Inside TYPOlight

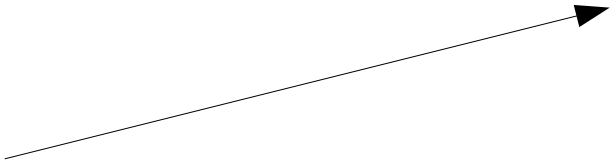
Überblick

- Teil 1: TYPOlight-Ordnerstruktur
- Teil 2: TYPOlight-Framework
- Teil 3: Libraries
- Teil 4: Lebenszyklus einer Frontend-Anfrage
- Teil 5: Data Container Arrays
- Teil 6: TYPOlight anpassen

Teil 1: TYPOlight-Ordnerstruktur

Teil 1: TYPOlight-Ordnerstruktur

- plugins
- system
- templates
- tl_files
- typolight



- system
  - config
  - drivers
  - html
  - libraries
  - logs
  - modules
  - themes
  - tmp

Teil 1: TYPOlight-Ordnerstruktur

- plugins
  - Externe Skripte, die systemübergreifend verwendet werden (TinyMCE, MooTools, SWFobject, phpmailer etc.)
- templates
  - Eigene Templates, Include-Dateien, SQL-Dumps
  - Der Ordner wird beim Live Update nicht überschrieben
- tl_files
  - Zentrale Dateiverwaltung
- typolight
  - Administrationsbereich in eigenem Unterverzeichnis ermöglicht zusätzlichen Schutz per .htaccess-Datei

Teil 1: TYPOlight-Ordnerstruktur

- system/config
  - Zentraler Speicherort für Konfigurationsdateien
- system/drivers
  - Mischung aus Controller, Model und View (z.B. DC_Table.php)
  - Datenbank-Adapter (z.B. DB_Mysql.php, DB_Oracle.php)
- system/html
  - Über HTTP erreichbares Cache-Verzeichnis (z.B. Thumbnails)
- system/libraries
  - Bibliotheken abstrahieren alle möglichen Aufgaben wie z.B. DB-Kommunikation, Dateiverwaltung (SMH), sicheres Abfragen von Benutzereingaben, E-Mail-Versand, Datumsberechnung etc.

Teil 1: TYPOlight-Ordnerstruktur

- system/logs
  - Über HTTP nicht erreichbarer Speicherort für Log-Dateien
- system/modules
  - Zentraler Speicherort für Module
  - Auch das Backend selbst ist „nur“ ein Modul
  - Beliebig erweiterbar um jede erdenkliche Funktionalität
- system/themes
  - Speicherort für Backend-Themen
- system/tmp
  - Über HTTP nicht erreichbares Cache-Verzeichnis

Teil 2: TYPOlight-Framework

Teil 2: TYPOlight-Framework

- TYPOlight und MVC
  - MVC = Model-View-Controller
  - MVC-Elemente sind in TYPOlight vorhanden
  - Trotzdem kein klassisches MVC-Framework
- Abweichungen vom MVC-Prinzip
  - Models werden nur für Benutzer verwendet
  - Divers sind eine Mischung aus Controller, Model und View mit erweiterter CRUD-Funktionalität (Create, Read, Update, Delete)
  - Backend-Views (z.B. Formulare) werden automatisch erstellt
  - Kein klassisches URI-Routing zu Gunsten Suchmaschinen-freundlicher URLs im Frontend

Teil 2: TYPOlight-Framework – Models

- Models
  - In TYPOlight nur für Benutzer implementiert
  - ```
    $this->import('BackendUser', 'User');
    echo $this->User->isAdmin; // False

    $this->User->admin = 1;
    $this->User->save();

    echo $this->User->isAdmin; // True
    ```
  - Models spielen nur eine untergeordnete Rolle, weil das Ziel war, umfassende Driver zu programmieren, die anhand von Meta-Informationen automatisch verschiedene Views und Formulare erstellen und verarbeiten können
  - Statt pro Tabelle ein Model, ein Controller und die benötigten Views anzulegen, soll der Driver alle Fälle automatisch abdecken
    ```

Teil 2: TYPOlight-Framework – Views

- Views
  - **Layouts** (z.B. fe_page)
  - **Views** (z.B. mod_newslist)
  - **Partials** (z.B. layout_short)
  - Im „TYPOlight-Sprachgebrauch“ wird diese Unterscheidung nicht getroffen; es hat sich der Begriff „Template“ durchgesetzt
- Views laden
  - Zuerst wird im templates-Verzeichnis gesucht
  - Danach in den „templates“-Ordnern der aktiven Module
  - Der erste Treffer gewinnt (ist das Template z.B. im Modul „backend“ vorhanden, wird ein eventuell gleichnamiges Template im Modul „news“ niemals geladen)

Teil 2: TYPOlight-Framework – Views

- Views parsen
  - **Template::parse()** lädt einen View
  - ersetzt die darin enthaltenen Platzhalter
  - gibt das Ergebnis als String zurück
- Views ausgeben
  - **Template::output()** lädt einen View
  - ersetzt die darin enthaltenen Platzhalter
  - führt je nach Bereich zusätzliche Aufgaben aus
  - gibt das Ergebnis auf dem Bildschirm aus

Teil 2: TYPOlight-Framework – Views

- BackendTemplate::output() im Detail
  - Laden der Rich Text Editor-Konfiguration
  - Einfügen der dynamischen JavaScript- und CSS-Dateien
  - Ausführen des „outputBackendTemplate“-Hooks
  - Einfügen des Copyright-Hinweises
  - Prüfen und Aktivieren der GZip-Kompression
  - Senden der HTTP-Header
  - Ausgabe des XHTML-Codes
  - Ausgabe der Debug-Informationen (falls aktiv)

Teil 2: TYPOlight-Framework – Views

- FrontendTemplate::output() im Detail
  - Generieren der Suchindex-URL
  - Einlesen der Suchbegriffe aus den Artikeln
  - Ausführen des „outputFrontendTemplate“-Hooks
  - Erstellen der Cache-Dateien und Senden der Cache-Header
  - Ersetzen der Insert-Tags (falls aktiv)
  - Hinzufügen der Datei zum Suchindex (falls aktiv)
  - Einfügen des Copyright-Hinweises
  - Prüfen und Aktivieren der GZip-Kompression
  - Senden der HTTP-Header
  - Ausgabe des XHTML-Codes
  - Ausgabe der Debug-Informationen (falls aktiv)

Teil 2: TYPOlight-Framework – Views

- Dynamische Skripte
  - **TL_CSS**: Hinzufügen von CSS-Dateien möglich
  - **TL_JAVASCRIPT**: Hinzufügen von JavaScript-Dateien möglich
  - **TL_HEAD**: Hinzufügen von beliebigem Code möglich
  - `$GLOBALS['TL_CSS'][] = 'system/modules/news/style.css';`
- Automatische Views im Backend
  - **List View**: Auflisten der Datensätze einer Tabelle
  - **Parent View**: Auflisten der Kind-Elemente eines Eltern-Elements
  - **Tree View**: Hierarchische Auflistung als Baum
  - Automatische Erstellung der Eingabeformulare für neue bzw. zu bearbeitende Datensätze (sonst hätte pro Tabelle und Action ein eigener View angelegt werden müssen)

Teil 2: TYPOlight-Framework – Controller

- Controller-Funktionalität (CRUD)
  - **list()**: Auflistung aller Datensätze
  - **show()**: Darstellung eines einzelnen Datensatzes
  - **create()**: Formular zum Anlegen eines neuen Datensatzes
  - **save()**: Speichern eines neues Datensatzes
  - **edit()**: Formular zur Bearbeitung eines Datensatzes
  - **update()**: Aktualisierung eines bestehenden Datensatzes
  - **delete()**: Löschen eines Datensatzes

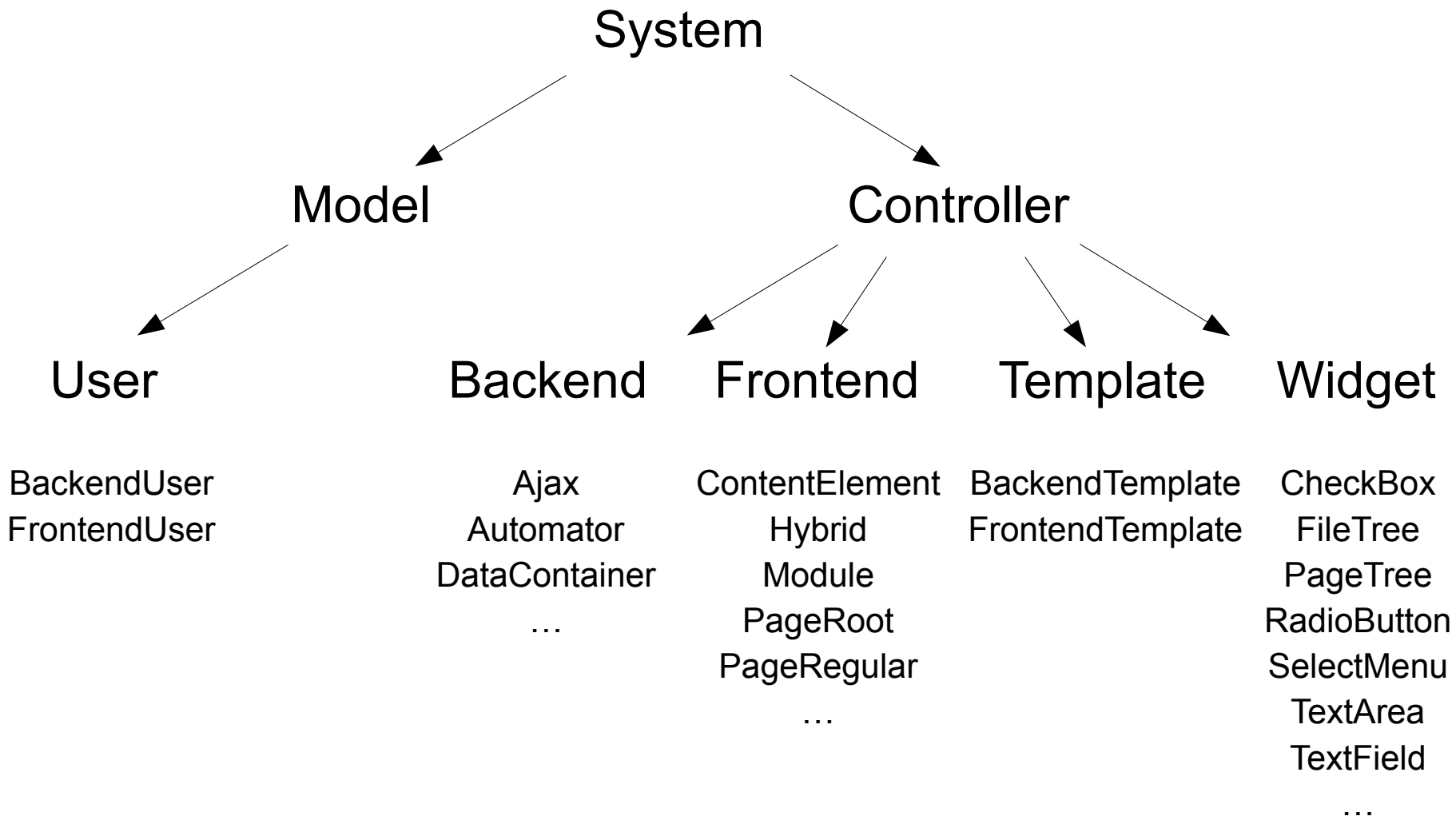
Teil 2: TYPOlight-Framework – Controller

- TYPOlight-Driver bieten zusätzlich
  - **cut()**: Verschieben eines Datensatzes
  - **copy()**: Duplizieren eines Datensatzes
  - **deleteAll()**: Löschen mehrerer Datensätze auf einmal
  - **editAll()**: Bearbeiten mehrerer Datensätze auf einmal
  - **undo()**: Wiederherstellung eines gelöschten Datensatzes
  - Wiederherstellung früherer Versionen eines Datensatzes
- „Virtueller Controller“
  - Zur Laufzeit wird anhand der DCA-Konfiguration ein virtueller Controller erzeugt, der sich um das Bereitstellen der Formulare, das Prüfen der Eingaben und das Speichern der Daten kümmert
  - Bietet deutlich mehr Möglichkeiten als ein CRUD-Controller

Teil 3: Libraries

Libraries

- Systemarchitektur



Teil 3: Libraries – System

- Basisklasse „System“
  - Bereitstellen systemweiter Methoden
  - **import()** instantiiert weitere Objekte
  - **log()** fügt einen Eintrag in die Log-Tabelle ein
  - **reload()** lädt die aktuelle Seite neu
  - **redirect()** leitet zu einer anderen Seite weiter
  - **parseDate()** gibt ein formatiertes Datum zurück
  - **setCookie()** setzt ein Cookie
- Vorteil von System::import()
  - Erkennt Singletons automatisch
  - Prüft selbständig, ob ein Objekt bereits existiert

Teil 3: Libraries – Controller

- class Controller extends System
  - **getFrontendModule()** gibt ein Frontendmodul zurück
  - **getArticle()** gibt einen Artikel zurück
  - **getContentElement()** gibt ein Inhaltselement zurück
  - **resizeImage()** erzeugt ein Thumbnail im Ordner system/html
  - **printArticleAsPdf()** gibt einen Artikel als PDF-Datei aus
  - **replaceInsertTags()** ersetzt Insert-Tags
  - **sendFileToBrowser()** öffnet den „Speichern unter …“-Dialog
  - **getFrontendUrl()** erzeugt eine Frontend-URL
  - **removeOldFeeds()** entfernt nicht mehr benötigte XML-Dateien
  - …

Teil 3: Libraries – Controller

- class Backend extends Controller
  - **getBackendModule()** gibt ein Backendmodul zurück
  - **getSearchablePages()** gibt alle durchsuchbaren Seiten zurück
  - **createPageList()** gibt die Seiten als Dropdown-Menü zurück
  - **createFileList()** gibt die Dateien als Dropdown-Menü zurück
- class Frontend extends Controller
  - **getPageIdFromUrl()** gibt die ID der aktuellen Seite zurück
  - **getRootIdFromUrl()** ermittelt den Startpunkt der Webseite
  - **jumpToOrReload()** aktualisiert die Seite bzw. leitet weiter
  - **getLoginStatus()** ermittelt, ob ein Benutzer angemeldet ist
  - **parseMetaFile()** liest eine „meta.txt“-Datei ein

Teil 3: Libraries – Datenbankabstraktion

- Datenbankabstraktion
  - SQL92-Standard als gemeinsamer Nenner
  - Nur spezifische Funktionen wie z.B. LIMIT werden in eigenen Methoden gekapselt, die pro Adapter definiert werden
  - Dadurch bleibt die Schnittstelle unkompliziert und flexibel
  - ```
    $db = $this->Database;

    $stmt = $db->prepare('SELECT * FROM tl_user WHERE name=?');
    $stmt->limit(1); // Nicht einheitlich, daher gekapselt
    $user = $stmt->execute('Theo Test');

    while ($user->next())
    {
        echo $user->name;
    }
    ```
  - Einfacher Zugriff auf die einzelnen Felder im Result-Set
    ```

Teil 3: Libraries – Datenbankabstraktion

- Vorteile der DB-Abstraktion
  - Beliebig komplexe Queries (Joins, Subqueries) möglich
  - Automatische Quotierung verhindern SQL-Injections
  - „Lazy Initialization“ der Result-Sets
  - Einheitliche und vom DB-System unabhängige Schnittstelle
- Einschränkungen der DB-Abstraktion
  - Keine Abstraktion zur Änderung von Tabellen vorhanden
  - Spezielle Handhabung von BLOB/CLOB-Feldern in Oracle wird von der Abstraktionsschicht nicht berücksichtigt
  - Vollständige Funktionalität de facto nur für MySQL
  - Keine Kontrolle, ob ein Programmierer sich tatsächlich an den SQL92-Standard hält (spezifische Queries sind möglich)

Teil 3: Libraries – Dateizugriff

- Dateizugriff und Safe Mode Hack
  - Wird PHP als Modul betrieben, läuft es normalerweise unter dem Benutzer „wwwrun“, „nobody“ oder „www-data“
  - Per FTP übertragene Dateien gehören jedoch dem jeweiligen FTP-Benutzer (z.B. „web5“ oder „xa2387“)
  - Daher verbietet der Server dem PHP-Prozess (und damit TYPOlight) den Zugriff auf die vermeintlich fremden Dateien
- Lösungen
  - PHP als CGI mit suPHP betreiben
  - Den PHP-Prozess unter demselben Benutzer betreiben, dem auch die Dateien gehören (z.B. bei allinkl.com möglich)
  - Dateizugriff per FTP (Safe Mode Hack)

Teil 3: Libraries – Dateizugriff

- Abstraktionsschicht „Files“
  - Die Files-Bibliothek lädt anhand der Konfiguration entweder einen PHP- oder einen FTP-Adapter für den Dateizugriff
  - **mkdir()** erstellt ein neues Verzeichnis
  - **rmdir()** entfernt ein Verzeichnis
  - **fopen()** öffnet eine neue Datei
  - **fputs()** schreibt in eine Datei
  - **fclose()** schließt eine Datei
  - **rename()** benennt eine Datei oder ein Verzeichnis um
  - **copy()** kopiert eine Datei oder ein Verzeichnis
  - **delete()** löscht eine Datei
  - **chmod()** ändert die Zugriffsrechte einer Datei oder eines Ordners

Teil 3: Libraries – Dateizugriff

- Dateizugriff mittels „Files“
  - Der Aufruf erfolgt wie bei den nativen PHP-Funktionen
  - ```
    $this->import('Files');

    $fh = $this->Files->fopen('system/tmp/test.txt', 'wb');
    $this->Files->fputs($fh, 'Dies ist ein Test.');
    $this->Files->fclose($fh);
    ```
  - Dateipfade müssen immer relativ angegeben werden!
- Vereinfachter Zugriff mittels „File“ und „Folder“
  - Hilfsklassen für den Zugriff auf Dateien bzw. Ordner
  - Verzeichnisse können rekursiv angelegt werden
  - Dateiinformationen wie Pfad, Dateiendung, Zugriffszeiten, Breite und Höhe sowie MIME-Typ stehen standardmäßig zur Verfügung
    ```

Teil 3: Libraries – Sicherheit in TYPOlight

- Sicherheit in TYPOlight
  - Input-Klasse kapselt den Zugriff auf Benutzereingaben
  - **Schritt 1**: HTML-Entities werden dekodiert
  - **Schritt 2**: Unicode-Entities werden dekodiert (XSS-Schutz)
  - **Schritt 3**: JavaScript-Anweisungen werden entfernt (im „Strict-Mode“ werden zusätzlich alle Event-Attribute entfernt)
  - **Schritt 4**: Nicht erlaubte HTML-Tags werden entfernt
  - **Schritt 5**: Potentiell gefährliche Sonderzeichen werden kodiert
- Zusätzlicher XSS-Schutz
  - Das <script>-Tag gehört **nicht** in der Liste der erlaubten Tags
  - Die Einbindung von JavaScript ist sonst in allen Feldern möglich, in denen HTML-Eingaben erlaubt sind!

Teil 3: Libraries – Sicherheit in TYPOlight

- Serverumgebung auslesen
  - Die Environment-Klasse ermöglicht den Betriebssystem-unabhängigen Zugriff auf Umgebungsvariablen
  - Potentiell gefährlicher Code wird dabei entfernt (z.B. kann `$_SERVER['HTTP_USER_AGENT']` JavaScript-Code enthalten)
- Formulare absichern
  - Beim Absenden von Formularen wird automatisch geprüft, ob sie tatsächlich von derselben Seite stammen (Referer-Prüfung)
  - Einige Anonymizer und Security-Tools unterdrücken die Referer-Adresse, was zu einer Fehlermeldung in TYPOlight führt
  - Bei deaktivierter Referer-Prüfung (keinesfalls empfohlen) sollten alle Formulare eine Sicherheitsfrage (Captcha) enthalten
  - Ein Captcha bietet zusätzlich Schutz gegen Spam

Teil 3: Libraries – Sicherheit in TYPOlight

- Login und Authentifizierung
  - Eine TYPOlight-Benutzersitzung ist an die PHP-Session und an die IP-Adresse des Benutzer gebunden
  - IP-Bindung seit Version 2.7 abschaltbar (nicht empfohlen)
  - Jede aktive Sitzung wird in der Datenbank gespeichert
  - Das Cookie enthält nur eine Prüfsumme, aber keine relevanten Daten (Verfallszeiten, IDs, Benutzerinformationen)
  - Recall-Erweiterung ermöglicht persistente Logins im Frontend
- Benutzerwechsel und Frontendvorschau
  - Die Implementierung erlaubt das Übernehmen von Sitzungen
  - Administratoren können zu anderen Benutzern wechseln (sowohl im Backend als auch in der Frontendvorschau)

Teil 3: Libraries – Sicherheit in TYPOlight

- Daten verschlüsselt speichern
  - Jedes Tabellenfeld kann verschlüsselt gespeichert werden
  - Konfiguration im Data Container Array
  - `$GLOBALS['TL_DCA']…['eval']['encrypt'] = true;`
  - Verschlüsselung erfolgt mit dem Encryption Key, der bei der Installation festgelegt wird (einmal verschlüsselte Daten können nur mit diesem Key wiederhergestellt werden!)
  - PHP-Modul „mcrypt“ muss verfügbar sein
- Verschlüsselung im TYPOlight-Core
  - Die Funktionalität wird im Core bislang nicht genutzt
  - Bei sensiblen Produkten kann das verschlüsselte Speichern von Mitgliederdaten sinnvoll sein (Änderung in der dcaconfig.php)

Teil 3: Libraries – Widgets

- Widgets
  - Widgets = Formularfelder
  - Standardfelder wie z.B. TextField, CheckBox, SelectMenu
  - TYPOlight-spezifische Felder wie z.B. PageTree, FileTree oder die verschiedenen Wizards (z.B. TableWizard)
- Basisklasse „Widget“
  - Bereitstellen der gemeinsamen Funktionalität
  - **generate()** gibt ein Formularfeld aus
  - **validate()** überprüft die Eingabe in ein Formularfeld
  - **hasErrors()** prüft, ob Fehler aufgetreten sind
  - **getErrors()** gibt die Fehlermeldungen als Array zurück

Teil 3: Libraries – Widgets

- Widgets ausgeben
  - **generateLabel()** gibt die Feldbezeichnung zurück
  - `<label for="ctrl_name">Name</label>`
  - **generate()** gibt das Formularfeld zurück
  - `<input type="text" id="ctrl_name" name="name" />`
  - **generateWithError()** gibt das Feld mit Fehlermeldung zurück
  - ```
    Bitte füllen Sie das Feld aus.
    <input type="text" id="ctrl_name" name="name" />
    ```
  - **generateWithError(**true**)** dreht die Reihenfolge um
  - ```
    <input type="text" id="ctrl_name" name="name" />
    <p class="error">Bitte füllen Sie das Feld aus.</p>
    ```

Teil 3: Libraries – Widgets

- Fehlermeldungen ausgeben
  - **getErrors()** gibt die Fehlermeldungen als Array zurück
  - **getErrorAsString()** gibt die erste Fehlermeldung zurück
  - **getErrorAsString(**2**)** gibt die dritte Fehlermeldung zurück
  - **getErrorsAsString()** gibt alle Fehlermeldungen getrennt durch einen Zeilenumbruch (`<br />`) zurück
  - **getErrorsAsString(**', '**)** gibt alle Fehlermeldungen getrennt durch ein Komma zurück
  - **getErrorAsHTML()** gibt die erste Fehlermeldung als HTML-Code zurück (`<p class="error">…</p>`)
  - **getErrorAsHTML(**2**)** gibt die dritte Fehlermeldung zurück

Teil 3: Libraries – Widgets

- Standardtemplate

```
<!-- View bzw. Partial -->
<?php echo $this->generateLabel(); ?>
<?php echo $this->generateWithError(); ?>

<!-- Ausgabe -->
<label for="ctrl_name">Ihr Name</label>
<input type="text" id="ctrl_name" name="name" />

<!-- Ausgabe mit Fehlermeldung -->
<label for="ctrl_name">Ihr Name</label>
<p class="error">Bitte füllen Sie das Feld aus.</p>
<input type="text" id="ctrl_name" name="name" />

<!-- Ausgabe bei generateWithError(true) -->
<label for="ctrl_name">Ihr Name</label>
<input type="text" id="ctrl_name" name="name" />
<p class="error">Bitte füllen Sie das Feld aus.</p>
```

Teil 3: Libraries – Widgets

- Komplexeres Beispiel

```
<!-- View bzw. Partial -->
<fieldset>
<?php if ($this->hasErrors()): ?>
<p class="flash"><?php echo $this->getErrorAsString(); ?></p>
<?php endif; ?>
<div>
  <?php echo $this->generateLabel(); ?><br />
  <?php echo $this->generateWithError(); ?>
</div>
</fieldset>

<!-- Ausgabe -->
<fieldset>
<p class="flash">Bitte füllen Sie das Feld aus.</p>
<div>
  <label for="ctrl_name">Ihr Name</label><br />
  <input type="text" id="ctrl_name" name="name" />
</div>
</fieldset>
```

Teil 3: Libraries – Widgets

- Eingabeprüfung
  - **Pflichtfeld**: das Feld darf nicht leer sein
  - **Mindestlänge**: Eingabe von mindestens n Zeichen
  - **Maximallänge**: Eingabe von maximal n Zeichen
  - **Zahlen & Buchstaben**: nur Zahlen bzw. Buchstaben erlaubt
  - **Datum & Zeit**: nur Datums- bzw. Zeitformate erlaubt
  - **E-Mail-Adresse**: Eingabe muss eine E-Mail-Adresse sein
  - **Telefonnummer**: Eingabe muss eine Telefonnummer sein
  - **URL**: Eingabe muss eine URL bzw. Domain sein
  - **Prozentwert**: Eingabe muss eine Zahl zwischen 0 und 100 sein
  - Beliebig erweiterbar über Hook „addCustomRegexp“

Teil 3: Libraries – Feeds erstellen

- class Feed extends System
  - Getter- und Setter-Methoden für Eigenschaften
  - **addItem()** fügt ein FeedItem hinzu
  - **generateRss()** gibt den Feed im RSS-Format aus
  - **generateAtom()** gibt den Feed im Atom-Format aus
- class FeedItem extends System
  - Getter- und Setter-Methoden für Eigenschaften
  - **addEnclosure()** fügt dem Element ein Enclosure hinzu

Teil 3: Libraries – Feeds erstellen

- Vereinfachtes Beispiel

```php
<?php

$feed = new Feed();

$feed->title = 'TYPOlight-Usertreffen 2009';
$feed->description = 'Live-Meldungen vom Usertreffen';

$item = new Item();

$item->title = 'Teilnehmerrekord';
$item->description = 'Rekord mit über 70 Teilnehmern!';

$feed->addItem($item);

echo $feed->generateRss();

?>
```

Teil 3: Libraries – Feeds erstellen

- Ausgabe im RSS-Format

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
  <channel>
    <title>TYPOlight-Usertreffen 2009</title>
    <description>Live-Meldungen vom Usertreffen</description>
    <link>…</link>
    <language>…</language>
    <pubDate>…</pubDate>
    <item>
      <title>Teilnehmerrekord</title>
      <description><![CDATA[Rekord mit …]]></description>
      <link>…</link>
      <pubDate>…</pubDate>
      <guid>…</guid>
    </item>
  </channel>
</rss>
```

Teil 3: Libraries – Periodic command scheduler

- Periodic command scheduler
  - Automatische Skriptausführung in bestimmten Intervallen
  - Stündliche, tägliche und wöchentliche Ausführung möglich
  - Keine exakte Terminierung wie beim Cron-Job
  - Nutzung in eigenen Erweiterungen möglich
  - `$GLOBALS['TL_CRON']['hourly'][] = array('Rates', 'update');`
- Tägliche Ausführung
  - Feed-Dateien neu erstellen
  - Bereinigung der temporären Verzeichnisse
- Wöchentliche Ausführung
  - Stylesheets und XML-Sitemaps neu erstellen

Teil 3: Libraries – Periodic command scheduler

- Nutzung mit echtem Cron-Job
  - Trigger des PCM durch einen echten Cron-Job möglich
  - Stündliche Ausführung der Datei cron.php im TYPOlight-Ordner
  - `0 * * * * php /home/www/typolight/cron.php`
- Entfernen der Trigger
  - Views be_login.tpl und fe_page.tpl
  - ```
    <!-- indexer::stop -->
    <img src="<?php echo $this->base; ?>cron.php" … />
    <!-- indexer::continue -->
    ```
  - Die drei Zeilen müssen komplett entfernt werden
    ```

Teil 4: Lebenszyklus einer Frontend-Anfrage

Teil 4: Lebenszyklus einer FE-Anfrage – Initialisierung

- __autoload()
  - Klassen werden in TYPOlight automatisch geladen
  - Zuerst wird im Ordner „libraries“ gesucht
  - Danach werden alle Modulordner durchsucht
  - Dabei keine Unterscheidung nach aktiv/inaktiv und strikte alphabetische Reihenfolge, da das Config-Objekt zum Zeitpunkt der Funktionsdefinition noch nicht besteht!
  - DOMPDF_autoload (falls installiert)
  - Fehlermeldung wenn die Klasse nicht gefunden wurde
- Controller::classFileExists()
  - Prüft die Existenz einer Klasse bzw. Klassendatei
  - Berücksichtigt aktive/inaktive Module

Teil 4: Lebenszyklus einer FE-Anfrage – Initialisierung

- scan()
  - Scannt ein Ordner nach Unterordnern und Dateien
  - Wie scandir(), gibt aber '.' und '..' nicht zurück
  - Eingebauter Cache und open_basedir-Kompatibilität
- specialchars()
  - Wandelt bestimmte Zeichen in HTML-Entities um
  - Wie htmlspecialchars(), lässt jedoch Und-Zeichen unverändert, um Doppelumwandlungen zu verhindern (`&amp;`)
- deserialize()
  - Wandelt ein serialisiertes Array zurück in ein Array
  - Wie unserialize(), gibt aber im Fehlerfall das Argument zurück

Teil 4: Lebenszyklus einer FE-Anfrage – Initialisierung

- trimsplit()
  - Zerlegt eine Zeichenkette anhand eines Trennzeichens
  - Wie preg_split(), jedoch zusätzliche Anwendung von trim()
  - Trennzeichen kann ein regulärer Ausdruck sein
- ampersand()
  - Wandelt alle Und-Zeichen in einem String in Entities (Argument true) bzw. einfache Und-Zeichen (Argument false) um
- natcaseksort()
  - Ergänzung zur PHP-Funktion natcasesort()
  - Erlaubt das natürliche Sortieren eines Arrays anhand der Schlüssel anstatt anhand der Werte

Teil 4: Lebenszyklus einer FE-Anfrage – Initialisierung

- array_insert()
  - Fügt einen Wert an einer bestimmten Stelle in ein Array ein
  - Der Wert kann selbst auch ein Array sein
- array_duplicate()
  - Dupliziert einen bestimmten Wert in einem Array
  - Die Kopie wird direkt nach dem Original eingefügt
- array_move_up()
  - Verschiebt einen bestimmten Wert eine Position nach oben
  - Entspricht einem Tausch zweier Werte

Teil 4: Lebenszyklus einer FE-Anfrage – Initialisierung

- array_move_down()
  - Verschiebt einen bestimmten Wert eine Position nach unten
  - Entspricht einem Tausch zweier Werte
- array_delete()
  - Entfernt einen bestimmten Wert aus einem Array
  - Die Schlüssel werden danach neu berechnet
- array_is_assoc()
  - Prüft, ob ein Array assoziativ ist
  - Sind alle Schlüssel durchgehend numerisch und in aufsteigender Reihenfolge angeordnet, ist das Array nicht assoziativ

Teil 4: Lebenszyklus einer FE-Anfrage – Initialisierung

- mbstring.php
  - Ersatzbibliothek für die PHP-Erweiterung mbstring
  - Zum Beispiel notwendig bei Strato Shared Hosting Accounts
  - Stellt Funktionen für die Bearbeitung von internationalen Zeichenketten und Sonderzeichen zur Verfügung
  - Besonders wichtig: utf8_strtolower() und utf8_strtoupper()
- php.ini anpassen
  - Session-IDs in URLs unterdrücken (PHPSESSID)
  - Error- und Exception-Handler definieren
  - Pfad für die error.log-Datei setzen
- PHP-Session starten

Teil 4: Lebenszyklus einer FE-Anfrage – Konfiguration

- Config-Objekt laden
  - Zuerst wird die localconfig.php eingelesen, um zu erfahren, welche Erweiterungen deaktiviert wurden
  - Danach system/modules/backend/config/config.php
  - Danach system/modules/frontend/config/config.php
  - Anschließend die Konfigurationsdateien (config.php) aller aktiven Erweiterungen in alphabetischer Reihenfolge
  - Abschließend noch einmal die localconfig.php, um die lokalen Anpassungen in die Konfiguration zu übernehmen
- Inaktive Erweiterungen
  - Werden nicht durchsucht und initialisiert
  - Je mehr inaktive Erweiterungen, desto besser die Performance

Teil 4: Lebenszyklus einer FE-Anfrage – Konfiguration

- Konfigurationsarrays
  - Backend-Module
    `$GLOBALS['TL_CONFIG']['BE_MOD']`
  - Backend-Formularfelder
    `$GLOBALS['TL_CONFIG']['BE_FFL']`
  - Backend-Seitentypen
    `$GLOBALS['TL_CONFIG']['TL_PTY']`
  - Frontend-Module
    `$GLOBALS['TL_CONFIG']['FE_MOD']`
  - Inhaltselemente
    `$GLOBALS['TL_CONFIG']['TL_CTE']`
  - Frontend-Formularfelder
    `$GLOBALS['TL_CONFIG']['TL_FFL']`

Teil 4: Lebenszyklus einer FE-Anfrage – Konfiguration

- Standardobjekte laden
  - Environment-Objekt zur Abfrage der Umgebungsvariablen
  - Input-Objekt zur Verarbeitung von Benutzereingaben
- Weitere Konfiguration
  - Error reporting gemäß localconfig.php setzen
  - Zeitzone gemäß localconfig.php setzen
  - Relativen Pfad zu TYPOlight setzen (falls noch nicht geschehen)
  - Mbstring-Kodierung gemäß localconfig.php setzen
  - Browsersprache ermitteln und speichern
- Referer-Prüfung
  - Nur wenn Formulare übermittelt wurden

Teil 4: Lebenszyklus einer FE-Anfrage – Seite finden

- Seite aus dem Cache laden
  - Suchen einer gespeicherten Version der Seite
  - Prüfung der Verfallszeit und ggf. Ausgabe
  - In diesem Fall wäre die Anfrage fertig abgearbeitet
  - Alle folgenden Schritte kann man also durch Cache einsparen
- FrontendUser-Objekt laden
  - Es wird eine Datenbankverbindung aufgebaut
  - User-Objekt wird vorerst nur initialisiert
  - Noch kein Aufruf von authenticate() oder login()
- Login-Status ermitteln
  - Prüfung, ob ein Backend- oder Frontend-Benutzer angemeldet ist

Teil 4: Lebenszyklus einer FE-Anfrage – Seite finden

- Seite anhand der URL finden
  - ID bzw. Alias der Seite aus der URL extrahieren
  - Laden der Seite aus der Datenbank
  - Eventuell Zuordnung zu einem Startpunkt einer Webseite, wenn ein Alias mehrfach vergeben wurde
  - Vererbung der Eigenschaften der übergeordneten Seiten
- Benutzer authentifizieren
  - Prüfung der Benutzersitzung anhand des Cookies
  - Zusätzlich Prüfung der Zugriffsrechte bei geschützten Seiten

Teil 4: Lebenszyklus einer FE-Anfrage – Seite laden

- Seitenobjekt laden
  - Standard: PageRegular (reguläre Seite)
- Seitenlayout laden
  - Doctype Definition und Meta-Robots-Tags
  - TYPOlight CSS-Framework
  - Dynamische Skripte (CSS-/JavaScript, <head>-Tags)
  - Google Analytics ID
- Module laden
  - Reihenfolge: header, left, main, right, footer, eigene Bereiche
  - Artikelmodul lädt Artikel und Inhaltselemente
  - Hinzufügen von Seitentitel und -beschreibung am Schluss

Teil 4: Lebenszyklus einer FE-Anfrage – Seite ausgeben

- Seite ausgeben
  - Ausgabe erfolgt durch Template::output()
- Template-Objekt übernimmt
  - Hinzufügen der Seite zum Suchindex
  - Anlegen bzw. Aktualisieren der Cache-Version
  - Senden der notwendigen HTTP-Header
  - Aktivieren der GZip-Kompression
  - Ausgabe des Views
  - (vgl. Teil 2: TYPOlight-Framework)

Teil 5: Data Container Arrays

Teil 5: Data Container Arrays – Funktion

- Meta-Daten für Tabellen
  - Ein Data Container Array beschreibt eine Tabelle
  - Generelle Konfiguration, Beziehungen zu anderen Tabellen sowie die Eigenschaften jedes einzelnen Feldes
  - Aufgrund dieser Meta-Daten weiß TYPOlight, wie die Datensätze dargestellt bzw. gespeichert werden sollen
  - Backend-Eingabeformulare werden anhand dieser Daten erstellt
- Einlesen von DCA-Dateien
  - Die DCA-Dateien der aktiven Module werden nacheinander eingelesen (backend, frontend und dann alphabetisch)
  - Jedes Modul kann die bestehende Konfiguration überschreiben
  - Abschließend wird die Datei dcaconfig.php eingelesen, in der lokale Anpassungen updatesicher gespeichert werden können

Teil 5: Data Container Arrays – Aufbau

- Configuration
  - Konfiguration der Tabelle an sich
  - Beziehungen zu anderen Tabellen
  - Versionierung aktivieren
  - Verhalten beim Duplizieren/Löschen von Daten
- Listing
  - Darstellung der Datensätze
  - „List view“, „parent view“ oder „tree view“
  - Standardsortierung der Datensätze
  - Filterkonfiguration (Suchen, Filtern, Sortieren, Limitieren)

Teil 5: Data Container Arrays – Aufbau

- Operations
  - Definieren der Operationen (z.B. bearbeiten oder löschen)
  - Definieren der globalen Operationen (z.B. mehrere bearbeiten)
  - Exakte Zugriffssteuerung mittels Button-Callbacks möglich
- Palettes
  - Ein Palette bezeichnet die Eingabefelder eines Formulars
  - Eingabefelder können gruppiert und ausgerichtet werden
  - Nur freigeschaltete Felder werden angezeigt, daher kann eine Palette je nach Benutzerrechten unterschiedlich aussehen
  - Paletten können sich dynamisch z.B. je nach Modultyp ändern
  - Per Ajax können Formularteile interaktiv nachgeladen werden

Teil 5: Data Container Arrays – Aufbau

- Fields
  - Definiert die einzelnen Felder der Tabelle
  - Input-Typ bestimmt die Art des Formularfeldes
- Evaluation
  - Detaillierte Feldkonfiguration
  - Validierung der Eingaben (z.B. Pflichtfeld oder Datum)
  - Größe des Feldes (z.B. Reihen und Spalten einer Textarea)
  - Aussehen des Feldes (z.B. stlye)
  - Rich Text Editor-Konfiguration
  - Verschlüsselte Speicherung aktivieren
  - …

Teil 5: Data Container Arrays – Callbacks

- onload_callback
  - Gehört zur „Configuration“-Sektion
  - Aufruf bei der Initialisierung des DataContainer-Objektes
  - Ermöglicht z.B. das Prüfen von Zugriffsrechten oder das dynamische Ändern des Data Container Arrays
- onsubmit_callback
  - Gehört zur „Configuration“-Sektion
  - Aufruf beim Absenden eines Backend-Formulars
  - Ermöglicht z.B. das Verändern der Formulardaten bevor diese gespeichert werden (Intervallberechnung im Kalendermodul)

Teil 5: Data Container Arrays – Callbacks

- ondelete_callback
  - Gehört zur „Configuration“-Sektion
  - Aufruf bei der Löschung eines Datensatzes
  - Wird vor dem Entfernen des eigentlichen Datensatzes aus der Datenbank ausgeführt
- paste_button_callback
  - Gehört zur „Listing“-Sektion
  - Erlaubt individuelle Paste-Buttons
  - Wird z.B. in der Seitenstruktur verwendet, um die Buttons je nach Benutzerrechten zu aktivieren bzw. deaktivieren
  - Zusätzliche Prüfung mittels load_callback notwendig, da Eingabe des eigentlichen Befehls in der URL immer noch möglich ist!

Teil 5: Data Container Arrays – Callbacks

- child_record_callback
  - Gehört zur „Listing“-Sektion
  - Darstellung der Kindelemente im „Parent View“
  - Seit Version 2.7 können Kindelemente per Drag & Drop verschoben werden (z.B. Inhaltselemente, FAQ etc.)
  - Individuelle Ausgabe möglich und notwendig
- label_callback
  - Gehört zur „Listing“-Sektion
  - Erlaubt individuelle Bezeichnungen in der Auflistung
  - Wird z.B. in der Benutzerübersicht verwendet, um die Status-Icons hinzuzufügen (Administrator/Benutzer, aktiv/inaktiv)

Teil 5: Data Container Arrays – Callbacks

- button_callback
  - Gehört zur „Operations“-Sektion
  - Erlaubt individuelle Navigationsicons
  - Wird z.B. in der Seitenstruktur verwendet, um die Buttons je nach Benutzerrechten zu aktivieren bzw. deaktivieren
  - Zusätzliche Prüfung mittels load_callback notwendig, da Eingabe des eigentlichen Befehls in der URL immer noch möglich ist!
- options_callback
  - Gehört zur „Fields“-Sektion
  - Erlaubt das Befüllen von Dropdown-Menüs anhand einer individuellen Funktion (beliebige Abfragen möglich)
  - Sinnvoll z.B. bei bedingten foreignKey-Verknüpfungen

Teil 5: Data Container Arrays – Callbacks

- input_field_callback
  - Gehört zur „Fields“-Sektion
  - Erlaubt das Anlegen individueller Felder
  - Wird z.B. im Backendmodul „Persönliche Daten“ verwendet, um das Feld „Daten bereinigen“ zu erstellen
  - Achtung: Feld wird nicht automatisch gespeichert!
- load_callback
  - Gehört zur „Fields“-Sektion
  - Aufruf bei der Initialisierung eines Formularfeldes
  - Kann z.B. zum Laden eines Standardwertes verwendet werden

Teil 5: Data Container Arrays – Callbacks

- save_callback
  - Gehört zur „Fields“-Sektion
  - Aufruf beim Abschicken eines Formularfeldes
  - Kann z.B. für eine individuelle Prüfung eines Eingabewertes oder erweiterte Berechnungen verwendet werden
  - Der Rückgabewert der Callback-Funktion wird gespeichert und sollte daher immer gesetzt sein!

Teil 6: TYPOlight anpassen

Teil 6: TYPOlight anpassen

- Bereits im TYPOlight-Handbuch behandelt
  - Erstellen einer eigenen TinyMCE-Konfigurationsdatei und Einbindung im Data Container Array
  - Anpassen der Übersetzung und updatesicheres Speichern in der Datei system/config/langconfig.php
  - Anpassen des Data Container Arrays und updatesicheres Speichern in der Datei system/config/dcaconfig.php
  - Erstellen einer eigenen Erweiterung, die ein zusätzliches Feld definiert und zu einer Tabelle hinzufügt
- In diesem Workshop
  - Funktion der verschiedenen Hooks
  - Ableiten von Klassen und Überschreiben von Methoden

Teil 6: TYPOlight anpassen – Hooks

- Benutzerregistrierung
  - **createNewUser**: wird ausgelöst wenn sich ein neuer Benutzer im Frontend registriert (Konto kann noch inaktiv sein)
  - **activateAccount**: wird ausgelöst wenn ein neu registriertes Frontend-Benutzerkonto aktiviert wird
  - **setNewPassword**: wird bei einer Passwortänderung ausgelöst
- An- und Abmeldung
  - **checkCredentials**: wird ausgelöst wenn ein Login z.B. aufgrund eines falschen Passwort nicht funktioniert
  - **importUser**: wird ausgelöst wenn ein Benutzerkonto nicht gefunden wurde (Import von LDAP-Server denkbar)
  - **postLogin/postLogout**: wird ausgelöst wenn sich ein Benutzer im Frontend an- bzw. abmeldet

Teil 6: TYPOlight anpassen – Hooks

- Formulare
  - **loadFormField**: wird ausgelöst wenn ein Formularfeld im Formulargenerator geladen wird
  - **validateFormField**: erlaubt das Hinzufügen einer eigenen Validierungsroutine zu einem Formularfeld
  - **addCustomRegexp**: erlaubt das Hinzufügen eines eigenen regulären Ausdrucks zum Widget-Validator
  - **postUpload**: wird ausgelöst nachdem eine Datei über ein Formular hochgeladen wurde
  - **processFormData**: wird ausgelöst nachdem ein Formular im Frontend abgeschickt wurde

Teil 6: TYPOlight anpassen – Hooks

- URL-Generierung
  - **getPageIdFromUrl**: erlaubt das Hinzufügen eigener Logik beim Extrahieren der Seiten-ID aus der URL
  - **generateFrontendUrl**: erlaubt das Hinzufügen eigener Logik beim Generieren von Frontend-URLs
- Templates
  - **parseBackendTemplate**: Parsen eines Backend-Templates
  - **outputBackendTemplate**: Ausgabe eines Backend-Templates
  - **parseFrontendTemplate**: Parsen eines Frontend-Templates
  - **outputFrontendTemplate**: Ausgabe eines Frontend-Templates

Teil 6: TYPOlight anpassen – Hooks

- Verschiedenes
  - **getAllEvents**: erlaubt das Hinzufügen eigener Logik bei der Abfrage von Events in einem Frontend-Modul
  - **getSearchablePages**: erlaubt das Hinzufügen eigener URLs zum Suchindex (URLs sollten gültige Seiten sein)
  - **postDownload**: wird ausgeführt nachdem eine Datei heruntergeladen wurde (z.B. für Download-Statistik)
  - **replaceInsertTags**: erlaubt das Hinzufügen eigener Insert-Tags

Teil 6: TYPOlight anpassen – Klassen ableiten

- Navigationsmodul anpassen
  - Das Navigationsmodul soll so angepasst werden, dass es auch ohne vorhandene Unterseiten erscheint und einen Text ausgibt
  - Dabei soll möglichst die Funktionalität der Originalklasse erhalten bleiben, damit Updates keine zusätzliche Arbeit verursachen
- Eigene Erweiterung anlegen
  - Modulordner „xcustom“ erstellen (wird als letztes geladen)
  - Darin wird die Datei ModuleMyNavigation.php gespeichert
  - In dieser Datei wird die Klasse ModuleMyNavigation definiert
  - Die Klasse ModuleMyNavigation erbt von ModuleNavigation
  - Nur die Methode generate() wird überschrieben

Teil 6: TYPOlight anpassen – Klassen ableiten

- class ModuleMyNavigation

```php
<?php

class ModuleMyNavigation extends ModuleNavigation
{
    public function generate()
    {
        // Aufruf der Originalmethode
        $buffer = parent::generate();

        if ($buffer == '')
        {
            $buffer = 'Es sind keine Unterseiten vorhanden';
        }

        return $buffer;
    }
}

?>
```

Teil 6: TYPOlight anpassen – Klassen ableiten

- Neue Klasse registrieren
  - Die neue Klasse muss TYPOlight bekannt gemacht werden
  - Überschreiben des globale Konfigurationsarrays FE_MOD in der Datei system/modules/xcustom/config/config.php
  - ```
    $GLOBALS['TL_CONFIG']['FE_MOD']['navigationMenu']
    ['navigation'] = 'ModuleMyNavigation';
    ```
- Dynamische Konfiguration
  - Dank der dynamischen Konfiguration wird beim nächsten Aufruf von TYPOlight automatisch die neue Klasse registriert
  - Das Navigationsmodul gibt nun den Hinweis „Es sind keine Unterseiten vorhanden“ aus anstatt gar nicht zu erscheinen
  - Die Anpassung ist updatesicher und damit pflegeleicht
    ```