



Copyright (C) Open Source Press

Frank Bültge ▪ Thomas Boley

Das WordPress-Buch

Vom Blog zum Content-Management-System

Copyright (C) Open Source Press

Alle in diesem Buch enthaltenen Programme, Darstellungen und Informationen wurden nach bestem Wissen erstellt. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grunde sind die in dem vorliegenden Buch enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor(en), Herausgeber, Übersetzer und Verlag übernehmen infolgedessen keine Verantwortung und werden keine daraus folgende Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht, auch nicht für die Verletzung von Patentrechten, die daraus resultieren können. Ebenso wenig übernehmen Autor(en) und Verlag die Gewähr dafür, dass die beschriebenen Verfahren usw. frei von Schutzrechten Dritter sind.

Die in diesem Werk wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. werden ohne Gewährleistung der freien Verwendbarkeit benutzt und können auch ohne besondere Kennzeichnung eingetragene Marken oder Warenzeichen sein und als solche den gesetzlichen Bestimmungen unterliegen.

Dieses Werk ist urheberrechtlich geschützt. Alle Rechte, auch die der Übersetzung, des Nachdrucks und der Vervielfältigung des Buches – oder Teilen daraus – vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlags in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Bibliografische Information Der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Copyright © 2009 Open Source Press, München
Gesamtlektorat: Dr. Markus Wirtz
Korrektorat: Franz Mayer
Satz: Open Source Press (L^AT_EX)
Umschlaggestaltung: www.fritzdesign.de
Gesamtherstellung: Kösel, Krugzell

ISBN 978-3-937514-70-3

<http://www.opensourcepress.de>

10.2 Eigene Themes für WordPress

Nach der allgemeinen Einführung in Layout-Frameworks soll es im Folgenden zunächst um die Bestandteile eines WordPress-Theme gehen, bevor anhand von zwei unterschiedlichen Layout-Frameworks Schritt für Schritt die Erstellung eines Theme erklärt werden soll.

10.2.1 Grundlagen – Themes und Templates

Wer sich mit WordPress und Themes auseinandersetzt, stolpert zwangsläufig über den Begriff Template. Dabei scheint oft nicht ganz klar zu sein, was denn genau ein Theme ist und was ein Template. Oft werden beide Begriffe im Zusammenhang mit WordPress auch synonym verwendet, obwohl es einen deutlichen Unterschied gibt. Das führt zu Missverständnissen – eine schlechte Grundlage für die Entwicklung eines eigenen Theme, darum also: Was ist ein Theme und was ein Template?

Template

Ein Template ist ein logisch zusammenhängender Block von PHP-Code, der eine bestimmte Aufgabe erfüllt. Die Loop, das Herzstück jedes WordPress-Theme, ist zum Beispiel ein solches Template.

Template-Datei

Die Template-Datei besteht aus einem oder mehreren Blöcken von Code (Templates). Innerhalb der Template-Datei können weitere Template-Dateien eingebunden werden.

Theme

Ein Theme ist eine Sammlung von Dateien, also Templates und CSS-Dateien, aber auch Grafiken, die die Darstellung des Weblog beeinflussen. Durch das Zusammenspiel der einzelnen Dateien und der Anweisungen in den Templates wird die Ausgabe der Inhalte im Frontend von WordPress festgelegt.

10.2.2 Ein neues WordPress-Theme in einer Minute

Bisher haben Sie in diesem Buch WordPress-Themes lediglich verwendet, vielleicht sind Sie aber auch neugierig geworden, wie Sie so ein Theme selber erstellen können. Wenn Sie ein wenig im Installationsverzeichnis von

WordPress stöbern, stoßen Sie auf den Ordner `themes`, der sich im Verzeichnis `wp-content` befindet. Innerhalb des `themes`-Ordners liegen, wie Sie anhand des Namens wahrscheinlich schon vermutet haben, die Themes, die Sie im Backend von WordPress auswählen können. Ein neues Theme wird an dieser Stelle installiert. Es besteht immer aus einem eigenen Ordner und verschiedenen Dateien.

Um Ihr erstes eigenes Theme zu erstellen, legen Sie im Verzeichnis `themes` einen Ordner an. Geben Sie diesem Ordner einen eindeutigen Namen (ohne Umlaute und Sonderzeichen), rufen Sie anschließend im Backend Ihres Blogs im Bereich **Themes** den Menüpunkt **Themes** auf. Bereits jetzt wird Ihr Theme von WordPress erkannt, allerdings erhalten Sie eine Fehlermeldung. Wenn nur der Ordner angelegt ist, betrachtet WordPress das Theme als beschädigt, da `Stylesheet` und `Template` fehlen. Genauer gesagt besteht ein Theme für WordPress also immer aus mindestens zwei Dateien: `index.php` und `style.css`.

Wenn Sie in dem gerade erstellten Ordner diese zwei Dateien anlegen, erkennt WordPress das Theme, und es lässt sich im Backend aktivieren. Wenn Sie anschließend Ihr Blog aufrufen, erhalten Sie nur eine leere Seite. Lassen Sie sich von Ihrem Browser den Quelltext der Seite anzeigen – hier ist ebenfalls nichts zu sehen. Wenn Sie die Datei `index.php` in Ihrem Theme-Ordner öffnen und dort `Hallo Welt` hineinschreiben (abspeichern nicht vergessen), so wird dieser Text beim erneuten Laden der Blog-Seite angezeigt.

Damit haben Sie Ihr erstes Theme erstellt, das aber noch über keine echte Funktion verfügt. Was sich aber bereits erkennen lässt, ist die Art der Verarbeitung eines Theme in WordPress: Es wird in WordPress nicht eingelesen und verarbeitet, sondern direkt an das Frontend weitergereicht. Sämtliche Bestandteile einer Blogseite müssen somit innerhalb eines Theme vorhanden sein. Das gilt auch für den HTML-Header, der nicht eigens von WordPress erstellt wird. Wenn Ihr Theme ein vollständiges HTML-Dokument erzeugen soll, müssen Sie also entsprechend in Ihrem Theme dafür Sorge tragen.

Funktionsweise eines Theme

Ein WordPress-Theme besteht aus HTML als Gerüst, das über PHP-Anweisungen mit Inhalt gefüllt wird. Dabei wird das gesamte Ergebnis über CSS in Form gebracht. Was dann im Frontend zu sehen ist, demonstriert das Default-Theme von WordPress. Wenn Sie sich die Verzeichnisinhalte des Theme ansehen, finden Sie eine Vielzahl von Dateien, die einzelne Aspekte des Blogs abdecken. Neben diesen Dateien, die die Funktionslogik eines Theme enthalten, gibt es noch Style-Dateien und einen Ordner mit Grafiken und Dateien, die für die Lokalisierung eines Theme benötigt werden. Abgerundet wird ein Theme durch die Datei `screenshot.png`, die im Ba-

ckend von WordPress ein Thumbnail des Theme anzeigen soll. Zentraler Eintrittspunkt eines Theme ist dabei immer die Datei `index.php`. Sofern neben der `style.css` keine weiteren Dateien im Ordner des jeweiligen Theme vorhanden sind, erwartet WordPress, dass sämtliche Darstellungsmöglichkeiten des Blogs von `index.php` geregelt werden.

Wie wir gesehen haben, reichen bereits zwei Dateien aus, damit sich ein Theme in WordPress aktivieren lässt. Inhalte liefert ein solches Theme allerdings zunächst nicht. Damit Sie aber nicht mühsam mittels PHP die Tabellen, in denen WordPress in der Datenbank seine Inhalte speichert, selber abfragen müssen, stellt WordPress Ihnen Befehle zur Verfügung: Die sogenannten Template-Tags sind Methoden und Funktionen von WordPress, auf die in einem Theme zurückgegriffen werden kann. Im Umkehrschluss bedeutet das auch, dass ein WordPress-Theme ausschließlich mit WordPress lauffähig ist.

Auch wenn sich ein Theme mit lediglich zwei Dateien erstellen lässt, ist es übersichtlicher, unterschiedliche Elemente und Funktionen einer Seite in eigene Dateien auszulagern. WordPress kennt bereits einige Dateien, die sich daher mit einem kurzen Befehl einbinden lassen, ohne dass dafür explizit der Pfad zur Datei angegeben werden muss. Drei dieser besonderen Dateien sind `header.php`, `sidebar.php` und `footer.php`. Die Namen der Dateien liefern schon einen Hinweis darauf, für welchen Zweck sie gedacht sind.

Bevor wir uns den weiteren Aufbau näher ansehen, sollten Sie die drei Dateien in Ihrem Theme-Ordner erstellen. Schreiben Sie in jede der Dateien einen kurzen Text, der die Funktion der Datei beschreibt. Wenn Sie jetzt im Frontend Ihres Blogs das Theme erneut aufrufen, werden Sie keinen Unterschied feststellen. Die soeben angelegten Dateien müssen über entsprechende Tags in `index.php` eingebunden werden. Öffnen Sie `index.php` Ihres Theme und fügen Sie die folgenden Tags hinzu:

```
<?php get_header(); ?>
<?php get_sidebar(); ?>
<?php get_footer(); ?>
```

Wie Sie sehen, sind die Tags kurze PHP-Anweisungen, die Funktionen von WordPress aufrufen. Mit diesen Befehlen haben Sie ohne weitere Zusätze die anderen Dateien in Ihrem Theme-Ordner eingebunden. Wenn Sie die Reihenfolge der Template-Tags in Ihrem Theme ändern, so werden die Dateien entsprechend dieser neuen Reihenfolge eingebunden.

Neben den bereits vorgestellten drei Include-Tags gibt es noch eine vierte Anweisung, mit der üblicherweise Kommentare in einem Theme eingebunden werden:

```
<?php get_comments(); ?>
```

11

Eigene Plugins für WordPress

WordPress bietet aktuell im offiziellen Plugin-Verzeichnis über 400 Plugins an, die alle unter der GPL stehen. Dennoch wird es vorkommen, dass Ihr ganz spezieller Lösungswunsch nicht darunter ist, und so werden Sie selbst Hand anlegen wollen. WordPress bietet ein ausgezeichnetes System zum Einbringen von Erweiterungen, gute PHP-Kenntnisse vorausgesetzt. Einen PHP-Kurs dürfen Sie hier allerdings nicht erwarten; Ziel dieses Kapitels ist, Verständnis für das Schnittstellen-Konzept von WordPress zu schaffen, um individuelle WordPress-Erweiterungen zu entwickeln.

11.1 Hintergrundwissen zu Plugins

Die Plugin-Schnittstelle erlaubt es, zusätzliche Funktionalität außerhalb des eigentlichen Systems (Core) zu halten, was auch Updates erleichtert. Sinnvoll ist es, so wenige Plugins wie möglich zu verwenden, da unter zusätzlichen Funktionen auch die Performance leidet. Allerdings ist es meist besser, ein Plugin einzusetzen als die Originaldateien zu verändern.

11.1.1 Grundlegende Hinweise zu Plugins

Für jedes Plugin gelten einige Grundregeln:

- Halten Sie die WordPress Coding Standards¹ ein.
- Plugins werden nur erkannt, wenn sie unterhalb von `wp-content/plugins` liegen.
- Ein Plugin kann als einzelnes File vorliegen oder auf mehrere verteilt sein. Mindestens in diesem Fall sollte es einen Ordner für das Plugin geben. Generell fördert es die Übersicht, jedes Plugin in einem eigenen Ordner unterhalb von `wp-content/plugins` zu haben.
- Der Nutzer sollte das Plugin über die Admin-Oberfläche von WordPress aktivieren bzw. deaktivieren können. Optimal ist ein Plugin, wenn die Anwender für die Inbetriebnahme keine Änderungen am Code vornehmen müssen.
- Ein Plugin sollte idealerweise über die Administrationsoberfläche konfigurierbar sein.
- Der Namensraum sollte eindeutig sein.
- Sobald das Plugin fertig ist und möglichst von mehreren Anwendern unter verschiedenen WordPress-Versionen getestet wurde, kann es veröffentlicht werden. Dazu empfehlen sich zwei Seiten:
`wordpress.org/extend/plugins/` als offizielles, zentrales Plugin-Verzeichnis
`wp-plugins.net` als Plugin-Datenbank, die sehr viele Plugins listet und vorstellt

WordPress erlaubt es, eigene Seiten in die Administrationsoberfläche zu integrieren, um beispielsweise Einstellungen und Ausgaben im Backend zu realisieren. Das Speichern und Laden von Daten geschieht über Hooks (siehe Seite 241). Ebenfalls sollten mögliche Einstellungen nicht hart im Quellcode geändert werden. Hierbei ist allerdings im Vorfeld zu überlegen, wie oft eine derartige Konfiguration wirklich geändert wird, um die Administrationsoberfläche nicht mit einmaligen Abfragen zu überfüllen.

Die wichtige Eindeutigkeit des Namensraums ist nicht immer einfach umzusetzen, besonders bei der großen Anzahl zusätzlicher Funktionen für WordPress. Vergeben Sie am besten einen Namen – das gilt auch für die Namen aller verwendeten Funktionen, nicht nur der Plugins – aus dem Namen des Plugins und der jeweiligen Funktion oder aus dem Autorennamen und der Funktion.

¹ http://codex.wordpress.org/WordPress_Coding_Standards

Statt eine Funktion einfach `reader()` zu nennen, empfiehlt es sich, den Plugin-Namen mit aufzunehmen; geht es um eine Reader-Funktion des Plugins „RSSImport“ also entsprechend `rssimport_reader()`. Noch eindeutiger wird es, wenn Sie den Namen des Autors oder dessen Initialen mit einbinden, also etwa `fb_rssi_reader()`.

Allerdings ist darauf zu achten, dass der Code lesbar bleibt. Dazu empfiehlt sich die Einhaltung des *PHP Coding Standard*.²

Den Namen des Plugins kann man auf den Seiten prüfen, die die WordPress-Plugins listen (siehe Seite 309). Der Name muss nicht zwingend, sollte aber eindeutig sein, da es weniger Verwechslungen bei Anfragen und Problemen gibt, falls das Plugin an Popularität gewinnt.

11.1.2 Plugin-Vereinbarung

Einige wenige Zeilen genügen, damit WordPress das Plugin im Verzeichnis der Administrationsoberfläche dem Anwender zeigt. Sie sollten sich in jedem Plugin am Anfang der Datei befinden:

```
<?php
/*
Plugin Name: Example (Plugin-Name)
Plugin URI: http://example.com/plugin_example/123/ (Link zur Pluginseite)
Description: Beschreibung (Kurzbeschreibung, XHTML möglich)
Author: Vorname Name (Autor)
Version: 1.1 (Versionsnummer)
License: GPL (Lizenz)
Author URI: http://example.com (Link zur Webseite des Autors)
*/
```

11.1.3 Plugin-API, Filter-Hooks und Tätigkeits-Hooks

WordPress stellt dem Anwender eine API bereit – die Plugin-API³, mit der der Zugriff und das Übergeben von Daten auf das System enorm erleichtert werden.

Grundsätzlich unterscheidet die WordPress-API zwei Arten von *Hooks* – Filter-Hooks und Tätigkeits-Hooks. Filter-Hooks (`add_filter()`) ermöglichen das Hinzufügen und das Entfernen von Funktionen, die nach spezifizierten Daten filtern (normalerweise Text). Dies ermöglicht Plugins, Inhalt oder Text schnell zu ändern. Tätigkeits- oder Action-Hooks (`add_action()`) ermöglichen das Hinzufügen und das Entfernen von Funktionen, deren Durchführung durch eine Core-Datei von WordPress ausgelöst wird.

² Die vollständige Liste samt Erläuterungen finden Sie unter <http://php-coding-standard.de>.

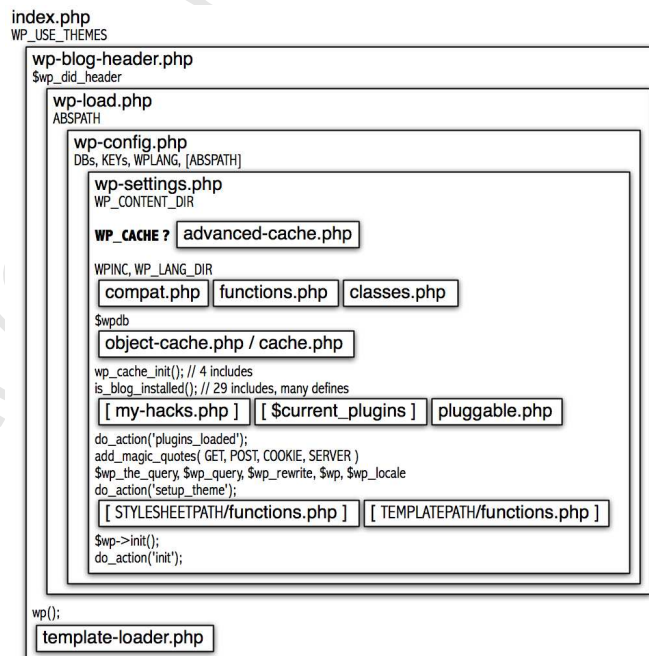
³ http://codex.wordpress.org/Plugin_API

Alle Filter- (*Filters*) und Tätigkeits-Hooks (*Actions*) sind auf der Plugin-API-Seite⁴ zu finden oder auf den Blogs „WordPress Hooks“⁵ und „WordPress Hooks, Filter“⁶ ausführlich erklärt.

11.1.4 WordPress Stack

Um Erweiterungen zu erstellen – seien diese über die Plugin- oder die Theme-Schnittstelle eingebunden – sind Kenntnisse über den Prozess hilfreich, wie WordPress diese aufruft. So kann man gezielt Konstanten und Variablen nutzen. In Abbildung 11.1⁷ ist der Prozess mit Nennung von Datei, Konstanten und Variablen schematisch dargestellt. Konstanten erkennen Sie an ihrer Schreibweise nach dem PHP Coding Standard in Großbuchstaben und Variablen am einleitenden \$. Ebenso finden Sie Hinweise auf entscheidende Funktionen und Hooks des jeweiligen Bereichs. Hooks erkennen Sie, weil sie immer über die Funktion `do_action()` im Core hinterlegt sind. Auf diese Weise können Sie auch in zukünftigen Versionen neue Hooks erkennen.

Abbildung 11.1:
WordPress Stack



⁴ http://codex.wordpress.org/Plugin_API

⁵ <http://wphooks.flatearth.org/>

⁶ http://adambrown.info/p/wp_hooks

⁷ Vielen Dank an Andy Skelton: <http://andy.files.wordpress.com/2008/10/wp-stack-1024.png>

Ein Hinweis zum Cache von WordPress: Die Datei `advanced-cache.php` wird nicht vom Core geliefert. WordPress fragt lediglich die Konstante `WP_CACHE` ab. Damit kann gezielt auf Funktionen eingegangen werden, die die Cache-Funktionalität von WordPress erweitern. Mithilfe von `advanced-cache.php` kann eine zusätzliche Cache-Funktionalität eingebracht werden.

WordPress Action- und Filter-Verlauf

Beim Schreiben von Erweiterungen, ob im Plugin oder Theme, kann eine Übersicht über Filter- und Action-Hooks nützlich sein, etwa zur Fehlersuche. Um den Fluss der Hooks auszulesen genügt es im Grunde, wenn man sich ein Bild des Inhalts der globalen Variable `$wp_filter` macht. Alternativ können Sie die Datei `wp-hooks-filter.php` im Download-Bereich zum Buch⁸ nutzen. Diese kopieren Sie in das Installationsverzeichnis Ihrer Entwicklungsumgebung und rufen sie direkt im Browser auf. Danach erhalten Sie alle Hooks, die von einer Funktion angesprochen werden, und die entsprechende Funktion dazu. Mithilfe dieser Methode lassen sich Hooks erkennen und Performanceprobleme entdecken, Zugriffe verstehen und Optimierungen prüfen. Darüber hinaus können Sie die Inhalte von Feldern und Arrays lesen.

WordPress Cache-Objekt

Der Cache von WordPress ist ein umfassendes Array und insbesondere bei der Fehlersuch sehr hilfreich. Daher gibt es auch dafür eine globale Variable, in der Sie alle Inhalte finden und erweitern: `$wp_object_cache`. Es genügt daher, wenn Sie diese Variable in lesbarer Form an Browser oder Editor ausleiten. Um Ihnen die Arbeit zu erleichtern, finden Sie dazu eine Datei im Download-Bereich zum Buch; kopieren Sie die Datei `wp-cache.php` in das Installationsverzeichnis Ihrer Entwicklungsumgebung, wo auch `wp-config.php` liegt, und rufen Sie sie direkt im Browser auf. Das Ergebnis gibt Ihnen Auskunft über alle Objekte im Cache.

Alternativ können Sie mit dem Plugin „WP Cache Inspect“⁹ den Inhalt des Cache direkt im Backend von WordPress auslesen.

11.1.5 Optionen ab WordPress 2.8

Bis WordPress 2.8 musste man Daten stets via `$_POST` weitergeben und in die Datenbank schreiben. Mit der neuen Version stehen neue Möglichkeiten zur Verfügung, die diese Arbeit erleichtern und die Integration im Back-

⁸ <http://www.wordpressbuch.de/downloads/>

⁹ <http://bueltege.de/wordpress-cache-steuern-plugin/819/>

end von WordPress übernehmen. Das folgende Beispiel-Plugin soll dies verdeutlichen; beachten Sie, dass wir aus Gründen der Übersicht die zu Beginn genannten Coding-Richtlinien hier nicht berücksichtigen und uns auf das Notwendige konzentrieren.

Wenn das Plugin aktiv ist, wird über den Hook `admin_init` die neue Option übermittelt. Dazu dient die Methode `example_options_init()`. Optional kann eine Prüfung übergeben werden, die in dem Beispiel die Methode `example_options_validate()` übernimmt. Dies ist nur notwendig, wenn es sich um die Übergabe von Inhalten handelt, die einer eigenen Prüfung bedürfen. Andernfalls können Funktionen von WordPress genutzt werden. Handelt es sich um die Übergabe von Integer-Werten, dann ist `intval` zu nutzen, bei HTML empfiehlt sich `wp_filter_nohtml_kses`.

```
register_setting( 'example_options_options', 'example_name',
array(&$this, 'intval') );
register_setting( 'example_options_options', 'example_name',
array(&$this, 'wp_filter_nohtml_kses') );
```

Optional lassen sich Einzelwerte oder ein Array in die Datenbank schreiben, was das System sauberer, sicherer und schneller macht. Die Verwaltung der Einträge in der Datenbank übernimmt ebenfalls WordPress.

```
<?php
/*
Plugin Name: Example Options
Plugin URI: http://bueltege.de/
Description: Options in WordPress 2.8 and higher, register_setting() API
Author: Frank B&uuml;ltge
Author URI: http://bueltege.de/
*/

if ( !function_exists('add_action') ) {
    header('Status: 403 Forbidden');
    header('HTTP/1.1 403 Forbidden');
    exit();
}

if ( function_exists('add_action') ) {
    //WordPress definitions
    if ( !defined('WP_CONTENT_URL') )
        define('WP_CONTENT_URL', get_option('siteurl') . '/wp-content');
    if ( !defined('WP_CONTENT_DIR') )
        define('WP_CONTENT_DIR', ABSPATH . 'wp-content');
    if ( !defined('WP_PLUGIN_URL') )
        define('WP_PLUGIN_URL', WP_CONTENT_URL . '/plugins');
    if ( !defined('WP_PLUGIN_DIR') )
        define('WP_PLUGIN_DIR', WP_CONTENT_DIR . '/plugins');
    if ( !defined('PLUGINDIR') )
        define( 'PLUGINDIR', 'wp-content/plugins' );
```