx_OurOwn_Domain_Model_Subject $subject,
Tx_OurOwn_Domain_Model_Item $item = null, $message = '') {
    if($item != null) {
        $item = t3lib_div::makeInstance('Tx_OurOwn_Domain_Model_Item');
        $item->setSubject($subject);
        $this->itemRepository->add($item);

        $persistenceManager = t3lib_div::makeInstance('Tx_Extbase_Persistence_Manager');
        $persistenceManager->persistAll();
    }
    $allItems = $this->itemRepository->findBySubject($subject);
    if(count($allItems) >= 6) {
        $subject->setStatus('closed');
        $this->subjectRepository->update($subject);
    }
    $this->view->assign('subject', $subject);
    $this->view->assign('allItems', $allItems);
}
```

Kleinere Standard-Extbase-Stolperfallen:

- @lazy im phpdoc benötigt, sofern sich Objekte gegenseitig referenzieren
  - Ansonsten erfolgt der automatische Ladevorgang in einer Endlosschleife

- @dontvalidate im phpdoc angeben bei newAction()
  - Speicherung erfolgt (üblicherweise) über separate Action, welche validiert
  - Wenn Validierung fehlschlägt, wird die vorherige Action wieder aufgerufen (newAction())
  - Endlosschleife sofern diese Fehleingaben zur Wiedervorlage auch validieren würde
- Falls Einträge nicht gefunden werden Storage-PID kontrollieren
  1. Ausgangspunkt im Formular des Plugins
  2. TypoScript-Setup „plugin.tx_extensionname.persistence.storagePid (kommaseparierte Liste)
  3. TypoScript-Setup „config.tx_extbase.persistence.storagePid“ (kommaseparatierte Liste)
  4. Auf Wurzelseite des Seitenbaums

Kleinere Standard-Extbase-Stolperfallen:

- Falls Einträge nicht gefunden werden Storage-PID kontrollieren
  1. Ausgangspunkt im Formular des Plugins
  2. TypoScript-Setup „plugin.tx_extensionname.persistence.storagePid (kommaseparierte Liste)
  3. TypoScript-Setup „config.tx_extbase.persistence.storagePid“ (kommaseparatierte Liste)
  4. Auf Wurzelseite des Seitenbaums

- Falls Einträge an falsche Stelle geschrieben werden Storage-PID kontrollieren
  1. TypoScript-Setup „plugin.tx_extensionname.persistence.classes.
     VollerKlassenName.newRecordStoragePid“ (einzelne PID)
  2. TypoScript-Setup „config.tx_extbase.persistence.classes.
     VollerKlassenName.newRecordStoragePid“ (einzelne PID)
  3. Ansonsten je erstes Objekt aus o.g. Punkten (1-4) von Lesereihenfolge,
     letztlich also Wurzelseite des Seitenbaums

- viewhelpertest: https://svn.typo3.org/TYPO3v4/CoreProjects/MVC/viewhelpertest/
- extbase_kickstarter: https://svn.typo3.org/TYPO3v4/Extensions/extbase_kickstarter/trunk/
- doc_extbase: https://svn.typo3.org/TYPO3v4/CoreProjects/MVC/doc_extbase/trunk/
- Extension "efempty" (siehe TER): Extbase/Fluid, leerer Container für erste Experimente
- Extbase/Fluid Cheat-Sheet:
  http://www.typofaktum.de/fileadmin/slides/ExtbaseFluidCheatSheetTypofaktum.pdf
- TYPO3 4.3: verstehen, implementieren, erweitern (Videotraining)
  Addison-Wesley/video2brain, Aster/Hauser/Neufeind, ISBN 978-3-8273-6290-2
- TYPO3-Extensions mit Extbase & Fluid (Buch)
  O' Reilly, Rau/Kurfürst, ISBN 978-3-8972-1965-6
- TYPO3-Extensions. Professionelle Frontend- und Backend-Programmierung
  Hanser, Ebner/Lobacher/Ulbrich, 978-3-4464-1557-7

Es gibt viel zu entdecken!

Danke fürs Zuhören

sowie

viel Erfolg und Spaß

mit Extbase/Fluid!

Link zu den Slides: http://talks.speedpartner.de

Bei Fragen stehen wir selbstverständlich gerne zur Verfügung:

Stefan Neufeind, neufeind@speedpartner.de
SpeedPartner GmbH, http://www.speedpartner.de/