Wordpress OOP Klassen Definitionen und Beispiele

Kevin Pfeifer – Version 1.0, 2019-03-19

Table of Contents

Repo: https://bitbucket.org/sunlimeitservices/wordpress/src/2019_git_local_setup_oop/ (https://bitbucket.org/sunlimeitservices/wordpress/src/2019_git_local_setup_oop/)

Verwendete lokale Software-Versionen

OS: **MacOS Mojave 10.14.3** | MySQL: **5.7.23** | Apache: **2.4.34** | PHP: **7.1.23**

Wieso?

Beim entwickeln eines Custom-Themes in Wordpress werden folgende Stukturen und Elemente sehr oft verwendet:

- Custom Post-Types (CPTs)
- Custom Taxonomien
- Advanced Custom Field Groups (ACF)
- VC Elemente

Die Erstellung solcher Strukturen und Elemente benötigt immer die gleichen Code-Snippets, was zu einer großen Anzahl an duplicate Code führt. Des weiteren sind die Datenstrukturen - welche diese Elemente darstellen sollen - meist so komplex, dass hier große Teile immer wieder kopiert werden müssen um z.b. neue Felder oder Elemente zu

erstellen.

Daher haben wir uns dafür entschieden hier einige Klassen zur Verfügung zu stellen, um die Anzahl des duplicate Code zu verringern und damit die Stukur und Übersichtlichkeit des Codes zu verbessern.

Custom Post-Types (CPTs)

Um einen CPT zu erstellen muss als erstes die Klasse "CustomPostType" in der functions.php des Themes includiert werden.

```php
require get_template_directory() . '/inc/classes/CustomPostType.php';
```

Danach kann ein Objekt dieser Klasse erstellt werden:

```php
$productions = new CustomPostType( 'Produktionen', 'Produktion', 'Produktionen', 'production' );
```

Die Reihenfolge der Parameter ist wie folgt definiert:

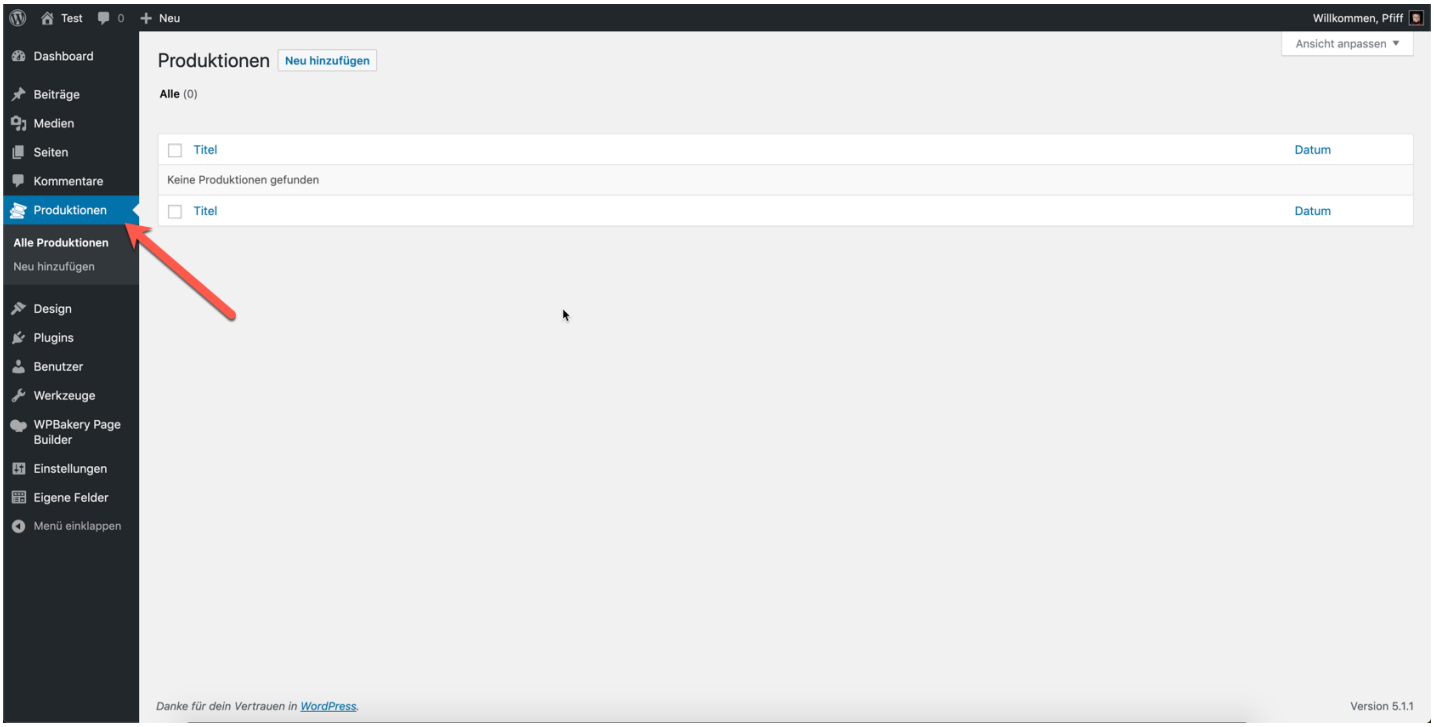
Menü-Name | Singular-Name | Plural-Name | Slug

- Menü-Name: Text, wie der CPT im Menü angezeigt werden soll
- Singular-Name: Text, wie ein einzelnes Element dieses CPTs benannt werden soll
- Plural-Name: Text, wie mehrere Elemente dieses CPTs benannt werden sollen

Falls noch weitere Einstellungen beim CPT getroffen werden müssen kann ein Array als letzten Parameter noch hinzugefügt werden:

```php
$production_args = array(
  'menu_icon' => 'dashicons-tickets-alt',
  'supports'  => array( 'title', 'editor', 'thumbnail' )
);
$productions = new CustomPostType( 'Produktionen', 'Produktion', 'Produktionen', 'production', production_args
);
```

Dadurch erhalten wir im Backend folgenden CPT :



Custom Taxonomien

Gleich wie beim CPT gibt es auch hier eine Klasse, die wir zuerst einbinden müssen:

```php
require get_template_directory() . '/inc/classes/CustomTaxonomy.php';
```

Danach kann ein Objekt dieser Klasse erstellt werden:

```php
$stage = new CustomTaxonomy( 'Bühnen', 'Bühnen', 'Bühne', 'stage', 'production' );
```

Die Reihenfolge der Parameter ist wie folgt definiert:

Menü-Name | Plural-Name | Singular-Name | Slug | Custom-Post-Type

- Menü-Name: Text, wie die Taxonomy im Menü angezeigt werden soll
- Plural-Name: Text, wie mehrere Terms dieser Taxonomy benannt werden sollen
- Singular-Name: Text, wie ein Term dieser Taxonomy benannt werden soll
- Slug: Text, wie der Slug definiert sein soll (= URL-Paramter)
- Custom-Post-Type: Slug des CPTs, bei welchem diese Taxonomy angewandt werden soll

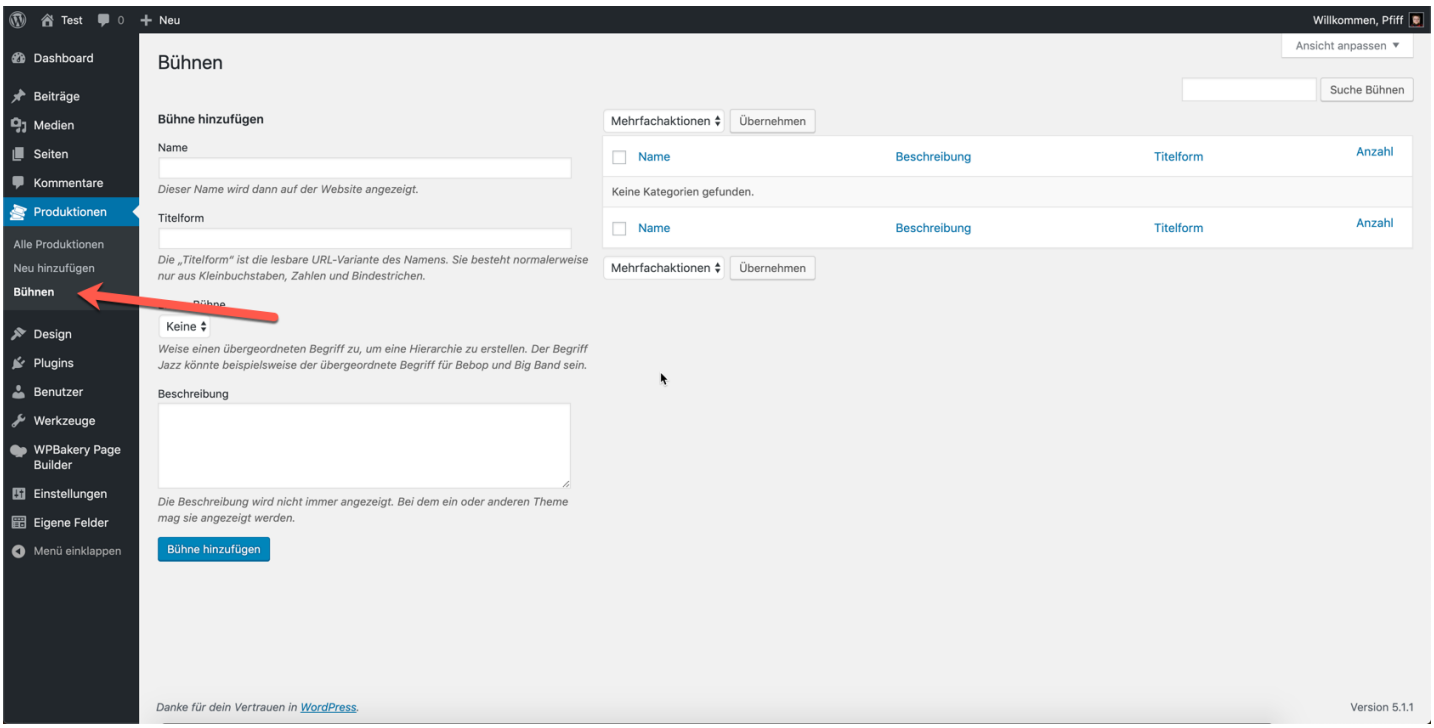
Gleich wie bei einem CPT können über einen weiteren Parameter die Standard-Einstellungen der Taxonomie angepasst werden:

```php
$stage_args = array(
  'hierarchical'          => false,
  'public'                => true,
  'show_in_nav_menus'     => false,
);
$stage = new CustomTaxonomy( 'Bühnen', 'Bühnen', 'Bühne', 'stage', 'production', $stage_args );
```

Die Standard-Labels können ebenso über einenen weiteren Paramter angepasst werden:

```php
$stage_args = array(
  'hierarchical'          => false,
  'public'                => true,
  'show_in_nav_menus'     => false,
);
$stage_labels = array(
  'name'                  => __( 'Name', 'test' ),
  'singular_name'         => __( 'Singular Name',  'test' ),
);
$stage = new CustomTaxonomy( 'Bühnen', 'Bühnen', 'Bühne', 'stage', 'production', $stage_args, $stage_labels );
```

Dadurch erhalten wir im Backend folgende Taxonomy:

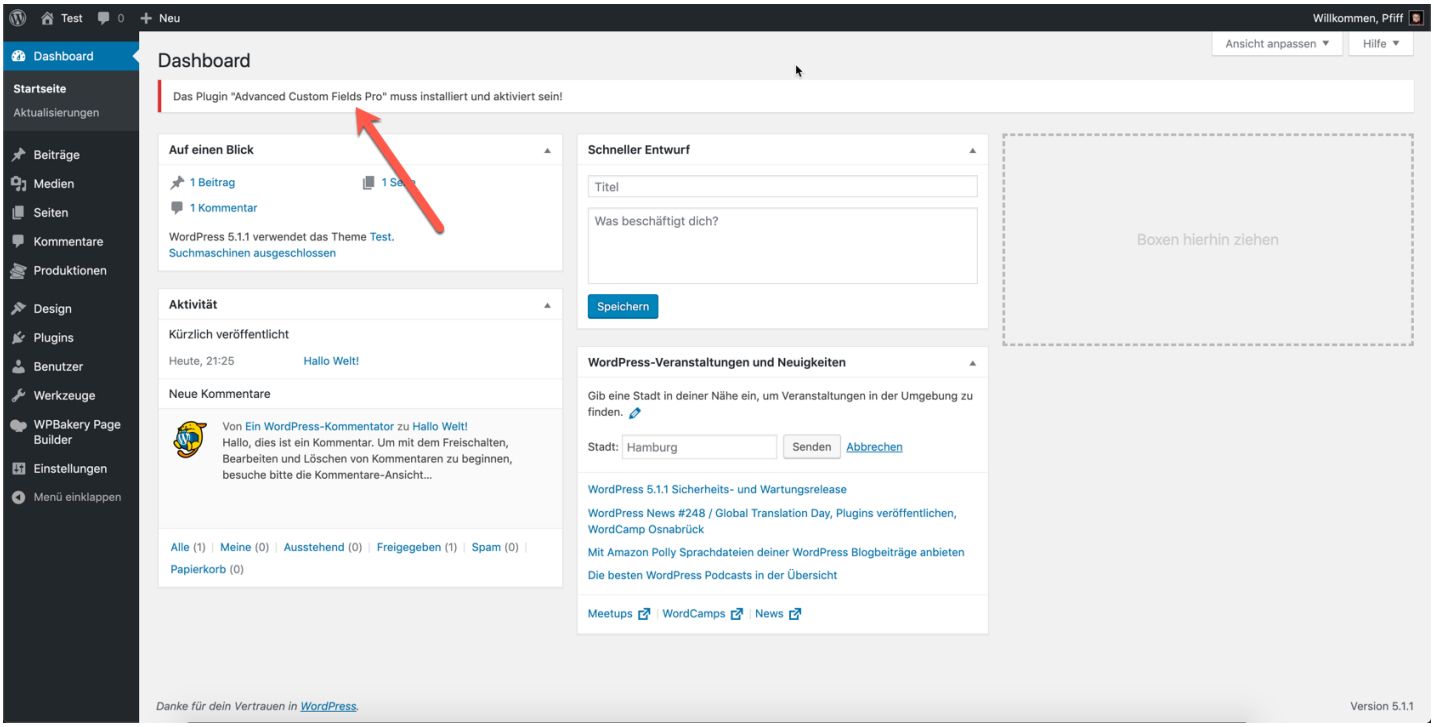


Advanced Custom Field Groups (ACF)

Um eine neue ACF Feldgruppe zu erstellen muss als erstes die Klasse "AcfFieldGroup" in der functions.php des Themes includiert werden.

```php
require get_template_directory() . '/inc/classes/AcfFieldGroup.php';
```

Dies allein führt bei einem blanken Wordpress schon dazu, dass folgende Meldung im Backend erscheint:



Als nächstes sollte die aktuellste Version vom ACF Pro Plugin installiert und aktiviert werden. Damit verschwindet auch diese Meldung.

Um eine ACF Field Group einem Post-Type o.Ä. hinzuzufügen benötigt es 2 Schritte:

- Erstellung des ACF Field Group Objektes
- Hinzufügen der benötigten Felder zu dem vorhin erstellten Objekt

Erstellung des ACF Field Group Objektes

PHP

```php
$productions_fieldgroup = new AcfFieldGroup('prod_', 'Produktion', 'post_type', 'production');
```

Dieser Befehlt definiert die allgemeine Feld-Gruppe, welche die unterschieldichen Felder beinhaltet und wo diese Felder angewendet werden sollen.

Die Reihenfolge der Parameter ist wie folgt definiert:

Gruppen-Slug | Gruppen-Titel | Location Param | Location Value

- Gruppen-Slug: Text, der zur Unterscheidung mehrere Feld-Gruppen dienen soll.
  - Best Practise: Kleingeschrieben, ohne Sonderzeichen und endet mit einem _
- Gruppen-Titel: Text, der bei der Feldgruppe am Anfang angezeigt werden soll
- Location Param: Text, auf welchem Typ von Element es angewandt werden soll.
  - Z.b. post_type | options_page | current_user | nav_menu_item
- Location Value: Text, auf welchen Typ von dem vorhin definierten Location Param die Feld-Gruppe angewandt werden soll. Kann auch ein Array sein damit z.b. eine Feldgruppe mehreren CPTs zugewiesen wird (immer ODER-Verbindung)
  - Z.b. production | theme-options | all | array('production', 'production_2')

Hinzufügen der benötigten Felder zu dem vorhin erstellten Objekt

Damit haben wir einmal unser Feld-Gruppen Objekt erstellt, jedoch müssen wir nun erst Felder zu dieser Feld-Gruppe hinzufügen. Dies geschieht wie folgt:

```php
$field_key = $productions_fieldgroup->addField('text', 'Test', 'test');
```

Die Reihenfolge der Parameter ist wie folgt definiert:

Typ | Title | Name

- Typ: Welcher Feld-Typ hinzugefügt werden soll
- Titel: Wie das Feld benannt sein soll
- Name: über welchen Namen dieses Feld abrufbar sein soll.

> Der Name des Feldes wird mit dem Gruppen-Slug des Feld-Gruppen-Objektes prefixed.

D.h. am obrigen Beispiel mit dem Feld-Namen "test" und dem Gruppen-Slug "prod_" können die Daten in diesem Feld über "prod_test" abgerufen werden.

Die Reihenfolge der einzelnen `$productions_fieldgroup->addField()` Funktionen definiert auch 1:1 die Reihenfolge wie diese dann beim CPT oder wo auch immer die Feldgruppe ausgegeben wird.

Hinzufügen von Sub-Fields

Es gibt ein paar Feld-Typen, die Sub-Fields zur Verfügung stellen.

Beispiele:

- Wiederholung
- Flexibler Inhalt

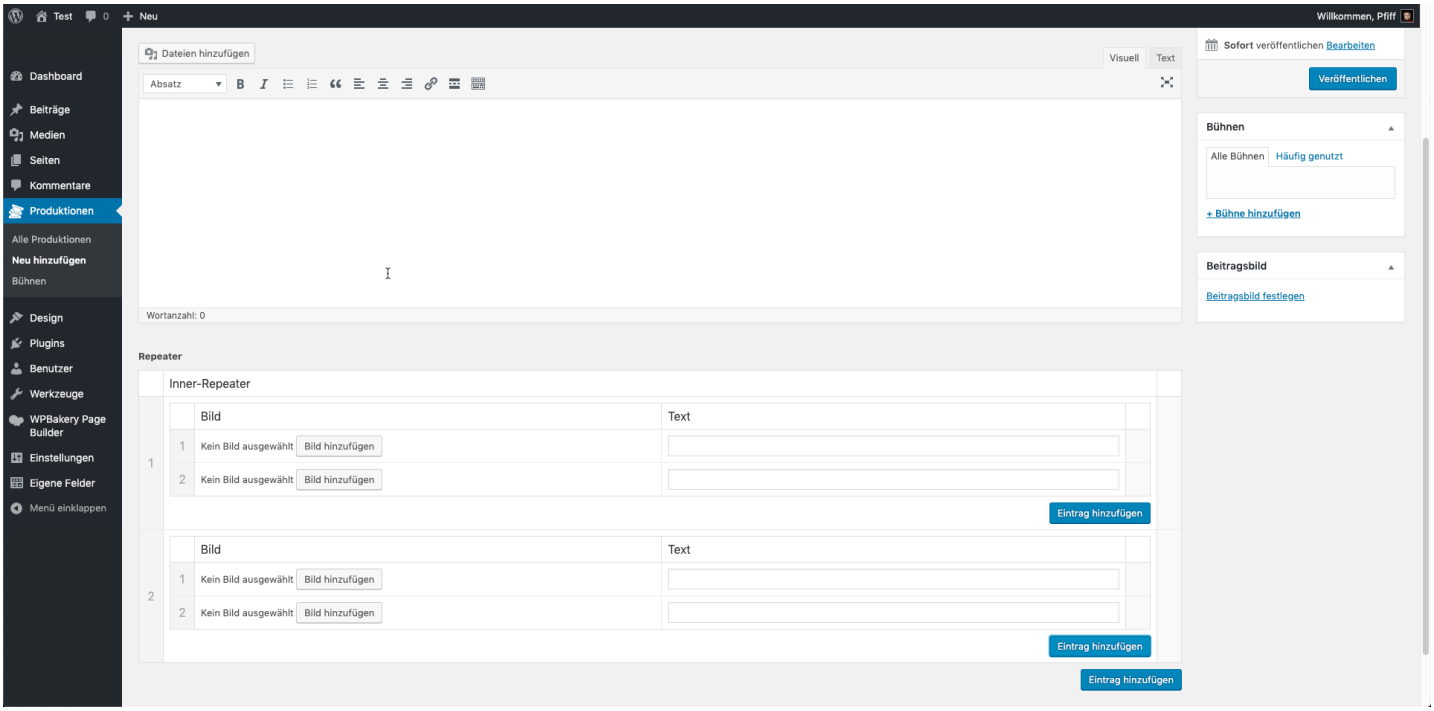
Diese müssen wie folgt hinzugefügt werden:

```php
$repeater_key = $productions_fieldgroup->addField('repeater', 'Repeater', 'repeater');
$inner_repeater_key = $productions_fieldgroup->addSubfield($repeater_key, 'repeater', 'Inner-Repeater',
'inner-repeater');
$image_key = $productions_fieldgroup->addSubfield($inner_repeater_key, 'image', 'Bild', 'inner-image');
$text_key = $productions_fieldgroup->addSubfield($inner_repeater_key, 'text', 'Text', 'inner-text');
```

In diesem Beispiel haben wir folgende Feld-Strukur:

- Wiederholung
  - Wiederholung
    - Bild
    - Text

Im Backend sieht das wie folgt aus:



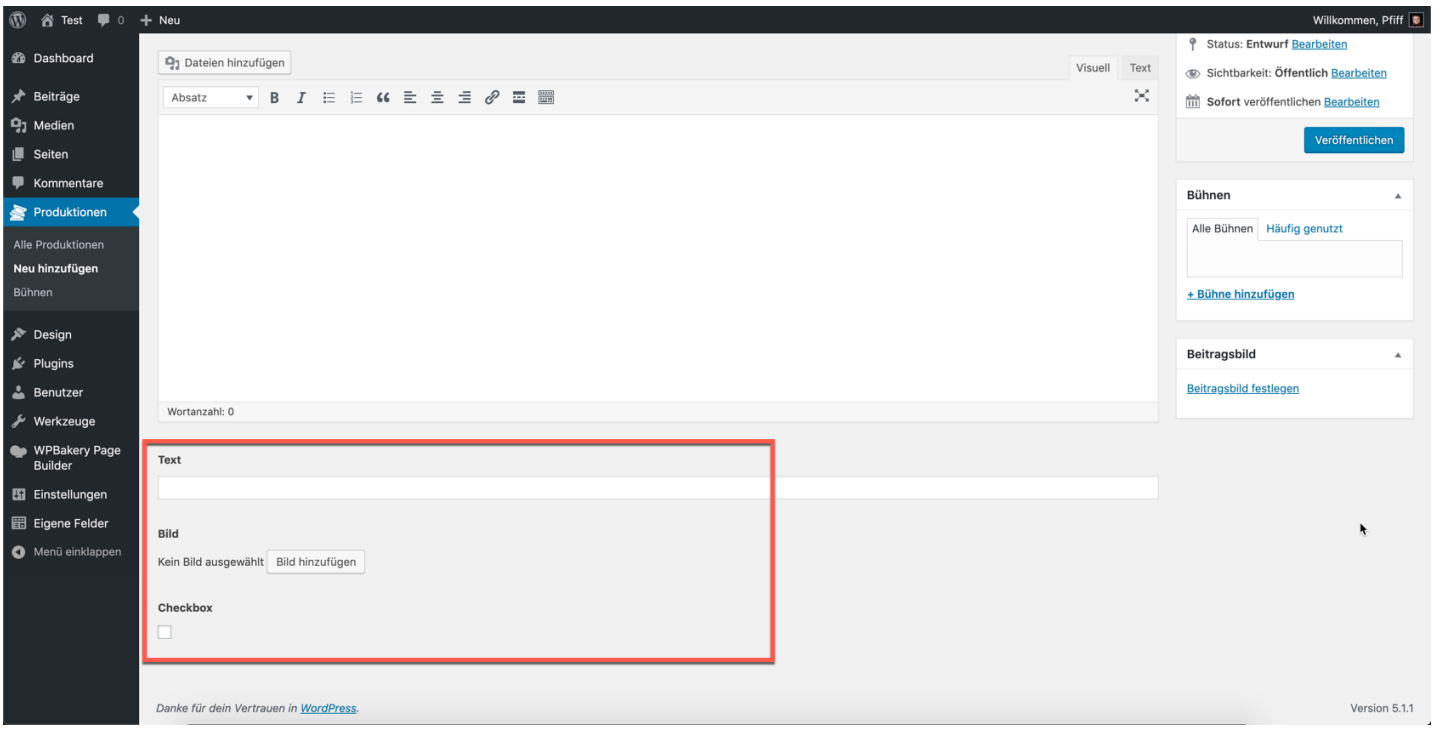
Zusammenfassend

Wenn wir für den CPT "Produktion" 3 Felder hinzufügen wollen muss folgender Code geschrieben werden:

PHP

```php
$productions_fieldgroup = new AcfFieldGroup('prod_', 'Produktion', 'post_type', 'production');
$productions_fieldgroup->addField('text', 'Text', 'text');
$productions_fieldgroup->addField('image', 'Bild', 'image');
$productions_fieldgroup->addField('true_false', 'Checkbox', 'checkbox');
```

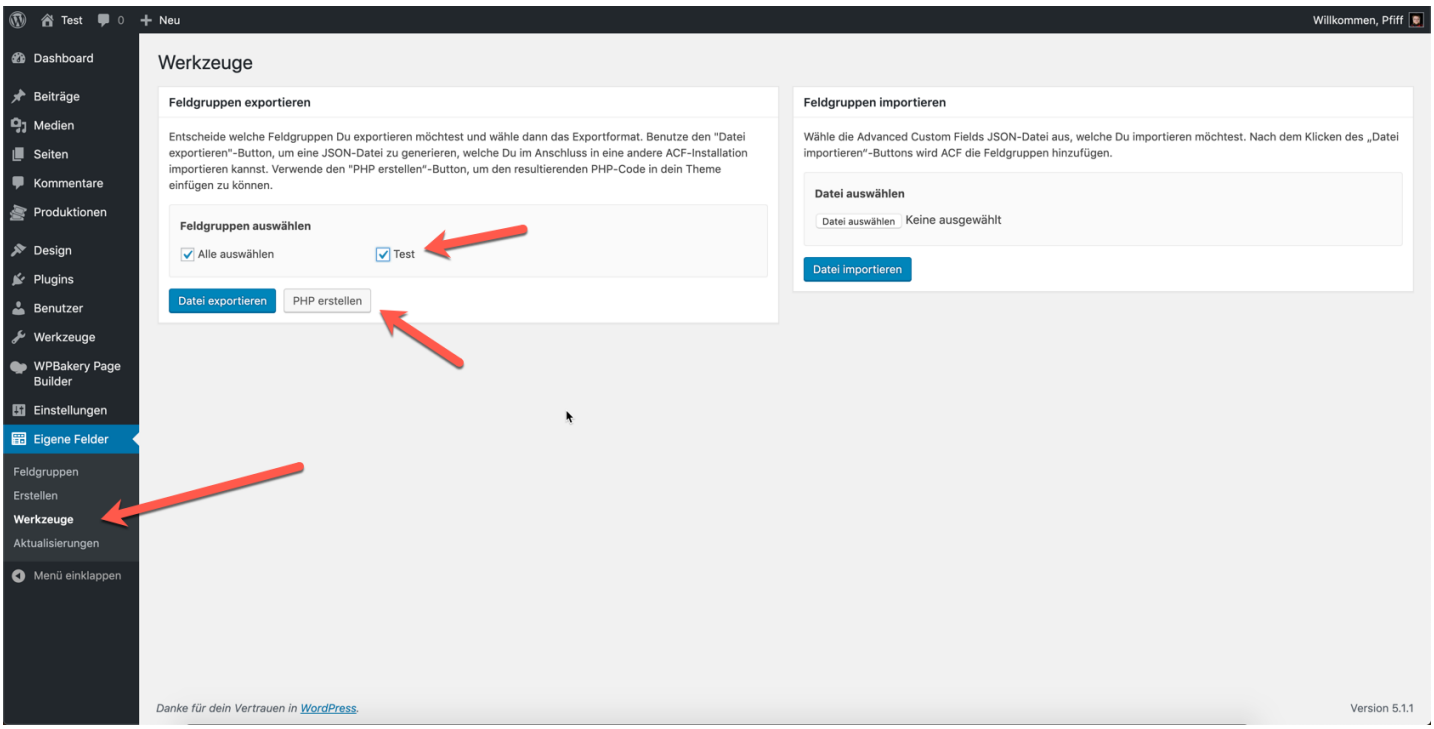
Dies erzeugt folgende Felder im Backend:



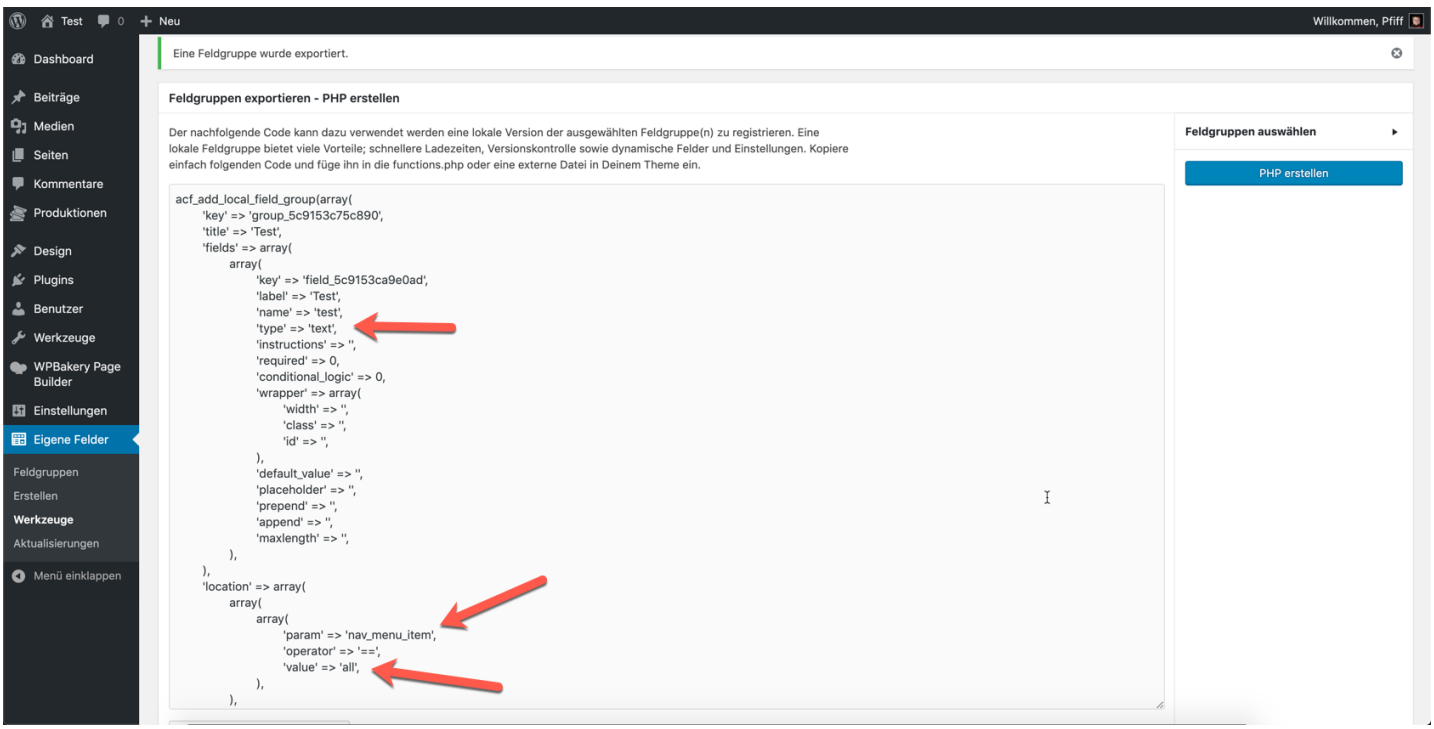
Wie finde ich herause welche Feld-Parameter ich für ein spezielles Feld brauche?

Am einfachsten ist es das benötigte Feld bzw. die Struktur in einer Feld-Gruppe im Backend wie gewohnt zu erstellen.

Danach kann unter `Eigene Felder` ⇒ `Werkzeuge` ⇒ `Feldgruppen exportieren` der PHP-Code für eine im Backend definierte Feldgruppe exportiert werden.



Der dort angezeigte PHP-Code bietet alle wichtigen Informationen was für die jeweiligen Felder-Typen eingegeben werden muss.

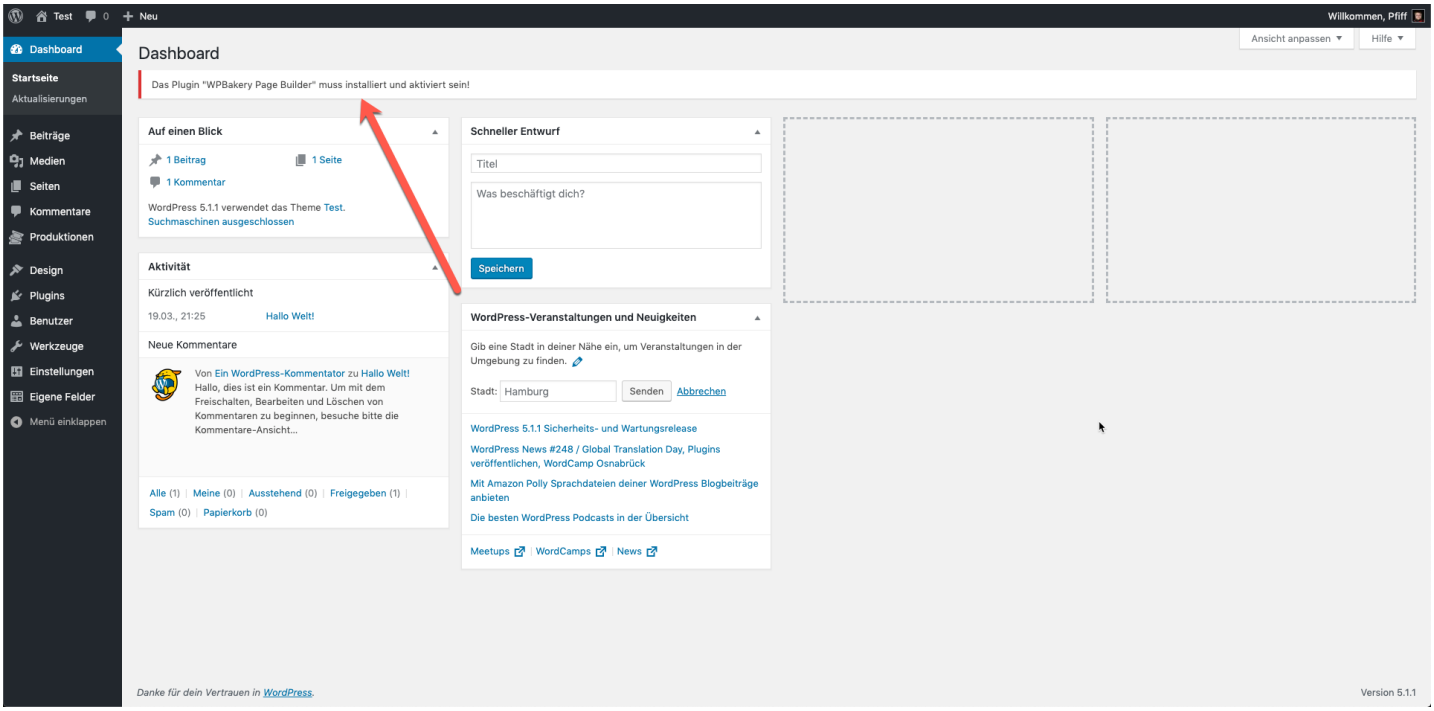


VC Elemente

Um ein neues VC Element zu erstellen muss als erstes die Klasse "VCElement" in der functions.php des Themes includiert werden.

```php
require get_template_directory() . '/inc/classes/VCElement.php';
```

Dies allein führt bei einem blanken Wordpress schon dazu, dass folgende Meldung im Backend erscheint:



Als nächstes sollte die aktuellste Version vom WPBakery Page Builder Plugin installiert und aktiviert werden. Damit verschwindet auch diese Meldung.

Um ein VC-Element hinzuzufügen benötigt es 2 Schritte:

- Erstellung des VC-Element Objektes
- Hinzufügen der benötigten Felder/Parameter zu dem vorhin erstellten Objekt

Erstellung des VC-Element Objektes

PHP

```php
$vc_element = new VCElement('Test-Name', 'test_base', 'Test-Kategore', array(), true);
```

Name | Base | Kategorie | Args | VC-Element in späteren Hook implementieren

- Name: Wie das VC Element heißen soll
- Base: Welche Base der Shortcode haben soll
- Category: In welcher Kategorie das VC Element erstellt werden soll
- Args (Optional): Weitere Paremter, die das VC-Element detailierter definieren
- Hook-Reihenfolge (Optional): true, wenn das VC-Element erst in "after_setup_theme" statt in "vc_before_init" integriert werden soll (Default: false)

Hinzufügen der benötigten Felder/Parameter zu dem vorhin erstellten Objekt

PHP

```php
$vc_element->addParam('textfield', 'Test-Feld', 'testfield');
```

Feld-Type | Feld-Titel | Parameter-Name

- Feld-Type: Welche Art von Feld hinzugefügt werden soll
- Feld-Titel: Wie das Feld bezeichnet werden soll

- Parameter-Name: Unter welchem Namen die Daten in dem Feld aufrufbar sein sollen

Hinzufügen von Sub-Parameter

Es gibt einen Param-Typen, der Sub-Parameter zur Verfügung stellen - param_group (=Repeater-Feld)

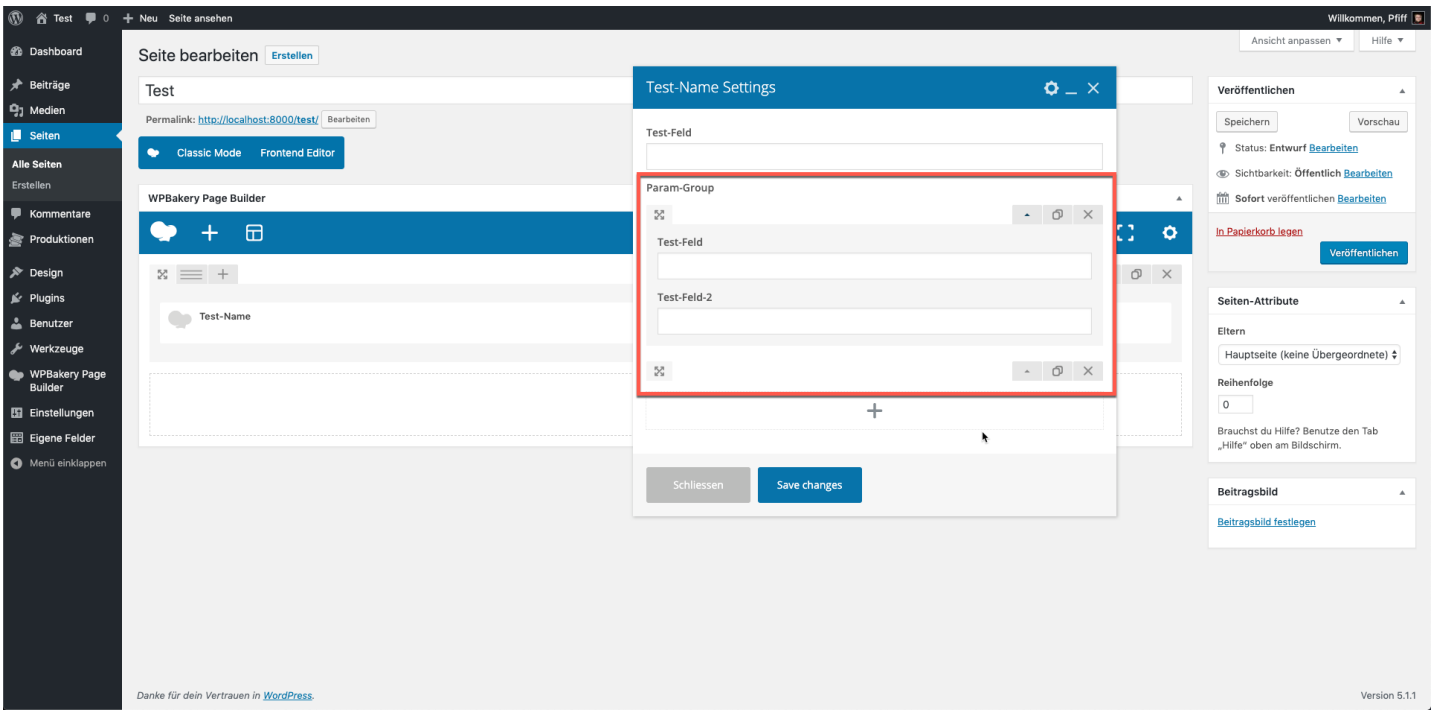
Diese müssen wie folgt hinzugefügt werden:

```php
$vc_element = new VCElement('Test-Name', 'test_base', 'Test-Kategore');
$vc_element->addParam('textfield', 'Test-Feld', 'testfield');
$vc_element->addParam('param_group', 'Param-Group', 'param_group_name');
$vc_element->addSubParam('param_group_name', 'textfield', 'Test-Feld', 'test_subfield');
$vc_element->addSubParam('param_group_name', 'textfield', 'Test-Feld-2', 'test_subfield_2');
```

In diesem Beispiel haben wir folgende Feld-Strukur:

- Wiederholung
  - Text
  - Text

Im Backend sieht das wie folgt aus:

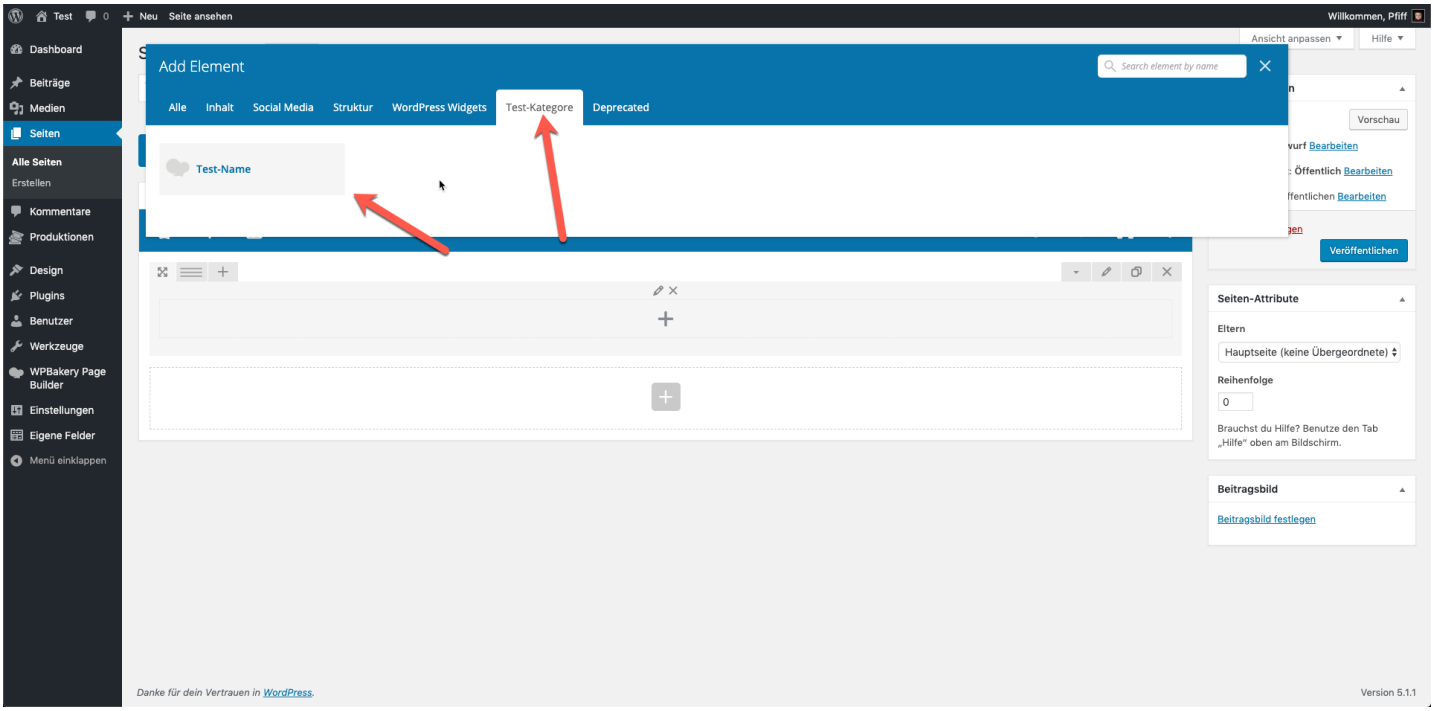


Zusammenfassend

Wenn wir ein VCElement "Test-Name" mit 1 Feld hinzufügen wollen muss folgender Code geschrieben werden:

```php
$vc_element = new VCElement('Test-Name', 'test_base', 'Test-Kategore');
$vc_element->addParam('textfield', 'Test-Feld', 'testfield');
```

Dies erzeugt folgendes VCElement im Backend:



Vorhandene VC-Elemente editieren

Erster Parameter ist die Base von dem vorhandenen VC Element, das editiert werden soll.

Restliche Parameter sind gleich wie bei addParam();

PHP

```
VCElement::addParamToExistingElem('vc_row', 'dropdown', 'Inhaltsbreite', 'contentwidth', array(
  'weight' => '1',
  'value' => array(
    __('Boxed', 'kirchdorfer') => 'boxed',
    __('Boxed test', 'kirchdorfer') => 'boxed_test',
    __('Boxed groß', 'kirchdorfer') => 'boxed_large',
    __('Boxed klein', 'kirchdorfer') => 'boxed_small',
    __('Volle Breite', 'kirchdorfer') => 'fullwidth'
  ),
  'description' => 'Breite des Inhaltsbereiches.',
));
```

PHP

```
VCElement::removeParamFromExistingElem('vc_row', 'contentwidth');
```

PHP

```
VCElement::removeExistingElement('test_base');
```

Custom Taxonomies in VC-Element verwenden

Durch die Art wie Custom Taxonomies erstellt werden ist es nicht so trivial diese in einem VC-Element z.b. in einem Dropdown zu verwenden.
Wir müssen das initialisieren des VC-Elements innerhalb eines Hooks machen, da wir in der functions.php sonst zu früh wären und keine Terms aus der Custom Taxonomy auslesen könnten.

Beispiel:

PHP

```php
function proj_custom_vc_element(){
  $custom_element = new VCElement('Speakers', 'proj_speakers', 'Project', array(), false);

  $tax_data = array(
    __('All','isnvh') => 'all'
  );

  $terms = get_terms(array('taxonomy' => 'custom-category'));
  if(!empty($terms) && !is_wp_error($terms)):
    foreach($terms as $term):
      $tax_data[$term->name] = $term->term_id;
    endforeach;
  endif;

  $custom_element->addParam('dropdown', 'Category', 'category', array(
    'value' => $tax_data,
  ));
}
add_action('init', 'proj_custom_vc_element');
```

Theme-Übersetzungen

Voraussetzung: PoEdit (https://poedit.net/download)

Unterschied .pot / .po /.mo

.pot-Dateien sind Template-Files, aus dem Übersetzungen für die jeweiligen Sprachen erstellt werden können.

.po-Dateien sind die jeweiligen Übersetzungs-Files, die von der ursprünglichen Sprache des Themes in die gewünschte Sprache übersetzen. Beispiel hier: de_DE.po oder en_EN.po

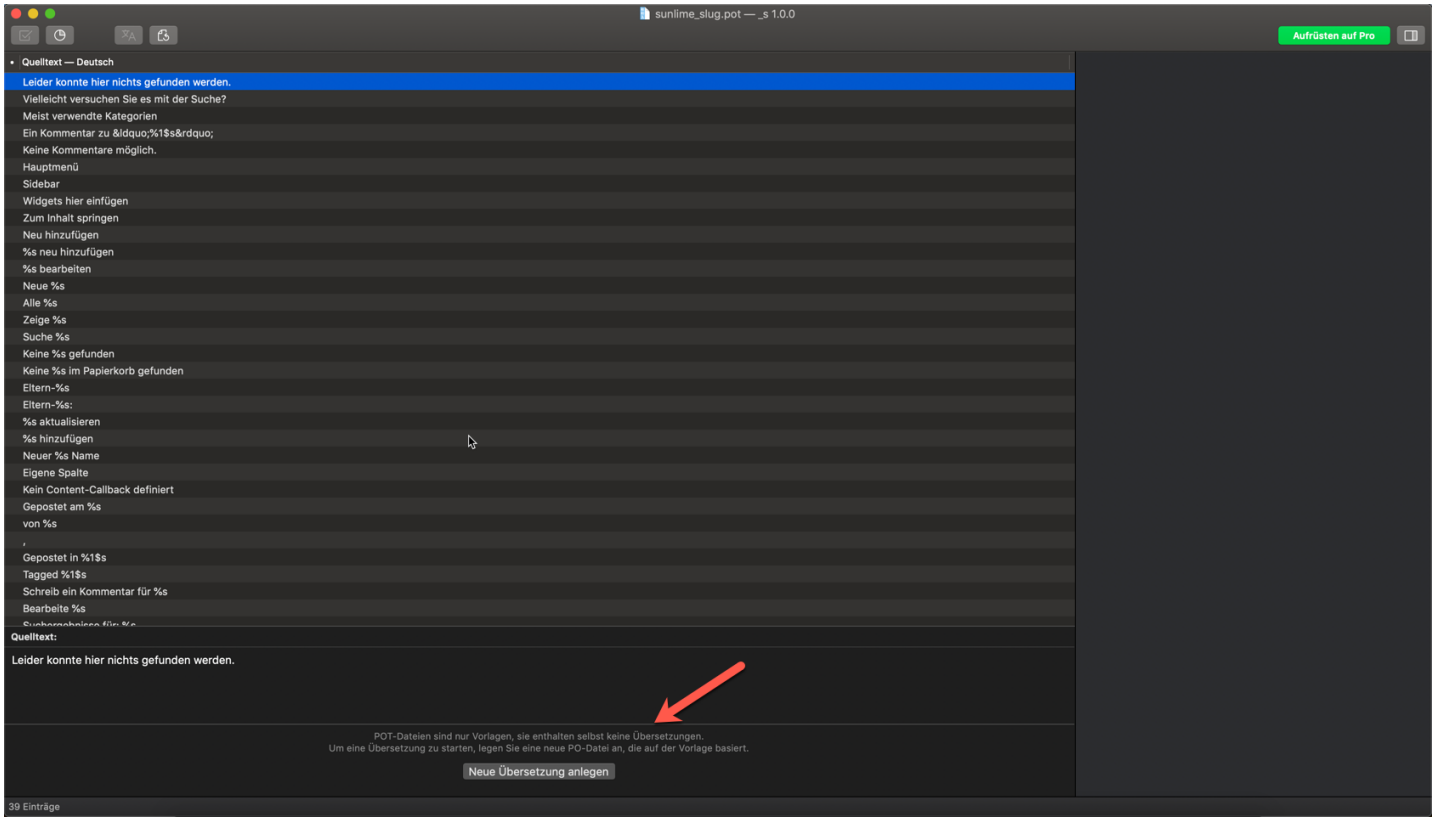
.mo-Dateien werden beim speichern einer .po Datei automatisch erstellt. Diese sind die "kompilierten" Sprach-Files die vom jeweiligen Programm (bei uns Wordpress) ausgelesen werden.

.po Datei erstellen

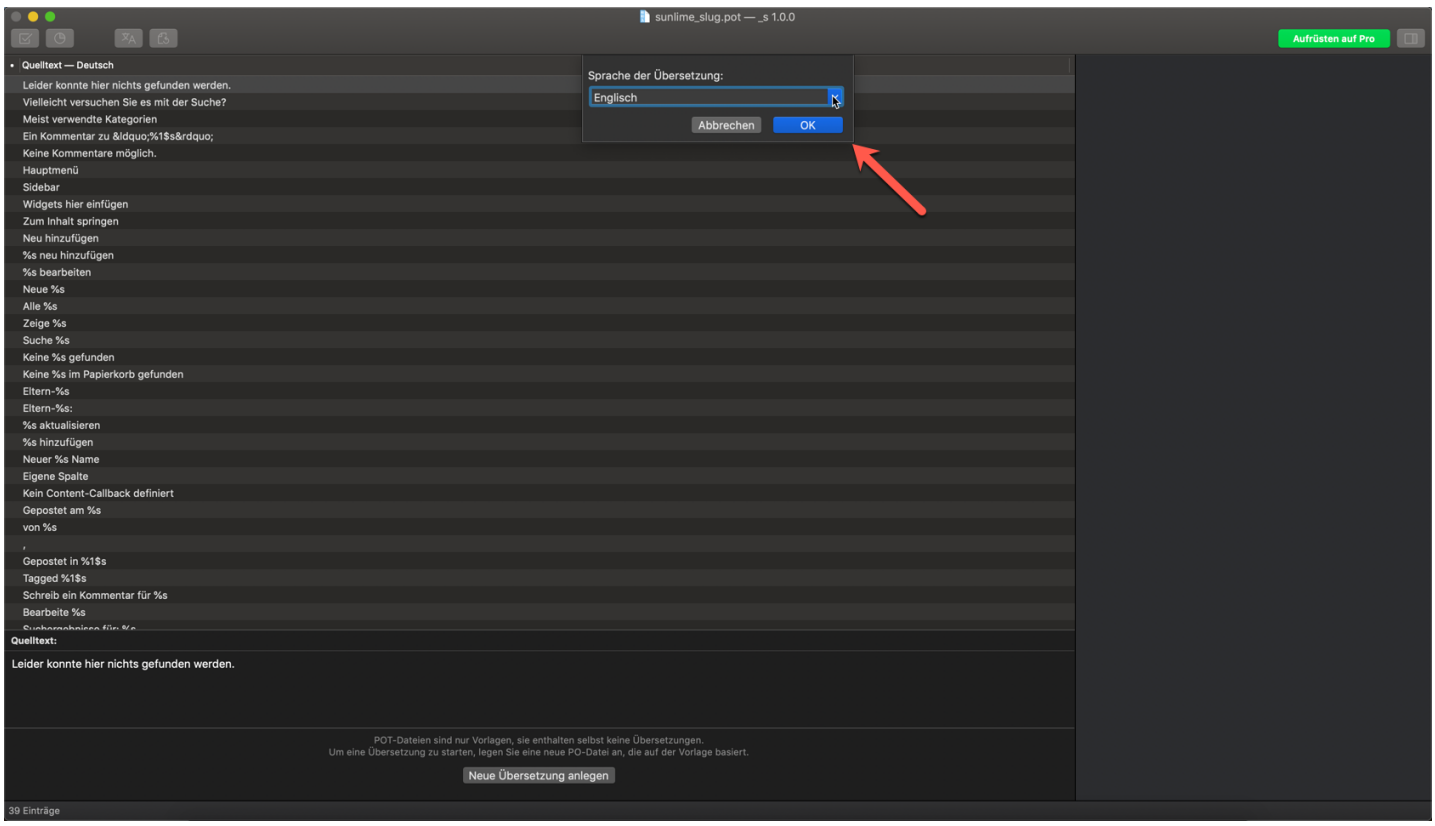
Als Basis für die Theme-Übersetzung gilt immer die .pot Datei im theme/languages Ordner.

Wenn PoEdit installiert ist werden .po und .pot Dateien automatisch mit diesem Programm verbunden.

Wenn z.b. eine .pot Datei mit PoEdit geöffnet wird, erscheint unten im PoEdit ein Bereich mit "Neue Übersetzung anlegen".



Bei Klick auf diesen Button wird nachgefragt in welche Sprache übersetzt werden soll.



Hier einfach die gewünschte Sprache auswählen, in der die Theme-Strings übersetzt werden sollen.

In unserem Beispiel übersetzen wir die Theme-Strings auf Englisch da sie schon auf Deutsch vorhanden sind.

Nachdem auf "OK" geklickt wurde kann die Übersetzung der Theme-Strings begonnen werden.

Wichtig hierbei ist aber beim speichern der Übersetzungen dass die nun .po-Datei wie folgt benannt wird:

`en_EN.po`

Diese Datei muss im /languages Ordner liegen (oder wie die Text-Domain in der functions.php definiert wurde)

Ohne diese Benennung wird die .mo Datei zwar generiert, Worpdress kann diese aber nicht richtig auslesen.

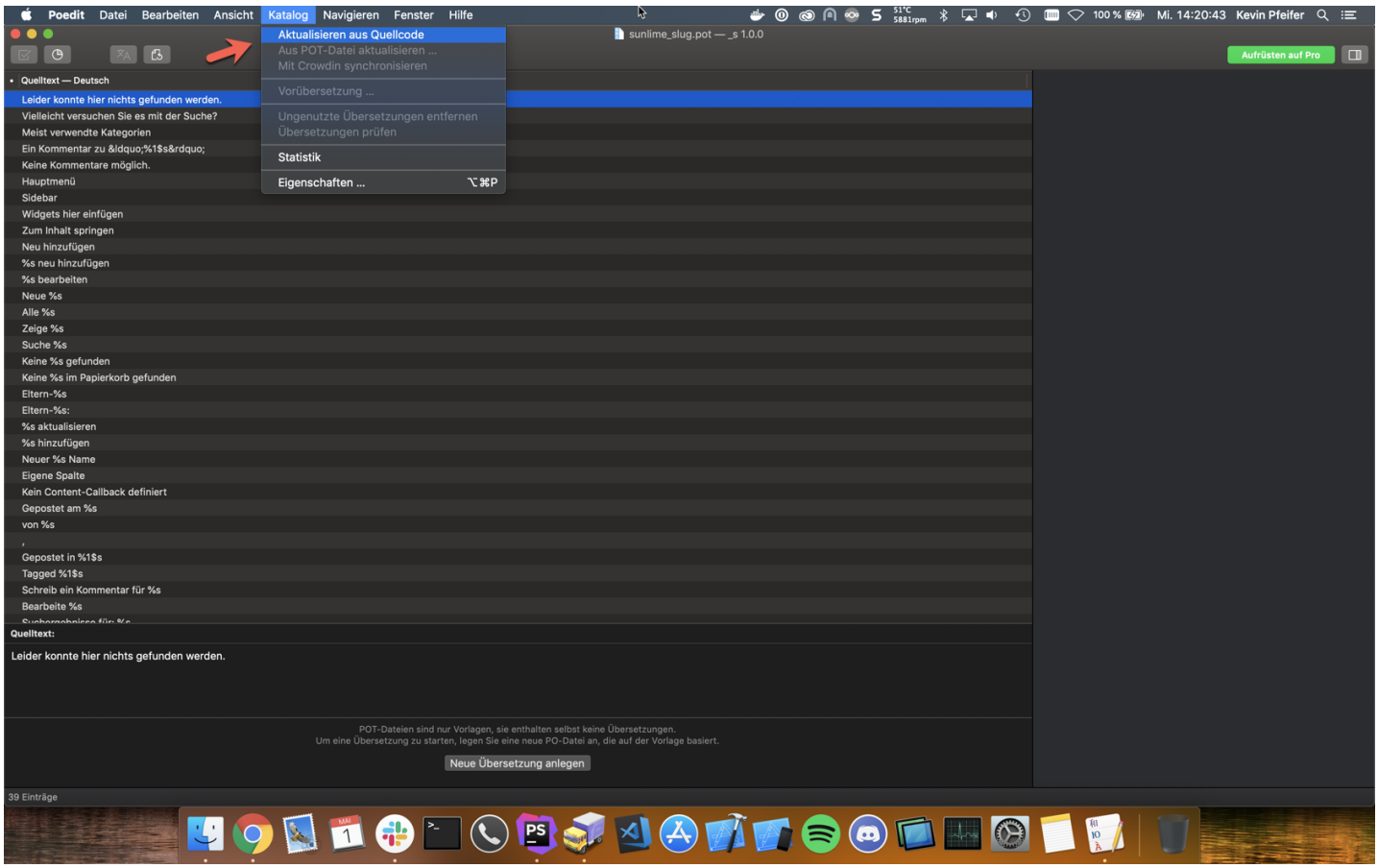
Bei jedem Speichern einer Übersetzung wird die .mo Datei neu generiert.

.po/.pot mit neuen Theme-Strings aktualisieren

Bei einer Custom-Theme Entwicklung ist es normal, dass vorhandene Theme-Strings geändert und neue Theme-Strings durch Elemente hinzugefügt werden.

Damit diese in die .pot bzw. schon vorhandenen .po Datei hinzugefügt werden muss folgendes im PoEdit durchgeführt werden:

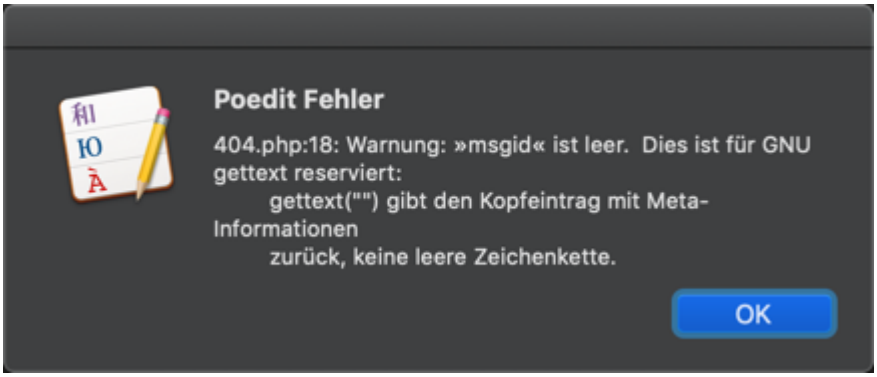
`Katalog ⇒ Aktualisieren aus Quellcode`



Hiermit durchsucht PoEdit den ganzen Source-Code im "theme" Ordner rekursiv nach neuen Übersetzungs-Strings und aktuallisiert die Datei.

Es kommt zu einem Fehler beim aktualisieren der Theme-Strings (msgid is leer)

Es kann sein, dass beim aktualisieren der Theme-Strings ein PoEdit-Fehler erscheint:



Grund dahinter ist, dass im Theme eine leerer String übersetzt werden soll, was an sich ja unlogisch ist.

In diesem Beispiel wurde der Fehler in der 404.php Zeile 18 durch folgende Funktion verursacht:

```
_e( '', 'sunlime_slug' );
```

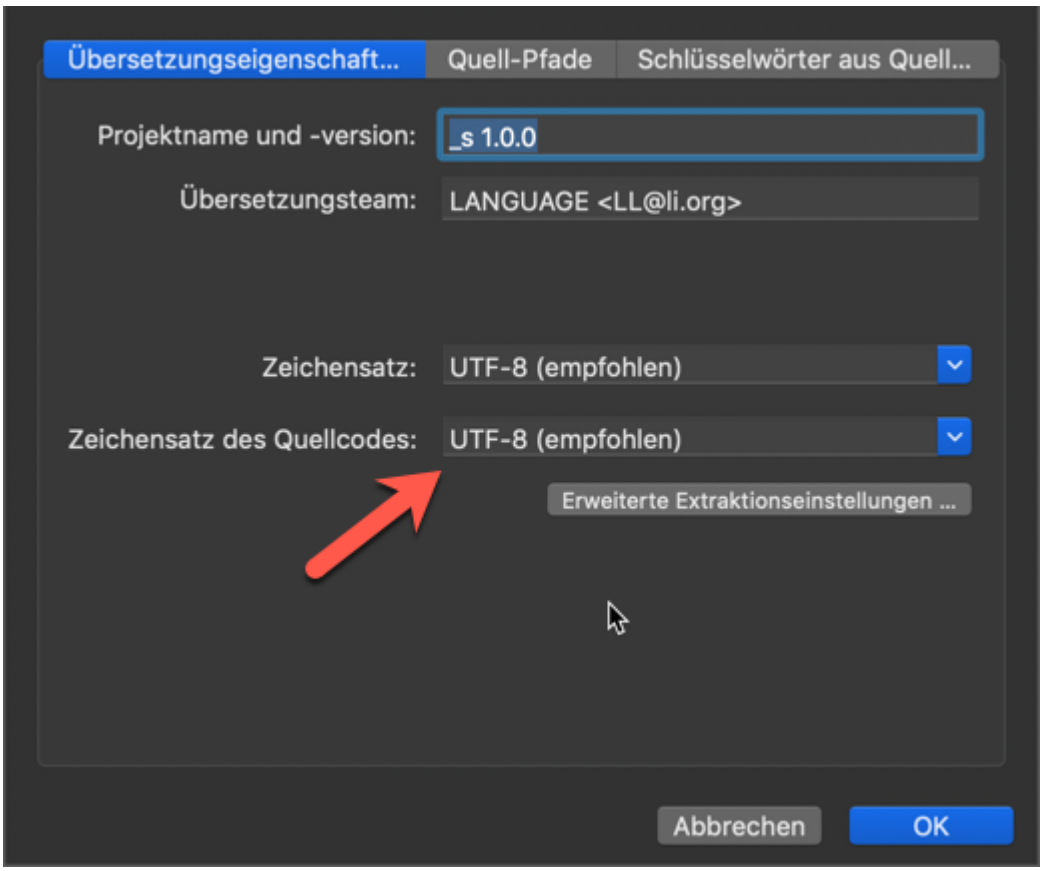
Hier einfach die Übersetzungsfunktion entfernen und die .po/.pot Datei aktualisieren, dann sollte kein Problem mehr erscheinen.

Woher weiß PoEdit was Übersetzungs-Funktionen sind bzw. es fehlen gewisse Theme-Strings in der .pot/.po Datei

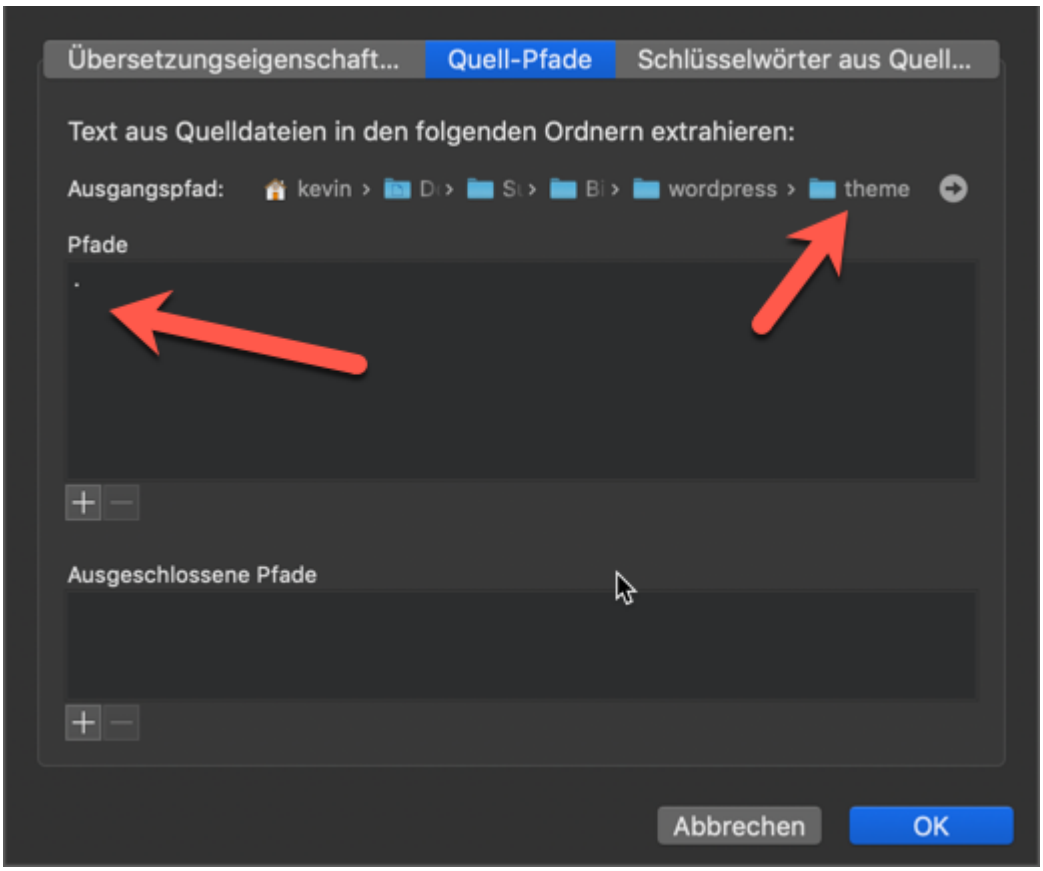
Im PoEdit kann man unter `Katalog` ⇒ `Eigenschaften` einige Einstellungen vornehmen.

Die wichtigsten Änderungen, die wir vorgenommen haben sind:

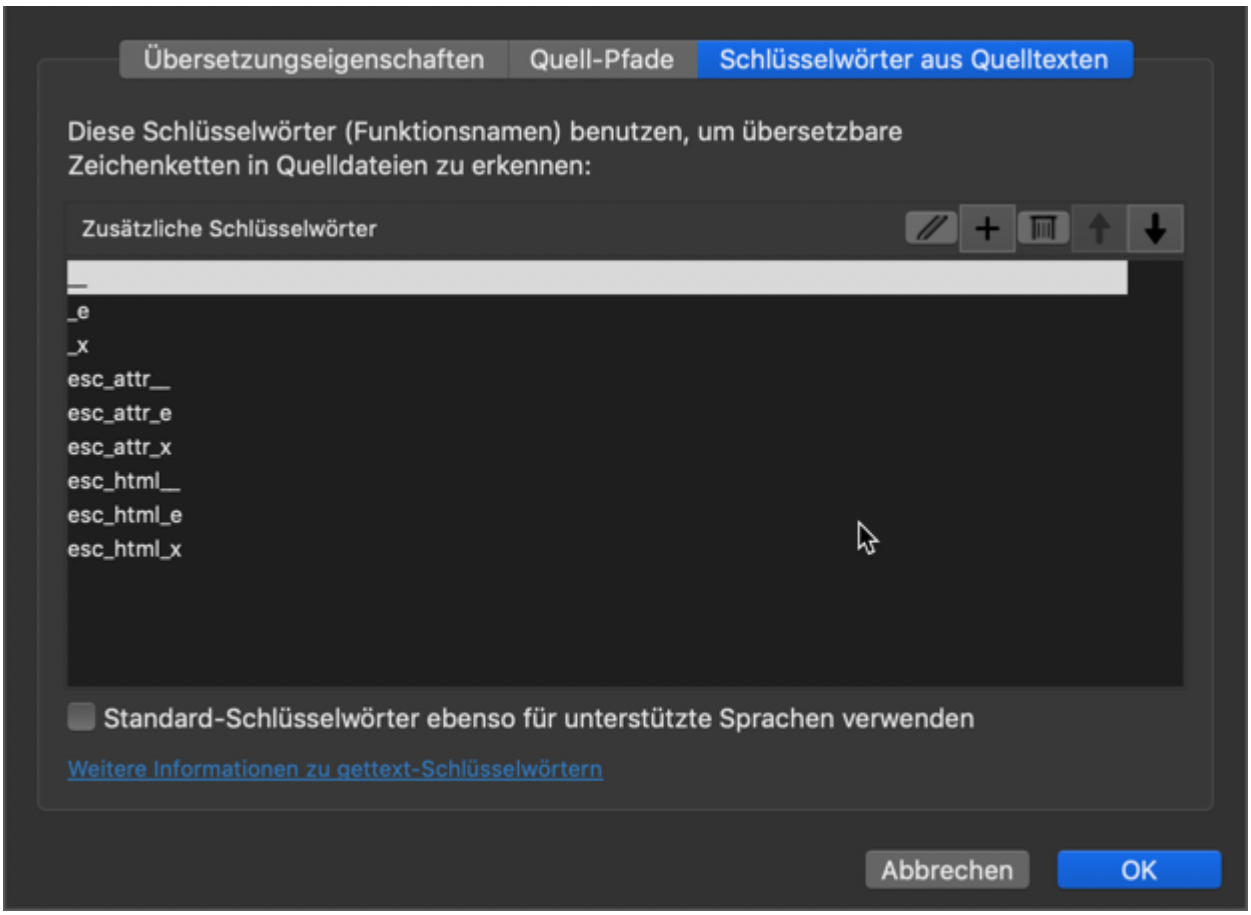
- Zeichensatz des Quellcodes auf UTF-8 gesetzt



- Quell-Pfad auf den "theme" Ordner eingestellt, sodass alle Theme-Strings im Theme-Ordner durchsucht werden.



- Schlüsselwörter aus Quelltexten definiert. Diese sind die Übersetzungs-Funtkionen die Wordpress zur Verfügung stellt.

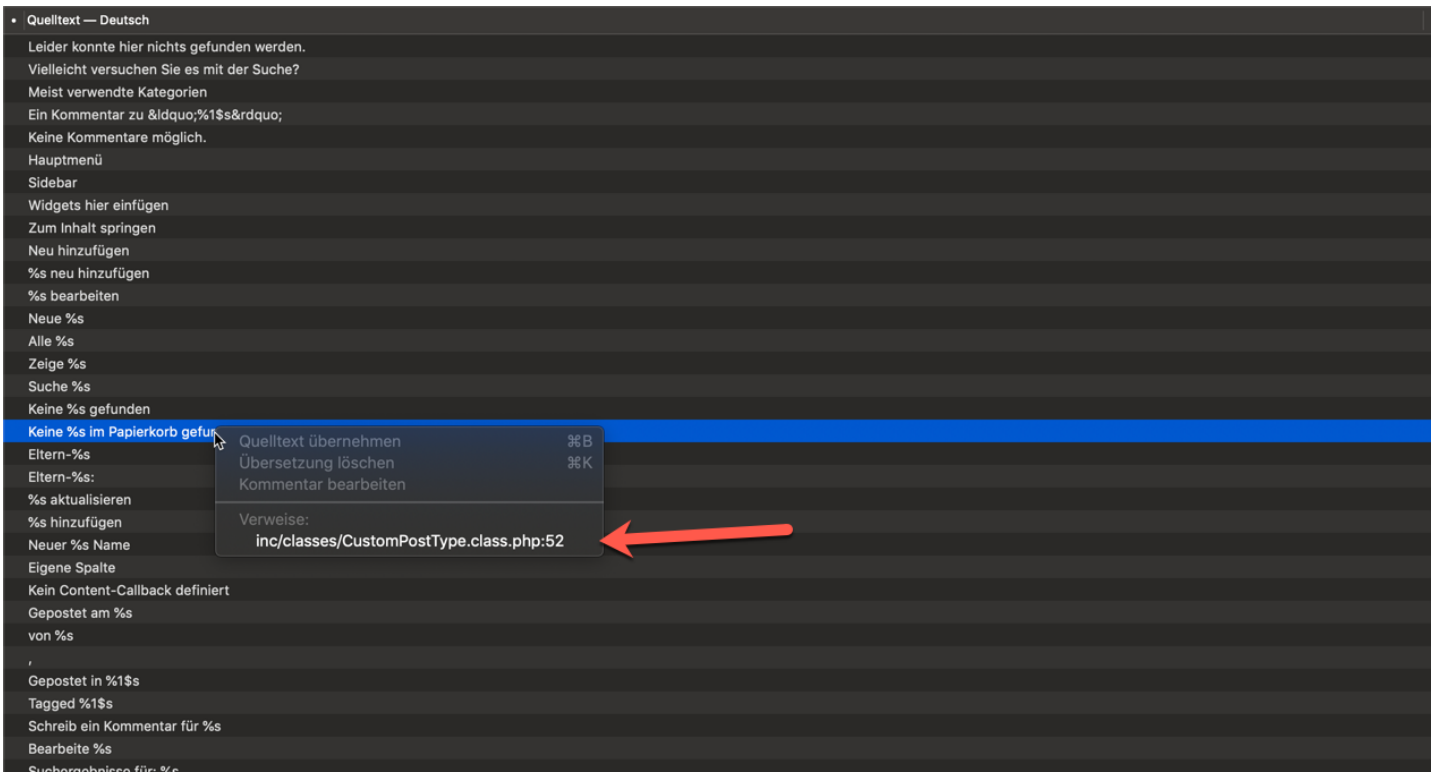


Falls hier Funktionen fehlen, die im Theme verwendet werden, müssen diese über das +-Zeichen hinzugefügt werden.

Wie finde ich heraus wo ein gewisser Übersetzungs-Text im Source-Code vorhanden ist?

Wenn die .pot bzw. .po-Datei wie oben beschrieben mit PoEdit erstellt wurde kann bei jedem Übersetzungs-String die Datei und Zeile des Strings wie folgt angezeigt werden:

`Rechtsklick auf den Übersetzungs-Text`



Version 1.0
Last updated 2019-06-25 09:22:24 +0200