

## Flexible Website-Verwaltung mit PostNuke, Teil 3

# Allround-Talent

von Axel Guckelsberger und Steffen Voß

Im letzten Teil unseres Workshops steigen wir langsam immer tiefer in die Architektur von PostNuke ein. Zum einen stellen wir Ihnen das Allround-Modul Pagesetter vor und zeigen Ihnen dann, wie Sie die Möglichkeiten des PostNuke-Frameworks für Ihre eigenen Implementierungen nutzen können.

Eine Stärke von vielen Content-Management-Systemen ist die Vielfalt der Erweiterungsmöglichkeiten. Für PostNuke gibt es dutzende Module, die verschiedenste Funktionen erfüllen. Doch oft ist es so, dass gerade die eine spezielle Funktion fehlt, die man unbedingt auf der Webseite benötigt. Je nach Anforderung gibt es dann verschiedene Möglichkeiten:

- Kleinigkeiten können Sie durch ein Plugin, wie im zweiten Teil des Workshops beschrieben, nachrüsten.
- Einfache Content-Verwaltungen können Sie mit Pagesetter selbst erstellen.
- Komplexere Funktionen können Sie als eigenes Modul programmieren.

### Pagesetter – die eierlegende Wollmilchsau

News, Downloads, Rezepte, Produkt-Informationen – das alles sind normalerweise homogene Daten, die bestimmte Felder in der Datenbank benötigen. Welche Informationen man aber angeben will, ist oft verschieden. Ein normales Modul für Rezepte hat deswegen oft bis zu fünf Extrafelder, die der Benutzer für sich anpassen oder weglassen kann. Das ist aber nur eine Krücke.

Mit Pagesetter können Sie sich Ihr eigenes Eingabeformular zusammenklicken. Die Daten werden entsprechend in der Datenbank gespeichert und Sie können am Ende Templates erzeugen, die noch ohne Layout einmal alle Daten ausgeben. Diese Kombination von Eingabeformular, Datenbankschema und Ausgabetemplates nennt sich in Pagesetter „Publication Type“. Es gibt verschiedene Feldtypen – einmal grundsätzlich String, Text und In-

teger, dann aber auch für URLs, Bilder aus einer Galerie, Uploads usw. Dazu können Sie für jedes Feld einstellen, ob es optional oder Pflicht ist und ob es mit der Site-Suche durchsuchbar sein soll.

Sie können unbegrenzt viele dieser Publication Types erstellen und mittels relationalen Feldern auch miteinander verknüpfen. In einer Musikerdatenbank können Sie einen Publication Type für die Künstler erstellen und wenn Sie in einem weiteren die Veröffentlichungen speichern, können Sie hier auswählen, zu welchem Künstler die CD gehört.

Diese ersten Schritte laufen in Pagesetter über einen Assistenten („Neuer Typ“), in dem Sie sich Ihren passenden Publication Type zusammenstellen können. Danach fängt die Handarbeit an: das Anpassen der Templates.

### Pagesetter Templates

Wie schon in den letzten zwei Teilen des Workshops gezeigt, sind Templates bei PostNuke keine Zauberei – und Programmierer würden die Behauptung zurückweisen, dass es sich bei der Bearbeitung um Programmierung handelt. Es gibt vier wichtige Templates für jeden Publication Type:

- *PubType-list-header.html*, *PubType-list-footer.html* und *PubType-list.html* für die Ausgabe der Listenansicht
- *PubType-full.html* für die Vollansicht ihrer Artikel, Rezepte ...

Jeder Publication Type hat eine ID, die in der Administration von Pagesetter ersichtlich ist. Diese ID ist die *tid* – jeder Artikel an sich hat eine Publication ID, die *pid*.

Entsprechend lässt sich die Listenansicht für Ihren Publication Type aufrufen, zum Beispiel mit `index.php?module=Pagesetter&func=view&tid=1` Die Vollansicht rufen Sie auf mit `index.php?module=Pagesetter&func=viewpub&tid=1&pid=1`

Dazu müssen Sie natürlich schon einen (oder mehrere Testartikel) eingegeben haben. Dieses können Sie in der Pagesetter-Administration tun, indem Sie dort neben Ihrem Publication Type auf NEU klicken. Füllen Sie alle von Ihnen angelegten Felder aus – alle anderen können Sie zunächst ignorieren. Aber Sie sehen hier schon, dass es die Möglichkeit gibt, Artikel zeitgesteuert on- und offline zu schalten, Sie können außerdem festlegen, in welchen Sprachen der Artikel angezeigt werden soll, und es gibt Hinweise auf die Workflows, auf die wir später kurz eingehen werden. Darüber hinaus müssen Sie sich keine Sorgen darüber machen, ob Sie einen Artikel aus Versehen löschen – Sie können jeden Publication Type so einstellen, dass er über eine Revisionskontrolle verfügt und so ermöglicht, alte Versionen von Artikeln wiederherzustellen.

Wie im zweiten Teil des Workshops vorgeschlagen, sollten Sie die Templates zur Bearbeitung von `/modules/pagesetter/pntemplates` nach `/themes/IhrThema/templates/modules/pagesetter` kopieren. So sind Ihre Anpassungen sicher vor künftigen Modulupdates.

Neben diesen Standard-Templates können Sie den URLs auch den Parameter `&tpl=irgendetwas` anhängen und mit zusätzlichen Templates *PubType-list-irgendetwas-header.html*, *PubType-list-irgendetwas-footer.html* und

*PubType-list-irgendetwas.html* bzw. *PubType-irgendetwas.html* zusätzliche Ansichten erzeugen. Darüber hinaus kann Pagesetter z.B. XML-Listen und einiges mehr erzeugen. Mehr dazu entnehmen Sie bitte dem Pagesetter-Manual [3].

Während Sie Ihre Testpublikationen erfasst haben, ist Ihnen sicherlich aufgefallen, dass Pagesetter automatisch Formulare basierend auf den Felddefinitionen des PubTypes erstellt. Dies ist eine sehr praktische Funktionalität, dennoch möchten Sie den Aufbau oder das Layout der Formulare vielleicht anpassen. In Pagesetter werden die Formulare mit Guppy verwaltet, einem kleinen Framework, welches aus XML-Definitionen HTML-Formulare erstellt. Weitere Informationen zu diesem Thema finden Sie unter [4].

### Pagesetter Workflows

Pagesetter unterstützt nicht nur viele Berechtigungsstufen der PostNuke-Zugriffsrechte, sondern zudem einen leistungsfähigen Workflow-Mechanismus. Wie in der Automatentheorie befindet sich eine Publikation immer in einem von mehreren frei definierbaren Zuständen (z.B. *waiting*, *approved*, *deleted*) und kann zwischen diesen wechseln, was durch eine oder mehrere Operationen passiert (z.B. das Freischalten oder Löschen der bearbeiteten Publikation).

Man kann es so formulieren, dass die PostNuke-Zugriffsrechte definieren, wer was in welchem PubType darf. Der Workflow bestimmt währenddessen, wer in dem PubType wann was darf. Die Erstellung eines angepassten Workflows läuft basierend auf eine Kopie eines existierenden Workflows, indem neue Operationen hinzugefügt und die Übergänge zwischen den Zuständen angepasst werden. Hierbei ist immer die Reihenfolge relevant, damit im Falle eines Fehlers in einer bestimmten Operation keine Datenleichen entstehen. Hilfestellung zum Einstieg in eigene Workflows finden Sie unter [5]. Mit dem Workflow-Mechanismus bietet Pagesetter somit eine offene Schnittstelle für jegliche funktionale Erweiterung der internen Prozesse.

Für die etwas kleineren Anwendungen reichen die Standard-Workflows für gewöhnlich aus, also können Sie direkt Ihre Inhalte nach Belieben mit PubTypes

zusammenbauen und anschließend die Darstellung in den Templates anpassen. Und es werden keinerlei Änderungen am Code nötig, wenn neue Felder hinzugefügt werden. Ein paar Mausklicks und – falls die Anzeige der neuen Felder gewünscht ist – eine Änderung in den Templates reichen in solchen Fällen vollkommen aus.

Die Praxis hat gezeigt, dass sich ein beträchtlicher Teil von Modulen problemlos mit Pagesetter abbilden lässt. Was nicht heißt, dass die Entscheidung nicht wohl überlegt sein sollte. Mithilfe von Relationen zwischen Publikationen, anpassbaren Formularen und Workflows lassen sich auch komplexere Applikationen umsetzen. Durch Plug-ins und den PostNuke-Funktionen lassen sich individuelle Erweiterungen leicht hinzufügen. Aktivieren Sie dazu noch einige Hooks für Pagesetter, um ohne viel Aufwand Funktionen wie Kommentare [6] oder Smilies [7] zu integrieren.

Nun möchten wir noch eine konkrete Funktion in einem Template einbauen, um zu demonstrieren, wie wenig Code nötig ist, um viel zu erreichen. Wir möchten in einem Publikationstyp unter der Vollansicht der Publikation 3 ein Kontaktformular einblenden, in allen anderen Publikationen soll der Login-Block erscheinen (der z.B. die Block-ID 9 hat). Das Kontaktformular wird in dem Fall mit Formicula abgebildet [8], einem Modul, welches die Erstellung verschiedener Formulare erlaubt. In diesem Beispiel verwenden wir das Formular mit der ID 2. Der notwendige Code im Template *<PubType>-full.htm* sieht wie folgt aus:

```
<!--[if $core.pid==3]-->
  <!--[pnmodfunc modname="formicula" type="user"
                                     func="main" form=2]-->
<!--[else]-->
  <!--[pnblock bid=9]-->
<!--[/if]-->
```

Natürlich ist dies ein recht simples Beispiel, aber es zeigt sehr gut das Potenzial, das hinter der PostNuke-Architektur steht und für ihren Einsatz spricht. Aber wie funktioniert das im Detail?

### PostNuke-Architektur

Innerhalb von PostNuke stehen eine Reihe von Funktionen zur Verfügung, die Ih-

nen das Erstellen von Erweiterungen und Anpassungen immens erleichtern. Insbesondere Validierungen und das Interagieren mit anderen Systemkomponenten werden so zu einem Kinderspiel.

Wechseln Sie in das Verzeichnis */modules/Example* – darin sehen Sie einige *.php*-Dateien, mitunter die folgenden:

- *pninit.php* – Methoden zum Initialisieren, Aktualisieren und Entfernen des Moduls
- *pntables.php* – Aufbau der verwendeten Datenbanktabellen
- *pnversion.php* – Informationen zum Modul, z.B. Name, Version, Berechtigungsschema

Wesentlich interessanter sind die nicht genannten Dateien, also *pnadmin.php*, *pnadminapi.php*, *pnuser.php* und *pnuserapi.php*. Hierbei handelt es sich um Funktionen für den Admin- und für den User-Bereich. Die Dateien *pn<name>.php* entsprechen der logischen Schicht und *pn<name>api.php* der Datenschicht eines Bereichs. So befinden sich in den Datenschichten meistens Funktionen mit Zugriffen auf externe Ressourcen wie beispielsweise eine Datenbank oder das Dateisystem. Die logische Schicht übernimmt indes Aufgaben wie Eingabevalidierung, Prüfung der Zugriffsrechte und die Kontrolle der Modul-Templates, welche sich im Unterverzeichnis *pntemplates* befinden.

Die beiden Bereiche *admin* und *user* sind per Konvention vorhanden, es sind jedoch auch ohne Weiteres andere Bereiche und APIs möglich. So benutzen manche komplexe Module bereits verschiedene APIs, um ihre Struktur zu verteilen, auch im Sinne einer leichten Wiederverwendbarkeit bestimmter Funktionen durch andere Komponenten.

Schauen wir uns nun an, wie Funktionen aus den beiden Ebenen in PostNuke aufgerufen werden. Dafür gibt es zwei Funktionen, die prinzipiell auf die gleiche Weise arbeiten:

```
pnModFunc($modname, $type='user', $func='main',
          $args=array())
pnModAPIFunc($modname, $type='user', $func='main',
            $args=array())
```

Beide Funktionen rufen die entsprechenden Modul-Funktion auf und liefern, falls vorhanden, den entsprechenden Rückgabewert zurück. Wir betrachten nun zwei Funktionen der Datei *pnuser.php*, öffnen Sie diese am besten zum Vergleich in einem Editor.

```
function Example_user_main()
{
```

Wie Sie sehen, spiegeln sich im Namen der Funktion die Parameter von *pnModFunc()* wieder. In den API-Dateien verhält sich die Namensgebung analog.

```
if(!pnSecAuthAction(0,'Example:','::',
                    ACCESS_OVERVIEW)){
    return pnVarPrepHTMLDisplay(_MODULENOAUTH);
}
```

*pnSecAuthAction()* ist eine Funktion, mit der Sie ganz einfach jegliche Zugriffsrechte überprüfen können. Mit *pnVarPrepForDisplay()* und *pnVarPrepHTMLDisplay()* werden Ausgaben mit und ohne HTML versehen, wodurch XSS-Attacken verhindert werden. Analog dazu existieren auch Funktionen zum Zugriff auf Datenbanken (*pnVarPrepForStore()*) und Dateisysteme (*pnVarPrepForOS()*).

```
$pnRender = new pnRender('Example');

return $pnRender->fetch('example_user_main.htm');
}
```

Mit *\$pnRender* wird eine neue Instanz der *pnRender*-Klasse erstellt. Deren *fetch()*-Methode liefert die Ausgabe eines per Dateinamen spezifizierten Templates zurück, was auch meistens der Ausgabe einer Funktion entspricht.

Die *display()*-Funktion des Example-Moduls zeigt einen einzelnen Eintrag an und bekommt im Gegensatz zur ersten ein Array mit Argumenten, meistens *\$args* genannt:

```
function Example_user_display($args)
{
    list($tid,
         $objectid) = pnVarCleanFromInput('tid',
                                         'objectid');
```

An dieser Stelle werten wir zwei GET-Parameter aus. Die Funktion *pnVarCleanFromInput()* dient in PostNuke der Eingabevalidierung und sollte konsequent eingesetzt werden.

```
...

$item = pnModAPIFunc('Example','user','get',
                    array('tid'=>$tid));

if (!$item) {
    return pnVarPrepHTMLDisplay(_EXAMPLEITEMFAILED);
}
```

Hier rufen wir eine API-Funktion des Example-Moduls auf, die eine DB-Abfrage

durchführt und uns die Daten des Eintrags als Array zurück liefert.

```
$pnRender->assign($item);

return $pnRender->fetch('example_user_display.htm');
}
```

Mit der Methode *assign()* des *pnRender*-Objektes können Variablen an das Template übergeben werden, sodass dort die Darstellung erfolgen kann.

## Modul-Templating mit pnRender

Sie haben bereits gesehen, wie die Template-Kontrolle in der logischen Schicht abläuft: Sie erstellen eine *pnRender*-Ins-

Anzeige

## Life is too short for Java!



**Aspera ist Marktführer** im schnell wachsenden Markt des Software-Lizenzmanagements. Um unseren Entwicklungsvorsprung weiter auszubauen, wollen wir unser Entwicklerteam erneut verstärken.

**Wir bieten:**

- Ein junges motiviertes Team mit guter Atmosphäre
- Einen agilen Entwicklungsprozess
- Eine moderne Applikationsarchitektur
- Eine spannende internationale Kundengruppe (Deutsche Bank, Plan International, Siemens, Bosch, BASF...)
- Eine sehr breite technische Spielwiese (PHP, Oracle, MSSQL, MySQL, XML, SOAP, LDAP, UNIXe, LINUXe, SAP, alles aus Redmond und vieles mehr)
- Free Coffee&Soda, schnelle Rechner, eine leistungsgerechte Vergütung und was auch immer Sie brauchen, um sich täglich gerne neuen Herausforderungen zu stellen

**Sie verfügen über:**

- Eine sehr selbständige, aber teamorientierte Arbeitsweise (z.B. Interesse am Pairprogramming)
- Exzellente PHP-Kenntnisse
- Exzellente SQL-Kenntnisse
- Sehr gute Kenntnisse relevanter Web-Technologien (DHTML, CSS, JS...)
- Sehr gute Englischkenntnisse
- Keine Angst vor dem Kunden und den Wunsch, ihn immer glücklich zu machen

**Passt?** Dann freue ich mich, über **Keith.Sauvant@aspera.com** mit einem kurzen Profil von Ihnen zu hören.



tanz, übergeben die Variablen und verarbeiten die Ausgabe weiter. Aus einem Templating-System entsteht insgesamt eine riesige Flexibilität, vor allem durch die konsequente Trennung zwischen Inhalt und Form. Sie können beispielsweise auch Teil-Templates für jede Iteration einer Schleife erstellen, deren Ausgaben Sie zusammenführen und wiederum einem Master-Template zuweisen.

Die Funktionsweise, Syntax und Flexibilität der Smarty-Befehle innerhalb der Templates haben Sie bereits im zweiten Teil kennen gelernt, als Sie das Xanthia-Thema nach Ihren Wünschen angepasst haben. Falls Sie ein wenig mit Smarty herumexperimentiert haben [1], wissen Sie bereits, dass Sie innerhalb eines Templates Bedingungen und Schleifen definieren können und welche Freiheiten durch die Verwendung von Plug-ins entstehen. Dieses Wissen können Sie auch bei der Modulentwicklung nutzen, denn auch Module können Plug-ins enthalten, die in den Modul-Templates benutzt werden. Diese Modul-Plug-

ins werden unter *pntemplates/plugins* gespeichert.

Wir erinnern uns daran, wofür Plug-ins gedacht sind: In diesen kleinen Funktionen können wir Logik unterbringen, die nur für die Darstellung relevant ist. Ein typisches Beispiel ist das Zusammenbauen einer Select-Auswahlliste in Abhängigkeit

### Die wichtigsten PostNuke-Funktionen werden als Plug-ins angeboten.

von verschiedenen Kriterien. Da sich ein Plug-in problemlos parametrisieren lässt, kann es in verschiedenen relevanten Templates wie gewünscht eingesetzt werden. Dies reduziert nicht nur Redundanzen, das wirklich Schöne an Modul-Plug-ins ist, dass diese natürlich auch in der Lage sind, pnRender und somit wiederum Templates zu benutzen, wodurch Sie Ihr Modul um einen kleinen Werkzeugkasten mit dazugehörigen Templates anreichern

können, die nicht nur im eigentlichen Modul, sondern auch in einem anderen Kontext einsetzbar sind. Damit in den Templates von Modul A die Plug-ins von Modul B zur Verfügung stehen, müssen Sie folgende Schritte durchführen:

- Erstellen Sie das Verzeichnis *modules/A/pntemplates/config/*.
- Legen Sie darin eine Datei namens *use-modules* an (ohne Endung).
- Öffnen Sie diese Datei mit einem Editor und schreiben *B* hinein.

Um das Arbeiten mit Templates noch vielseitiger zu machen, werden die wichtigsten PostNuke-Funktionen auch als Plug-in angeboten, deren Ergebnisse sich meistens mithilfe des *assign*-Parameters einer Variablen zuweisen lässt. Tabelle 1 zeigt einige Beispiele, die verschiedenen Parameter entnehmen Sie bitte dem jeweiligen Plug-in unter */modules/pnRender/plugins/plugin.php*.

Obwohl diese Tabelle einen recht schlichten Eindruck macht, sind die ge-

### Ausblick auf PostNuke .8

Seit langer Zeit schon arbeiten die Core-Entwickler an der PostNuke Version .8. Im Mai ist nun endlich ein erster Milestone erschienen. Im Laufe der Parallelentwicklung von .7 und .8 zeichneten sich immer mehr strukturelle Änderungen ab, sodass die Unterschiede zum bestehenden Produktsystem immer umfangreicher und weitreichender wurden. Aus diesem Grunde wurden in den letzten Releases der .7er-Serie bereits einige Features aus .8 zurückportiert, wie eben die Einführung des Smarty-Templatings durch Xanthia und pnRender. Dennoch steht mit PostNuke .8 ein derart bedeutender Sprung an, dass wir Ihnen zumindest die wichtigsten Neuerungen kurz vorstellen wollen, damit Sie einen Eindruck von der zukünftigen Entwicklung bekommen.

#### Weg mit den Kummerpfunden!

Mit PN .8 werden endlich alle PHPNuke-Altlasten entfernt sein. In der aktuellen Version .762 gibt es noch einen Legacy-Modus, mit dem veraltete Funktionen auf Wunsch aktiviert werden können, beispielsweise für Drittmodule. Veraltete zentralistische Ansätze wie etwa die MyAccount-Seite oder die Suchfunktion werden dank spezieller APIs auf die Module verteilt.

#### Objekte!

Gleichzeitig führt .8 eine Menge von neuen Klassen ein, die im Gesamten eine konsistente Objektbibliothek bilden und dem Entwickler neue Werkzeuge bieten. So gibt es zum Beispiel mit DBUtil eine Zwischenschicht für Datenbankzugriffe, die die manuelle Formulierung von SQL-Abfragen seitens des Modulentwicklers weitestgehend überflüssig macht und somit Fehlerquellen in diesem Bereich reduziert. Insgesamt haben Messungen mit Core-Modulen ergeben, dass durch die neue Codebase trotz neuer Funktionalität etwa 40 Prozent Code gespart werden. Ein Connection Stack ermöglicht eine leichte Verwendung zusätzlicher Datenbanken, beispielsweise für die Anbindung externer Applikationen. Der Core von PostNuke .8 wird außerdem zur Zeit auf Oracle-Kompatibilität hin optimiert.

Die Authentifizierung kann in .8 von verschiedenen Quellen wie etwa einem LDAP-Server erfolgen. Mit dem Categories-Modul gibt es eine zentrale Stelle für die systemweite Kategorisierung von Inhalten. Das FormsAPI ist eine Art Nachfolger von Guppy, der Hintergedanke ist es, dank auf XML basierenden Formularen die Darstellung zu vereinheitlichen und Fehlerquellen zu reduzieren.

.8 wird ein Workflow-System enthalten, ähnlich zu dem bestehenden in Pagesetter. So können Anpassungen der Arbeitsabläufe durch eigene Workflows realisiert werden, was eine Anpassung der Module, die die Arbeiten durchführen, unnötig macht.

Darüber hinaus bietet der .8er Core dem Entwickler integrierten AJAX-Support durch bekannte Bibliotheken wie prototype [9] und script.aculo.us [10], wovon auch bereits einige Core-Module Gebrauch machen. Weitere Features sind unter anderem erweitertes Error Logging, die Definition von Modulabhängigkeiten, das Informationsmodul SysInfo und die Zentralisierung aller sicherheitsrelevanten Optionen im SecurityCenter.

Die bisherigen Core-Module werden der aktuellen Codebase angepasst und später interessierten Modulentwicklern unterstellt. So soll die Entwicklung des eigentlichen Core-Systems separiert werden, was sich positiv auf die Release-Zyklen auswirken dürfte. Für Endbenutzer wird es dann wahrscheinlich verschiedene Distributionen zum Download geben, in denen mit gewählten Modulpaketen gängige Szenarien wie Blog, Business oder Community unterstützt werden.

# Beste Bücher für besten Code!

nannten Plug-ins derart vielseitig, dass sich ein Blick in die jeweiligen Dateien lohnt, zumal dort Kommentare mit Beispielen stehen, die die Parameter beschreiben. Viele Funktionen, die auf den ersten Blick eher wenig zu tun scheinen, offenbaren ihren Sinn erst mit Blick auf das Gesamtsystem. So bekommt die Funktion `pnModURL()`, die eine URL zusammenbaut, einen größeren Stellenwert als es zunächst scheint, wenn man bedenkt, dass sich Module in der Modules-Administration umbenennen lassen, was natürlich nur funktionieren kann, wenn die in den Templates benutzten URLs nicht hart kodiert sind.

Bei Modulen, die API-konform programmiert sind, können Sie obendrein sogar im Content selbst (z.B. im Text des News-Artikels) Plug-in-Aufrufe platzieren. Dieser wird dann von `pnRender` behandelt als stünde er im Template.

Sie sehen, das Ganze ist eine recht komplexe Sache, aber dafür auch eben rundum anpassbar. Sie können beliebig einzelne Teile verschiedener Komponenten kombinieren und zusammenführen. Mit etwas Übung ist bei einem fremden Modul schnell ersichtlich, welche Funktion aufgerufen werden muss, um gewünschte Inhalte zu erhalten.

Hinter dem Plug-in-System verbirgt sich eine riesige Mächtigkeit, aber an diese können Sie sich problemlos Stück für Stück herantasten. Und je mehr Erfahrungen Sie sammeln, desto mehr werden Sie diese Art der Entwicklung zu schätzen wissen. Sie lernen neue Module kennen und betrachten irgendwann PostNuke nicht mehr als eine CM-Lösung, sondern begreifen es als ein flexibles Framework, mit dem Sie schnell und intuitiv bestehende Komponenten verknüpfen können.

Plug-in	Bedeutung
<code>&lt;![pager]-&gt;</code>	Pager-Komponente
<code>&lt;![pnblock]-&gt;</code>	Zeigt einen Block an
<code>&lt;![pnimg]-&gt;</code>	Stellt ein Bild dar
<code>&lt;![pnml]-&gt;</code>	Zeigt eine Sprachkonstante an
<code>&lt;![pnmodapifunc]-&gt;</code>	Ruft eine API-Funktion auf
<code>&lt;![pnmodfunc]-&gt;</code>	Ruft eine Modul-Funktion auf
<code>&lt;![pnmodurl]-&gt;</code>	Bildet eine URL
<code>&lt;![pnusergetlang]-&gt;</code>	Liefert die Sprache des aktuellen Benutzers

Tabelle 1: Plug-in-Beispiele

Wenn Sie diesen Artikel lesen, sind wahrscheinlich bereits PostNuke .763 und PostNuke .8 MS2 als Cross-Release veröffentlicht worden. Sie können sich auch schon den Milestone lokal installieren und damit arbeiten – es wird keine grundsätzlichen Änderungen bis zur Final mehr geben. Der Unterschied zwischen dem Milestone und der Final ist vielmehr die Möglichkeit von bestehenden .7er-Seiten updaten und PostNuke .8 auf Produktivsystemen einsetzen zu können.

Wir würden uns freuen, wenn Sie dieser Artikel überzeugt hat, sich ein wenig mit PostNuke zu beschäftigen. Falls Sie Anmerkungen oder Fragen haben, können Sie gerne die deutsche Support-Seite [12] besuchen, wo man Ihnen mit Sicherheit schnell und kompetent weiterhelfen wird.



Axel Guckelsberger und Steffen Voß sind beide als Autoren im PostNuke e.V. als Entwickler und Organisatoren für die Community tätig. Sie erreichen sie per Mail unter [info@quite.de](mailto:info@quite.de) (Axel) bzw. [kontakt@kaffeeringe.de](mailto:kontakt@kaffeeringe.de) (Steffen).

## Links & Literatur

- [1] Handbuch für Smarty: [smarty.php.net/manual/de/](http://smarty.php.net/manual/de/)
- [2] Pagesetter: [www.elfisk.dk](http://www.elfisk.dk)
- [3] Pagesetter-Handbuch: [www.elfisk.dk/modules/pagesetter/docs/manual/PagesetterManual.html](http://www.elfisk.dk/modules/pagesetter/docs/manual/PagesetterManual.html)
- [4] Guppy-Handbuch: [www.elfisk.dk/modules/pagesetter/docs/manual/GuppyManual.html](http://www.elfisk.dk/modules/pagesetter/docs/manual/GuppyManual.html)
- [5] Workflow-Handbuch: [www.elfisk.dk/modules/pagesetter/docs/manual/Workflow\\_Manual.html](http://www.elfisk.dk/modules/pagesetter/docs/manual/Workflow_Manual.html)
- [6] EZComments: [noc.postnuke.com/projects/ezcomments/](http://noc.postnuke.com/projects/ezcomments/)
- [7] pn\_bb smile: [noc.postnuke.com/projects/pnbb smile/](http://noc.postnuke.com/projects/pnbb smile/)
- [8] Formicula: [noc.postnuke.com/projects/formicula/](http://noc.postnuke.com/projects/formicula/)
- [9] Prototype: [prototype.conio.net](http://prototype.conio.net)
- [10] [script.aculo.us](http://script.aculo.us)
- [11] Testseite für .8: [pn8.pn-cms.de](http://pn8.pn-cms.de)
- [12] Support bei der deutschen PostNuke-Community: [support.pn-cms.de](http://support.pn-cms.de)



Adam Bien

### Enterprise Architekturen

Leitfaden für effiziente Softwareentwicklung

234 Seiten, Hardcover, 39,90 €  
ISBN 3-935042-99-X



Lars Wunderlich

### Managing Java

Operating, Monitoring, Performancetests

210 Seiten, Softcover, 29,90 €  
ISBN 3-939084-13-1

Weitere Informationen und Bestellmöglichkeiten finden Sie unter [www.entwickler.press.de](http://www.entwickler.press.de). Unsere Bücher erhalten Sie auch in jeder gut sortierten Buchhandlung.