



**Ez publish 3.\***

**Grundlegende Konzepte**

**&**

**Grundlagen der Modulprogrammierung**



Dipl.-Ing. Felix Woldt (FH)

[felix@jac-systeme.de](mailto:felix@jac-systeme.de)

[www.jac-systeme.de](http://www.jac-systeme.de)



# Inhalt

<b>1 Einführung in das CMS eZ publish 3.*</b>	<b>2</b>
1.1 Interner Aufbau von eZ publish 3.*	2
1.1.1 Kernel, Klassenbibliotheken und Module	2
1.1.2 Verzeichnis-Struktur	4
1.2 Content und Design	5
1.2.1 Trennung von Content und Design	5
1.2.2 Templates	6
1.2.3 Speicherung der Daten	7
1.3 Content Management in eZ publish 3.*	7
1.3.1 Das Administrations-Interface	8
1.3.2 Content-Struktur, Klassen, Objekte, Attribute und Datentypen	9
1.3.3 Versionierung von Content-Objekten	10
1.3.4 Unterstützung mehrerer Sprachen	12
1.3.5 Content-Knotenbaum und Knoten	13
1.3.6 Sektionen	15
1.4 Site Management in eZ publish 3.*	17
1.5 Site	17
1.5.1 Site Interface	17
1.5.2 Site Access	17
1.6 URLs in eZ publish 3.*	19
1.6.1 System-URL	19
1.6.2 Virtuelle-URL	20
<b>2 Grundlagen der Modulprogrammierung</b>	<b>22</b>
2.1 Struktureller Aufbau einer Extension / eines Moduls	22
2.2 Einrichten einer Extension	23
2.3 Modulzugriffsmechanismen	26
2.4 Zugriffsrechte auf ein Modul	28
2.5 INI-Dateien	29
2.6 Template-System	31
<b>3 Verzeichnisse</b>	<b>32</b>
3.1 Tabellenverzeichnis	32
3.2 Abkürzungsverzeichnis	32
3.3 Literaturverzeichnis	33

# 1 Einführung in das CMS eZ publish 3.\*

Dieses Kapitel gibt einen groben Überblick über den Aufbau und die Funktionsweise von eZ publish 3.\*. Es stellt die Grundlage dar, um eZ publish 3.\* in der Funktionsvielfalt zu verstehen. Als Grundlage für dieses Kapitel diente das englischsprachige Dokument eZ Publish basics [eZDoc Basics]. Aus diesem wurden Grafiken und Texte übernommen, übersetzt und ergänzt.

## 1.1 Interner Aufbau von eZ publish 3.\*

Dieser Abschnitt beschreibt den internen Aufbau von eZ publish 3.\*, indem ein Überblick über die verschiedenen Software-Schichten gegeben wird. Es wird beschrieben, wo Daten in eZ publish 3.\* gespeichert werden.

### 1.1.1 Kernel, Klassenbibliotheken und Module

eZ publish 3.\* ist eine komplexe Anwendung, die vollkommen objektorientiert in der Programmiersprache PHP entwickelt wurde. Die Anwendung besteht dabei aus 3 Teilen (siehe Abbildung 1.1.1).

1. **Module** sind eine Sammlung von Funktionen, die eine Art Schnittstelle (Interface) zu einem Mechanismus aus dem Kernel darstellen. Jedes Modul besitzt bestimmte Funktionen für die Verwaltung und Ausführung verschiedener Aufgaben. Beispiele sind „content“, „error“, „search“ und „section“. Eine vollständige Liste aller in eZ publish enthaltenen Module gibt es auf der Webseite von eZ publish [eZDoc Appendix B]. Das Modul „map“ aus der zu entwickelnden Extension „mapsystem“ wird genau an dieser Stelle in das System eingefügt.

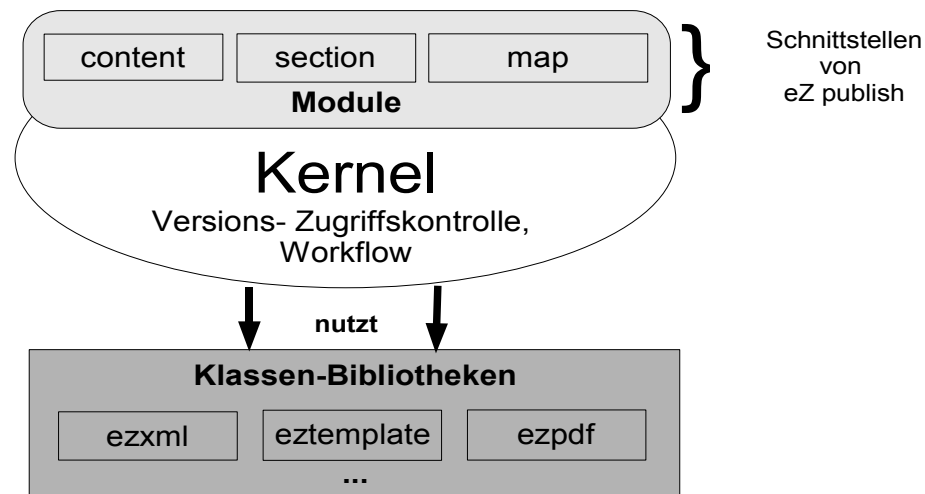
















Abbildung 1.1.1 Zusammenhang Module, Kernel und Klassenbibliotheken

2. Der **Kernel** stellt den Kern des Systems dar. Dieser enthält alle Low Level-Funktionen, wie die Handhabung der Inhalte (Content), des Workflows, der Versions- und Zugriffskontrolle. Grundsätzlich ist der Kernel eine Sammlung von Mechanismen, die auf den Klassen-Bibliotheken aufbauen.
3. Die **Klassenbibliotheken** (Libraries) bilden den Hauptteil von eZ publish 3.\*. Sie sind wiederverwendbare Mehrzweck-PHP-Klassen. Da Sie in keiner Abhängigkeit mit dem Kernel von eZ publish stehen, können Sie sehr einfach in anderen PHP-Anwendungen genutzt werden. Beispiele sind „ezpdf“, „eztemplate“ und „ezxml“. Eine vollständige Liste aller verfügbaren Klassenbibliotheken gibt es auf der Webseite von eZ publish unter [eZDoc Appendix A].

## 1.1.2 Verzeichnis-Struktur

Das Hauptverzeichnis von eZ publish 3.\* ist unterteilt in verschiedene Unterverzeichnisse. Jedes Unterverzeichnis stellt aus einer Sammlung von logisch verwandten Dateien. Die Verzeichnis-Struktur siehe Tabelle 1.1.1 ist eindeutig und somit einfach zu verstehen.

Tabelle 1.1.1 eZ publish 3.\* Verzeichnisstruktur

 <b>Hauptverzeichnis</b>	
 <b>bin</b>	Enthält verschiedene Perl- und Shell-Scripte. Diese dienen hauptsächlich Wartungszwecken.
 <b>cronjobs</b>	Enthält alle PHP-Scripte die von der „runcronjob.php“ ausgeführt werden
 <b>design</b>	Enthält alle Dateien die im Zusammenhang mit dem Design stehen, wie Templates, Banner-Bilder, Stylesheets, Schriften ...
 <b>doc</b>	Enthält die Dokumentation von eZ publish und Informationen über die Änderungen am System zur vorhergehenden Version (Change Logs)
 <b>extension</b>	Enthält alle Dateien von eigenen Erweiterungen / Modulen
 <b>kernel</b>	Enthält alle Dateien des Kernels, wie Klassen, Ansichten, Datentypen ... . Ist das „Herz“ von eZ publish. Änderungen nehmen hier nur Experten vor.
 <b>lib</b>	Enthält die Mehrzweckklassenbibliotheken. Die Bibliotheken sind eine Sammlung von Klassen für verschiedene Aufgaben. Der Kernel nutzt diese Bibliotheken, siehe Abbildung 1.1.1.
 <b>packages</b>	Enthält Content / Layout Erweiterungen für eZ publish
 <b>settings</b>	Enthält dynamische, seitenspezifische Konfigurationsdateien.
 <b>support</b>	Zusätzliche Programme, wie der Sprachentemplateparser ezlupdate
 <b>update</b>	Enthält alle Scripte die bei einem Systemupdate benötigt werden
 <b>share</b>	Enthält statische Konfigurationsdateien, wie Zeichenumsetzungstabellen, ortsbezogene Informationen und Übersetzungen.
 <b>var</b>	Enthält Zwischenspeicher- (Cache) und Log-Dateien. Enthält ebenfalls seitenspezifische Bilder und Files.  Die Größe des Verzeichnisses wächst mit der Zeit an!

## 1.2 Content und Design

Dieses Kapitel zeigt kurz wie eZ publish 3.\* Daten in ein Design integriert. Dabei wird gezeigt, welche Bedeutung Templates in diesem Zusammenhang haben.

### 1.2.1 Trennung von Content und Design

eZ publish 3.\* trennt den **Content** (Inhalt) vom Design. Content bezeichnet alle Informationen, die in einer Struktur für die spätere Weiterverarbeitung gespeichert werden. Der gesamte Content wird in einer Struktur der sogenannten **Content-Struktur** abgelegt.

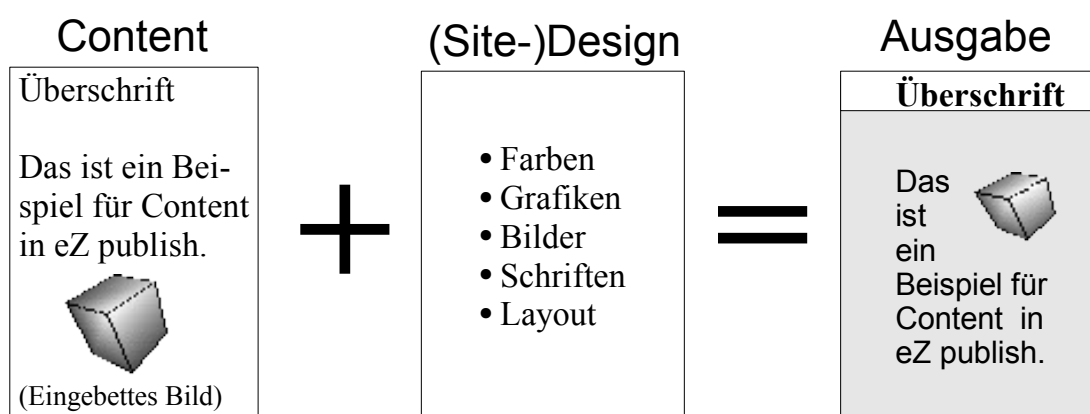


Abbildung 1.2.1 Trennung von Inhalt und Design

Das **Design** legt fest, wie die Informationen aus der Content-Struktur verarbeitet und für die **Ausgabe** formatiert werden sollen. Die Informationen können z.B. für das Internet oder für die PDF<sup>1</sup>-Ausgabe formatiert werden. Gesteuert wird das Design in eZ publish 3.\* über das sogenannte **Site Design**, die alle Dateien für diesen Zweck enthalten z.B. Stylesheets, Templates, Banner-Bilder. Abbildung 1.2.1 verdeutlicht dies noch einmal.

<sup>1</sup> Das Portable Document Format (PDF) ist ein Dateiformat, das von Adobe Systems entwickelt und 1993 mit Acrobat 1 veröffentlicht wurde. Es kann Texte, Schriftsätze und Bildinformationen in einem einheitlichen Dokument zusammenfassen. [NetLexikon 04]

## 1.2.2 Templates

eZ publish 3.\* nutzt Templates als grundlegenden Bestandteil in einem Site Design. Ein Template ist im Grunde eine erweiterte HTML<sup>2</sup>-Datei (andere Formate sind auch möglich). Es ist vergleichbar mit einer Formatvorlage und legt fest, wie bestimmte Inhalte formatiert und angezeigt werden sollen. Der Template-Mechanismus „eztemplate“ von eZ publish 3.\* besitzt für diese Zwecke eine mächtige Anzahl von speziellen Template-Funktionen und -Operatoren. Eine Auflistung dieser befinden sich im Internet unter [eZDoc Appendix E] (Template-Operatoren) und [eZDoc Appendix F] (Template-Funktionen). Die Abbildung 1.2.2 zeigt einmal schematisch wie eZ publish aus Daten und Templates die entsprechende Ausgabe erzeugt.

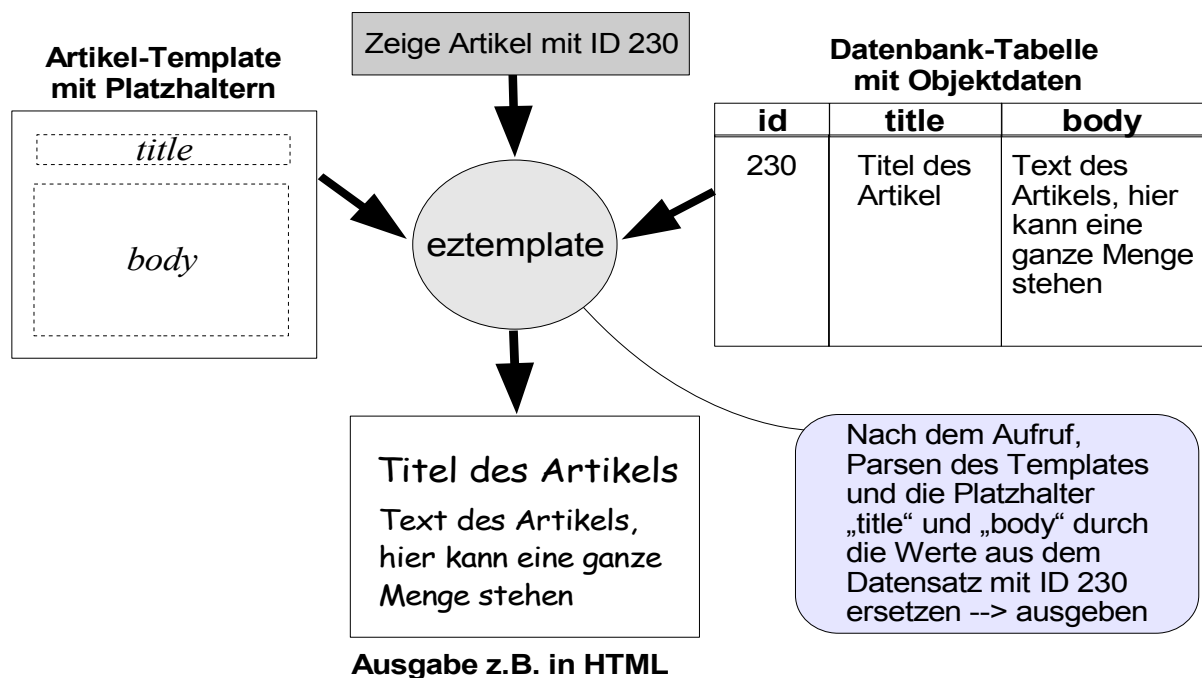


Abbildung 1.2.2 Template-Mechanismus eztemplate

### 1.2.3 Speicherung der Daten

eZ publish 3.\* strukturiert und speichert den Content in einer SQL-Datenbank. Ausgenommen hiervon sind Bilder und binäre Dateien. Diese werden im Dateisystem unter dem Verzeichnis „var“ abgelegt. Alle Dateien, die mit dem Design zusammenhängen, werden ebenfalls im Dateisystem unter dem Verzeichnis „design“ abgelegt.

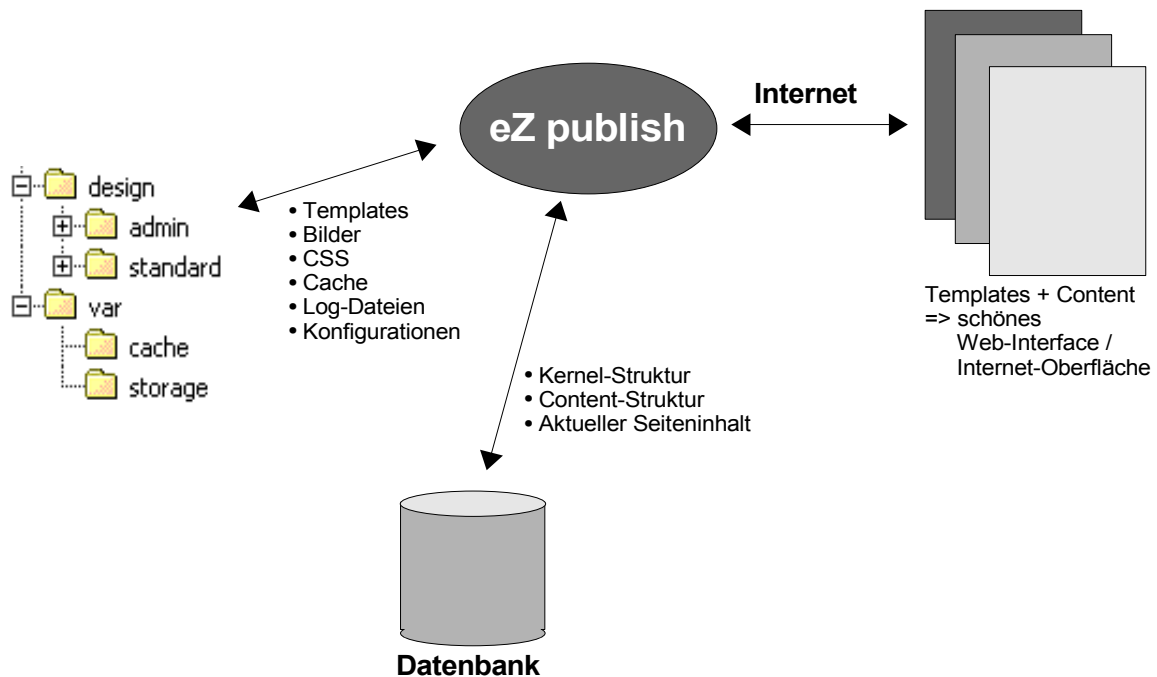


Abbildung 1.2.3 Schema - Speicherung von Daten unter eZ publish 3.\*

### 1.3 Content Management in eZ publish 3.\*

Jede erstellte Datei, ob es ein Dokument, ein Bericht, ein Bild, ein Video, ein Kundenkontakt oder ein Backup ist, kann als Content klassifiziert werden. Die Aufgabe eines Content Management Systems ist es, jedes einzelne Element dieses Contents unabhängig vom Typ und von der Komplexität zu organisieren. Dies erfordert eine Organisation mit einer strukturierten, automatischen und zugleich flexiblen Lösung, die eine freie Veröffentlichung und sofortige Aktualisierung über verschiedene Kommunikationswege z.B. Internet, Intranet, „Front-“ und „Back End“-Systeme erlaubt.

Dieses Kapitel beschreibt, wie eZ publish 3.\* Content strukturiert und verwaltet.

### 1.3.1 Das Administrations-Interface

In eZ publish 3.\* wird der Content mit Hilfe einer grafischen Benutzeroberfläche, dem Administrations-Interface verwaltet. Informationen, wie dieses bedient wird, gibt es auf der Webseite von eZ publish [eZDoc Admin].



Abbildung 1.3.1 Administrations-Interface von eZ publish 3.7.5

### 1.3.2 Content-Struktur, Klassen, Objekte, Attribute und Datentypen

Die meisten Content Management Systeme am Markt geben eine Struktur für die Daten fest vor. Der Endbenutzer hat dann meist keine Möglichkeit ohne größeren Aufwand diese zu verändern. eZ publish 3.\* geht an dieser Stelle einen anderen Weg. Es ermöglicht dem Webseiten-Administrator eine eigene Content-Struktur zu definieren. Dadurch ist es sehr einfach, benutzerdefinierte Daten zu strukturieren, zu speichern, wiederzufinden und zu präsentieren. Diese neue Möglichkeit der „dynamischen“ Strukturierung nähert sich stark der objektorientierten Denkweise. Objektorientiert denken bedeutet, die Umwelt in Form von Objekten zu sehen z.B. in einem Zimmer stehen Stühle, ein Tisch, ein Bett und eine Lampe. An Hand bestimmter Merkmale können diese Objekte identifiziert und einer bestimmten Objektklasse zugeordnet werden. Die objektorientierte Struktur wird in eZ publish mit folgenden Mitteln beschrieben (siehe Abbildung 1.3.2):

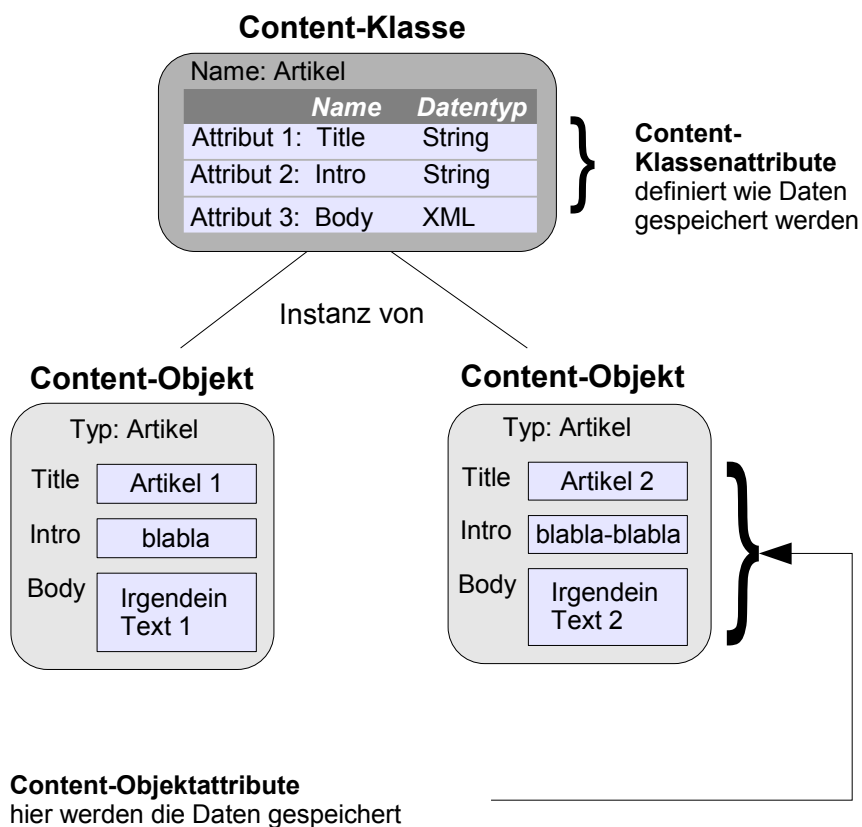


Abbildung 1.3.2 Content-Klassenobjekte und Content-Klassenattribute

1. Ein **Datentyp** bezeichnet in eZ publish 3.\* die kleinste mögliche Speichereinheit. eZ publish besitzt schon vordefinierte, grundlegende Datentypen wie z.B. „string“, „XML text“, „image“ und „binary file“ [eZDoc Appendix C]. Mit einem Datentyp wird die Speicherform eines Attributes einer Content-Klasse festgelegt. Wenn die vorhanden Art von Datentypen nicht ausreicht, ist es möglich, eigene Datentypen zu erstellen, was aber mit Programmieraufwand verbunden ist.
2. Eine **Content-Klasse** ist eine Definition einer willkürlichen Datenstruktur. Die Datenstruktur wird mit sogenannten **Content-Klassenattributen** (Merkmale) beschrieben. Jedes Content-Klassenattribut besitzt einen bestimmten Datentyp. Content-Klassen in eZ publish sind z.B. Ordner, Artikel, Bilder , siehe Content-Klassenliste im Internet [eZDoc Appendix D].  
Es ist zu beachten, dass eine Content-Klasse keine Daten speichert.
3. Ein **Content-Objekt** ist eine Instanz einer Content-Klasse. Es können mehrere Content-Objekte einer Content-Klasse existieren. Jedes Content-Objekt besteht aus unabhängigen **Content-Objektattributen**. Jedes dieser Attribute ist durch die Content-Klassenattribute definiert. In einem Content-Objekt werden alle Daten in einem oder mehreren Content-Objektattributen gekapselt.

### 1.3.3 Versionierung von Content-Objekten

Die Daten eines Content-Objektes können in eZ publish 3.\* in einer oder mehreren Versionen existieren. Bei jeder Datenänderung wird eine neue Version angelegt. Dabei bleibt die alte Version unberührt. Dadurch können alle Änderungen, auch von unterschiedlichen Benutzern, aufgezeichnet werden und es ist möglich, Änderungen rückgängig zu machen. Damit bei häufiger Datenänderung die Datenbank nicht zu stark anwächst, kann der Seiten-Administrator die maximale Anzahl an Versionen eines Objektes festlegen. Dies kann auf Content-Klassenebene durchgeführt werden.

Jede Content-Objektversion kann einen anderen Besitzer haben, aber nur eine Version kann den Status „veröffentlicht“ besitzen. Diese Version nennt man „aktuelle Version“. Die nachfolgende Abbildung 1.3.3 soll die Arbeitsweise des Versionierungssystems an Hand von Journalisten und Artikeln verdeutlichen.

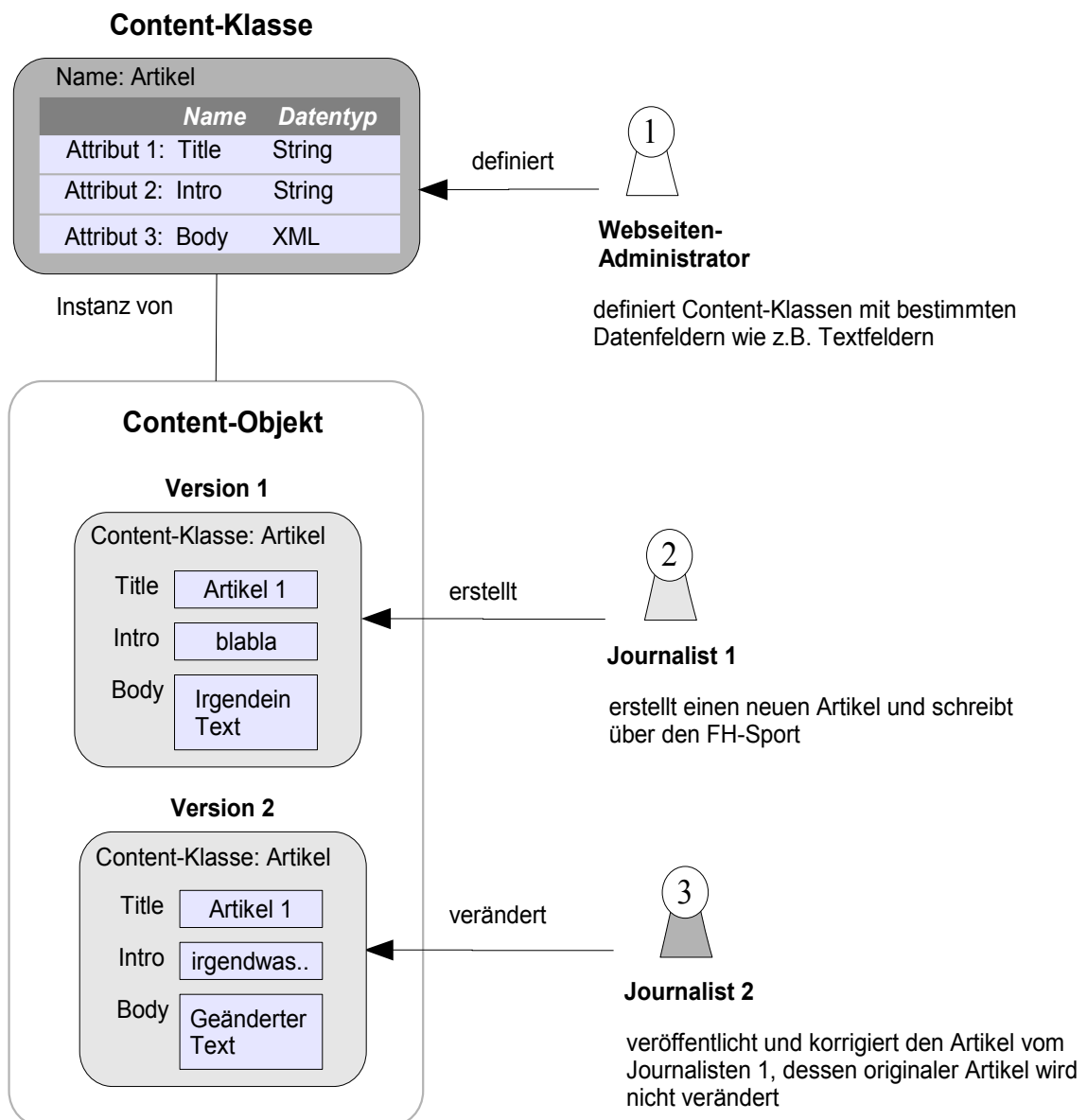


Abbildung 1.3.3 Versionierungssystem in eZ publish 3.\*

Fazit: Das in eZ publish 3.\* integrierte Versionierungssystem ist eine flexible und mächtige Funktionalität, die das Versionieren jeder Art von Content, d.h. auch von Bildern und binären Dateien, unabhängig vom Typ, von der Struktur und Größe ermöglicht.

### 1.3.4 Unterstützung mehrerer Sprachen

Zusätzlich zum Versionierungssystem unterstützt eZ publish 3.\* die Verwaltung von Content in mehreren Sprachen. Die Sprachverwaltung ist so zu sagen die dritte Dimension von einem Content-Objekt (1.Objekt, 2. Version, 3. Sprache). Jedes Content-Objekt kann dabei in mehreren Sprachen existieren. Diese sehr wichtige Funktionalität kann genau wie das Versionierungssystem mit jeder Art von Content genutzt werden. Die nachfolgende Abbildung 1.3.4 zeigt wie die Sprachverwaltung in das Versionierungssystem integriert ist.

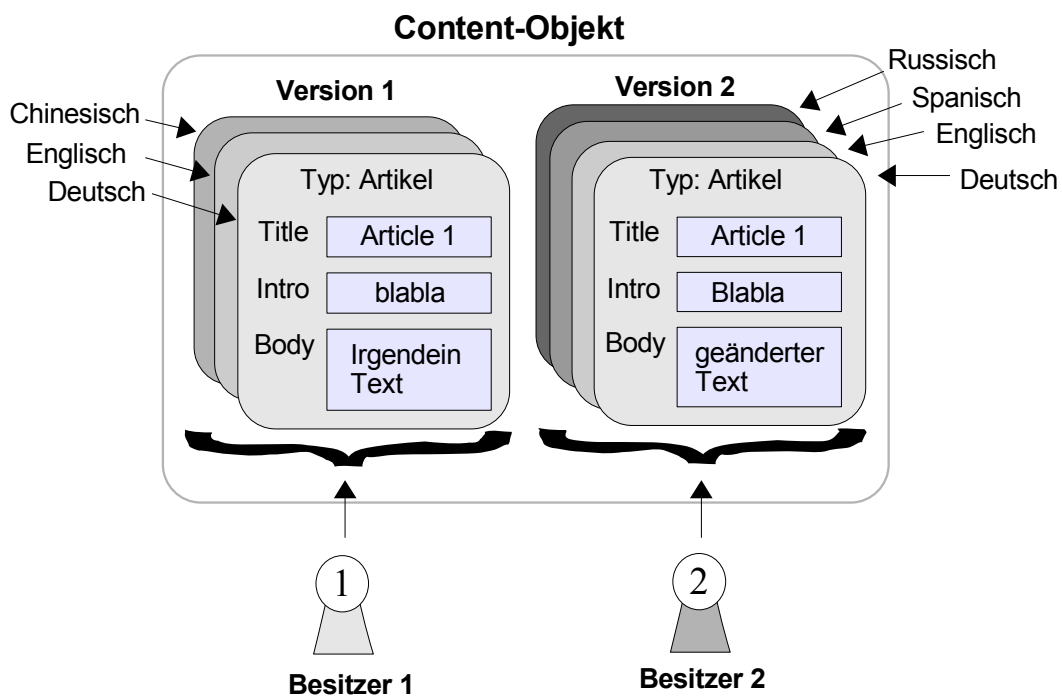


Abbildung 1.3.4 Integration Sprachverwaltung in das Versionierungssystem

### 1.3.5 Content-Knotenbaum und Knoten

Ein Content-Objekt ist eine Instanz von einer Content-Klasse. Während der Benutzung von eZ publish werden neue Content-Objekte erstellt. Beim Verfassen eines Artikels wird ein Content-Objekt erstellt, das die Daten des Artikels speichert. Dabei speichert das Content-Objekt auch alle verschiedenen Versionen und Sprachen des Artikels.

Content-Objekte werden in eZ publish mit Hilfe von Knoten (Nodes) und dem Content-Knotenbaum (Content Node Tree) strukturiert.

Ein **Knoten** ist vergleichbar mit einer Kurzfassung von einem Content-Objekt. Dieser enthält nur einen Zeiger (Pointer) auf das eigentliche Content-Objekt.

Der **Content-Knotenbaum** besteht aus Knoten. Dieser ist eine hierarchische Baumstruktur, die die gesamten Content eines eZ publish Systems repräsentiert. Anders ausgedrückt, mit diesem Mechanismus werden alle Content-Objekte in einem eZ publish System organisiert.

- Ein gängiger Weg, Content-Objekte zu kategorisieren, ist das Anlegen von Behältern (Containern) z.B. mit Ordnern siehe Abbildung 1.3.5. In diesen werden relevante Content-Objekte abgelegt. So ähnlich wird es auch im lokalen Dateisystem praktiziert, um alle mögliche Dateien zu organisieren.

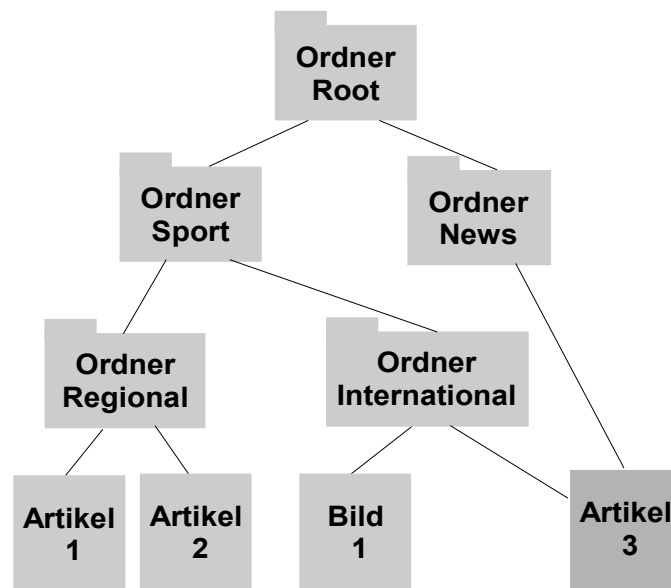


Abbildung 1.3.5 Organisieren von Content mit Ordnern (Behältern)

- Wenn im Administrations-Interface Ordner, Artikel, Bilder und Dateien verwaltet werden, wird mit dem Content-Knotenbaum interagiert. Der Content-Knotenbaum ist sehr nützlich, um eine Sitemap automatisiert zu erstellen.
- Die Abbildung 1.3.5 veranschaulicht, wie Knoten und Content-Objekte miteinander verknüpft sind. Jedes Blatt im Baum ist ein Knoten. Der Content-Knotenbaum besteht dabei mindestens aus einem Knoten, den Hauptknoten (Root Node). Dieser wird repräsentiert durch den Ordner „Root“, unter dem die gesamten Content-Objekte abgelegt werden.
- Jeder Knoten, außer der Hauptknoten besitzt zwei Zeiger. Einer zeigt auf den Vater-Knoten und einer zeigt auf das Content-Objekt.
- Ein Knoten kann immer nur auf einen Vater-Knoten und ein Content-Objekt zeigen. Es ist aber möglich, dass mehrere Knoten auf ein und dasselbe Content-Objekt zeigen. Das bedeutet, dass ein Content-Objekt an verschiedenen Orten im Content-Knotenbaum platziert sein kann.
- Ein nicht veröffentlichtes Content-Objekt (Entwurf) besitzt keinen Knoten im Content-Knotenbaum.

- Archivierte Content-Objekte verlieren ebenfalls Ihren Knoten im Baum. Im Content-Knotenbaum befinden sich also nur Content-Objekte, die veröffentlicht wurden. Die Anzahl der Verzweigungen (Breite und Tiefe) in der Baumstruktur ist unbegrenzt. Weil die Knoten im Grunde nur Zeiger auf Content-Objekte sind, können so beliebige Arten von Content-Objekten im Content-Knotenbaum organisiert werden.

### 1.3.6 Sektionen

Wie schon im vorhergehenden Abschnitt erwähnt, ist der Content-Knotenbaum eine hierarchisch aufgebaute Baumstruktur, die den gesamten Content eines veröffentlichten eZ publish Systems repräsentiert. Der Content wird hier durch Container (Behälter) wie z.B. Ordner strukturiert. Zusätzlich zu dieser hierarchischen Struktur, kann der Baum in logische Sektionen zerlegt werden, siehe Abbildung 1.3.6. **Sektionen** werden dazu genutzt, den Content-Knotenbaum aufzuteilen.

Sektionen machen es einfach:

- Benutzerdefinierte Templates zu nutzen
- Benutzerzugriffe auf Content-Objekte zu kontrollieren bzw. zu erlauben

Die Benutzung der Sektionen erfolgt auf Content-Objektebene. Jedes Content-Objekt kann nur einer Sektion angehören. Standardmäßig gehört ein Content-Objekt der Sektion Standard-Sektion an.

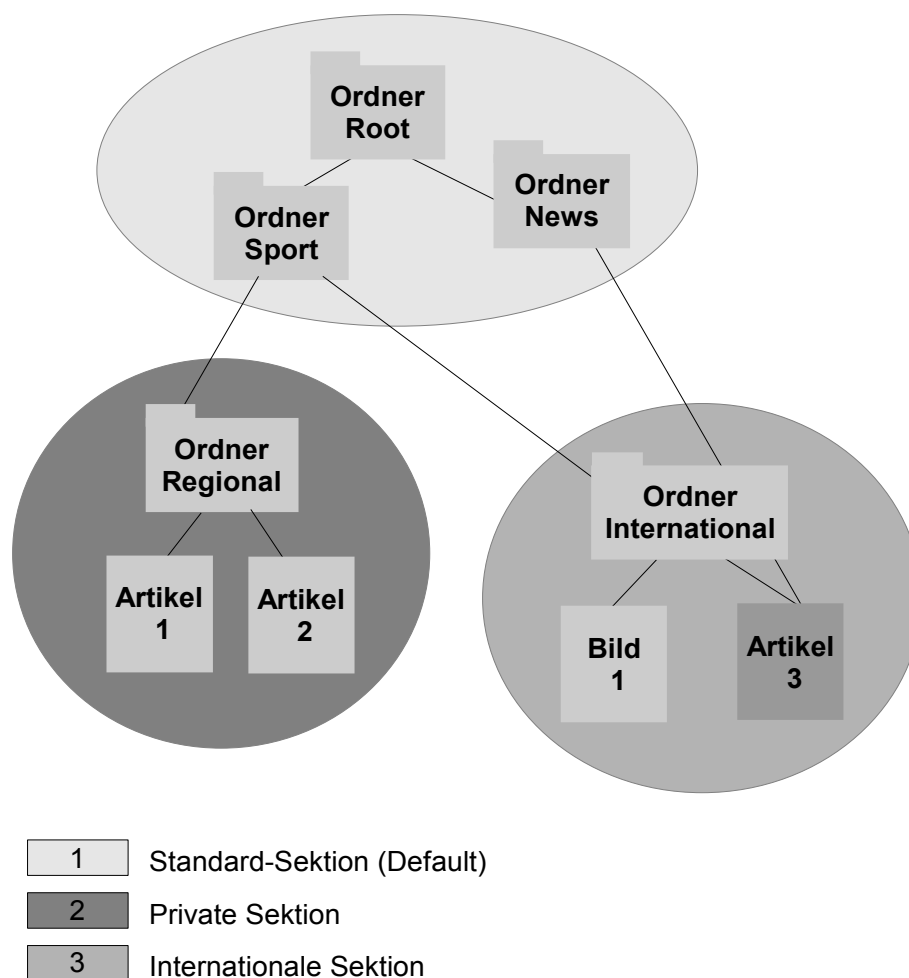


Abbildung 1.3.6 Abgrenzung von Content durch Sektionen

Das Administrations-Interface ermöglicht eine sehr einfache Nutzung von Sektionen. Sektionen werden dabei Knoten zugeordnet. Immer wenn einem Knoten eine Sektion zugeordnet wird, erhalten alle Kind-Knoten die selbe zugewiesen. Wenn ein neues Content-Objekt erstellt wird, dann wird diesem die Sektion des Vater-Knotens zugewiesen.

## 1.4 Site Management in eZ publish 3.\*

Mit eZ publish 3.\* ist es möglich mehrere verschiedene Sites unter einer Installation zu betreiben. Eine Site ist die Gesamtheit einer Konfiguration und aller Inhalte (Content). Sie kann dabei auf unterschiedliche Art und Weise angelegt werden. Es können z.B. mehrere Designs mit verschiedenen Teilen (z.B. über Sektionen) einer Site verknüpft werden.

Dieses Kapitel soll einen kurzen Einblick darüber geben, wie eZ publish Sites und Site Interfaces verwaltet und wie benutzerdefinierte Templates eingerichtet werden.

## 1.5 Site

Der Ausdruck Site beinhaltet alles was mit einer speziellen Webseite in Verbindung steht:

- Konfigurationseinstellungen
- Eine Datenbank, die die Content-Struktur und die aktuellen Content-Daten enthält
- Dateien, die mit dem Content verknüpft sind, wie z.B. Bilder
- Dateien, die zu einem Design gehören

### 1.5.1 Site Interface

Der Content einer Site kann auf unterschiedliche Art und Weise angezeigt und verändert werden. Ein Site Interface regelt dabei, welches spezielle Design genutzt werden soll, wenn auf einem bestimmten Weg auf eine Site zugegriffen wird. Folgende Interfaces besitzt eine Site mindestens:

- Administrations-Interface, für die Site-Verwaltung
- Benutzer-Interface, für den Benutzer, der sich die Inhalte einer Site ansehen möchte

### 1.5.2 Site Access

Für die Verknüpfung von einem Site Interface mit einer Site, kommt ein Site Access zum Einsatz. Die Konfiguration von einem Site Access beinhaltet:

- Wie eZ publish die Site bzw. das Site Interface erkennt, auf das gerade zugegriffen wird.
- Verschiedene Konfigurationseinstellungen, die die Standardeinstellungen überschreiben z.B. die Datenbank- und Design-Einstellungen durch benutzerdefinierte Templates

Informationen, wie ein Site Access eingerichtet wird, gibt es in der Online-Dokumentation von eZ publish im Kapitel Configurations, siehe [eZDoc Site access].

Für das Einrichten einer mehrsprachigen Site, sind Site Accesses eine Möglichkeit. So kann z.B. über verschiedene Site Accesses die Sprache für die Anzeige des Contents gesteuert werden (Datenbank und das Site Design bleiben dabei die selben) [eZDoc MultiLanguage].  
Abbildung 1.5.1 veranschaulicht den Sachverhalt noch einmal.

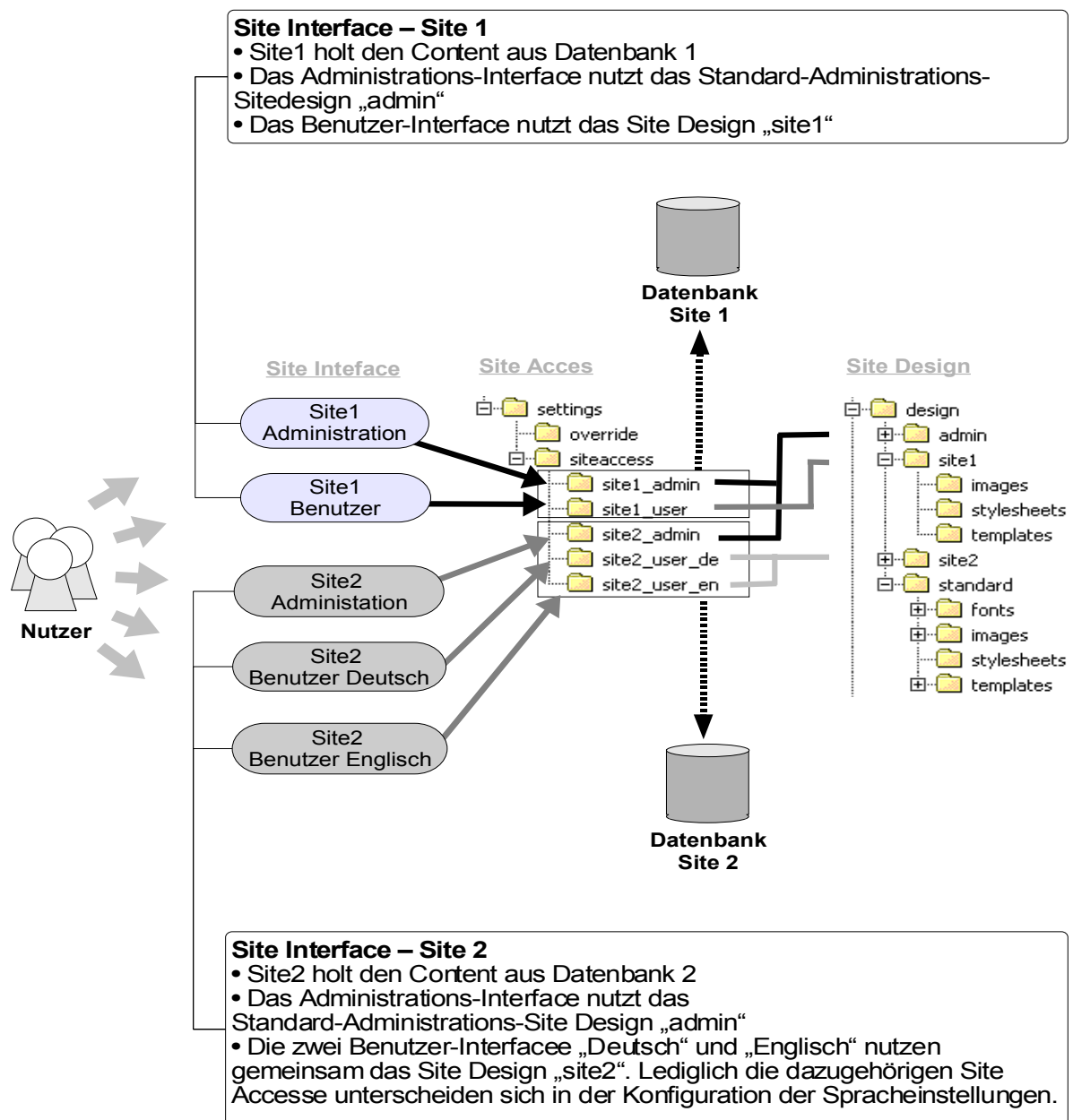


Abbildung 1.5.1 Zusammenhang Site - Site Interface – Site Access - Site Design

Wie schon erwähnt, enthält ein Site Access mehrere Konfigurationsdateien, die sogenannten „\*.ini.append“ Dateien. Die wichtigsten Konfigurationsdateien sind:

- „site.ini.append“ - Datenbank-, Site Design- und Sprach-Konfiguration
- „override.ini.append“ - Festlegung, welches Template aus „design/{sitename}/templates/“, beim Site-Zugriff über ein Interface, die Standard-Templates aus „design/standard/templates/“ überschreiben soll. Die Template-Konfiguration kann auf Sektions-, Content-Klassen- und Content-Objektebene erfolgen. Nachfolgendes Beispiel zeigt eine Definition für ein benutzerdefiniertes Template (.../templates/full\_view\_article.tpl), das angezeigt wird, wenn auf ein Content-Objekt vom Content-Klassentyp „article“ (class=2) in der Standard-Sektion (section=1) zugegriffen wird.

```
[full_view_article]
Source=node/view/full.tpl
MatchFile=full_view_article.tpl
Subdir=templates
Match[class]=2
Match[section]=1
```

Informationen, wie benutzerdefinierte Templates eingerichtet werden, gibt es im Internet unter [eZDoc Override] und [eZDoc Custom Template].

## 1.6 URLs in eZ publish 3.\*

In diesem Abschnitt werden kurz die verschiedenen Zugriffsmöglichkeiten mit URLs<sup>3</sup> auf das eZ publish System erklärt. Im Wesentlichen gibt es 2 Arten von URLs in eZ publish 3.\*:

1. System-URL
2. Virtuelle-URL

### 1.6.1 System-URL

Die System-URL ist die unbearbeitete URL. Diese enthält verschiedene Informationen darüber, auf welche Inhalte und Funktionen zugegriffen wird. Eine System-URL sieht z.B. wie folgt aus:

```
http://www.example.com/content/view/full/44
```

---

3 Uniform Resource Locator (URL) - Adresse eines Objekts im Internet [NetLexikon 04]

## 1.6.2 Virtuelle-URL

Eine Virtuelle-URL ist eine vereinfachte Form einer System-URL. Diese ist schöner, einfacher und meistens auch kürzer als die entsprechende System-URL. Eine Virtuelle-URL wird häufig auch als „URL-Alias“ bezeichnet:

```
http://www.example.com/news
```

Virtuelle-URL werden vom eZ publish teilweise automatisch erstellt. Wird z.B. auf einen Artikel „Neuigkeiten vom Montag“ (NodeID 55) im Ordner „news“ zugegriffen, so wird folgende Virtuelle-URL generiert:

```
http://www.example.com/news/neuigkeiten_vom_montag
```

Mit folgender System-URL wird der gleiche Artikel angezeigt:

```
http://www.example.com/content/view/full/55
```

Manuell ist es ebenfalls möglich, eine Virtuelle-URL einzurichten. Dazu gibt es im Administrations-Interface den URL-Übersetzer. So ist es z.B. sinnvoll, für ein Kontakt-Formular (Knoten-ID 60) eine verkürzte Virtuelle-URL einzurichten:

```
http://www.example.com/content/view/full/60 (System-URL) = wird zu =>  
http://www.example.com/kontaktformular (Virtuelle-URL)
```

Wird diese verkürzte Virtuelle-URL in mehreren verschiedenen Templates manuell als statischer Link eingebunden, sind spätere Änderungen einfach durchführbar. Wenn z.B. ein neues Kontakt-Formular genutzt werden soll, ist nur die Verknüpfung von System-URL mit der Virtuellen-URL im Administrations-Interface zu aktualisieren (keine Änderungen an den Templates notwendig!). Die Abbildung 1.6.1 veranschaulicht den Aufbau von System- und Virtuellen-URLs.

Der Webserver-spezifische Teil der URL (Abbildung 1.6.1) ist abhängig von der Webserver-Konfiguration (z.B. Apache httpd.conf). Es ist möglich, mit sogenannten „Virtual Hosts“, alle Anfragen direkt auf die „index.php“ im Hauptverzeichnis von eZ publish zu leiten, siehe URL 2 und 3 in Abbildung 1.6.1. Je nachdem wie die Webserver-Konfiguration ist, muss die Konfigurationsdatei (site.ini.append.php) angepasst werden. Folgende Möglichkeiten für die Konfiguration gibt es :

- **URL:** Pfad zum eZ publish -Hauptverzeichnis

```
http://www.example.com/ezroot
```

- **Hostname und Port:** Zugriff über „Virtual Host“

```
http://www.example.com oder http://www.example.com:8080
```

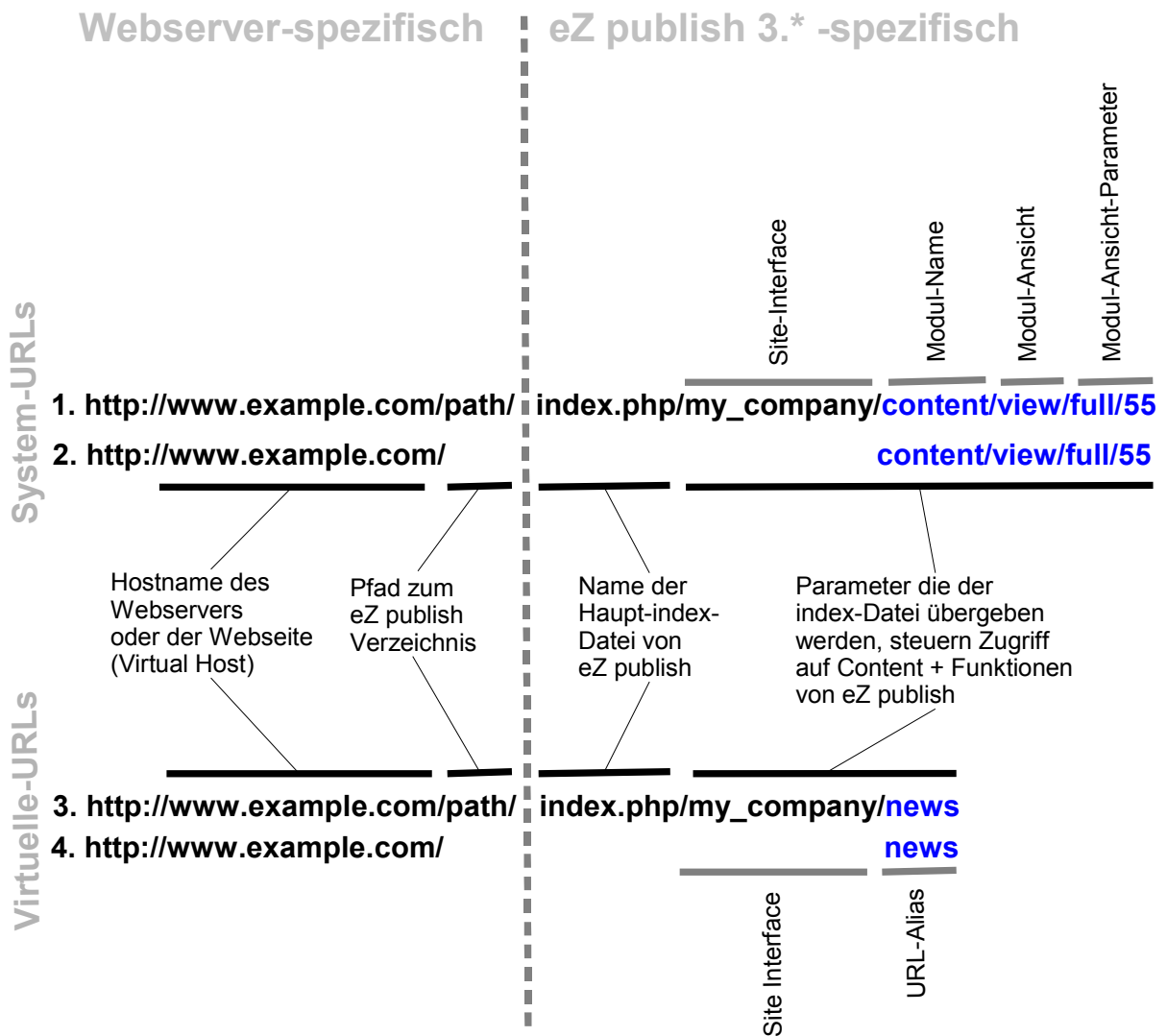


Abbildung 1.6.1 Aufbau System-URLs und Virtuelle-URLs in eZ publish 3.\*

Standardmäßig verwendet eZ publish in einem Site Access den Zugriffstyp URL, siehe URL Nr. 1 und 3 in Abbildung 1.6.1. Wie die Konfigurationsdateien jeweils aussehen müssen, kann im Internet unter [eZDoc Site access Config] nachgelesen werden.









## 2 Grundlagen der Modulprogrammierung

In diesem Kapitel werden grundlegende Mechanismen beschrieben, die bei der Programmierung von Erweiterungen unter eZ publish verwendet werden können. Es wird auch kurz darauf eingegangen welche Konfigurationen in eZ publish vorgenommen werden müssen, um die Erweiterungen nutzen zu können.

### 2.1 Struktureller Aufbau einer Extension / eines Moduls

In eZ publish werden Funktionserweiterungen als Extension bezeichnet. Eine Extension kann verschiedene Funktionsgruppen, die als Module bezeichnet werden, enthalten. Jedoch existiert in den meisten Fällen nur eine Funktionsgruppe. Der Aufbau einer Extension wird von eZ publish vorgegeben. Die Tabelle 2.1.1 zeigt den allgemeinen Aufbau einer Extension in eZ publish. [eZDoc extension]

Tabelle 2.1.1 eZ publish 3.\* Extension-Verzeichnisstruktur

 <b>extension</b>	
 <b>actions</b>	Enthält neue Aktionen für Formulare
 <b>datatypes</b>	Enthält Definitionen für neue Datentypen
 <b>design</b>	Enthält alles, was das Design der „extension“ beeinflusst
 <b>eventtypes</b>	Enthält neue Event-Arten für Workflows (Arbeitsabläufe)
 <b>modules</b>	Enthält PHP-Klassen und Scripte der einzelnen Funktionsgruppen der „extension“
 <b>settings</b>	Enthält Einstellungen und Konfigurationsdateien der „extension“
 <b>translations</b>	Enthält Übersetzungsdateien der „extension“ die automatisch durch eZ publish erstellt werden können

Typisch ist die Nutzung der Ordner „settings“, „design“, „module“ und „translations“.

## 2.2 Einrichten einer Extension

Dieser Abschnitt beschreibt am Beispiel der Extension „mapsystem“, welche Einstellungen vorgenommen werden müssen, damit diese genutzt werden kann. Die Abbildung 2.2.1 gibt einen Überblick über die wichtigsten Konfigurationsdateien und die vorzunehmenden Änderungen, um eine Extension mit Modulen in eZ publish zu aktivieren.

(Quelle [eZDoc extension], [eZDoc Hello World], [eZDoc module])

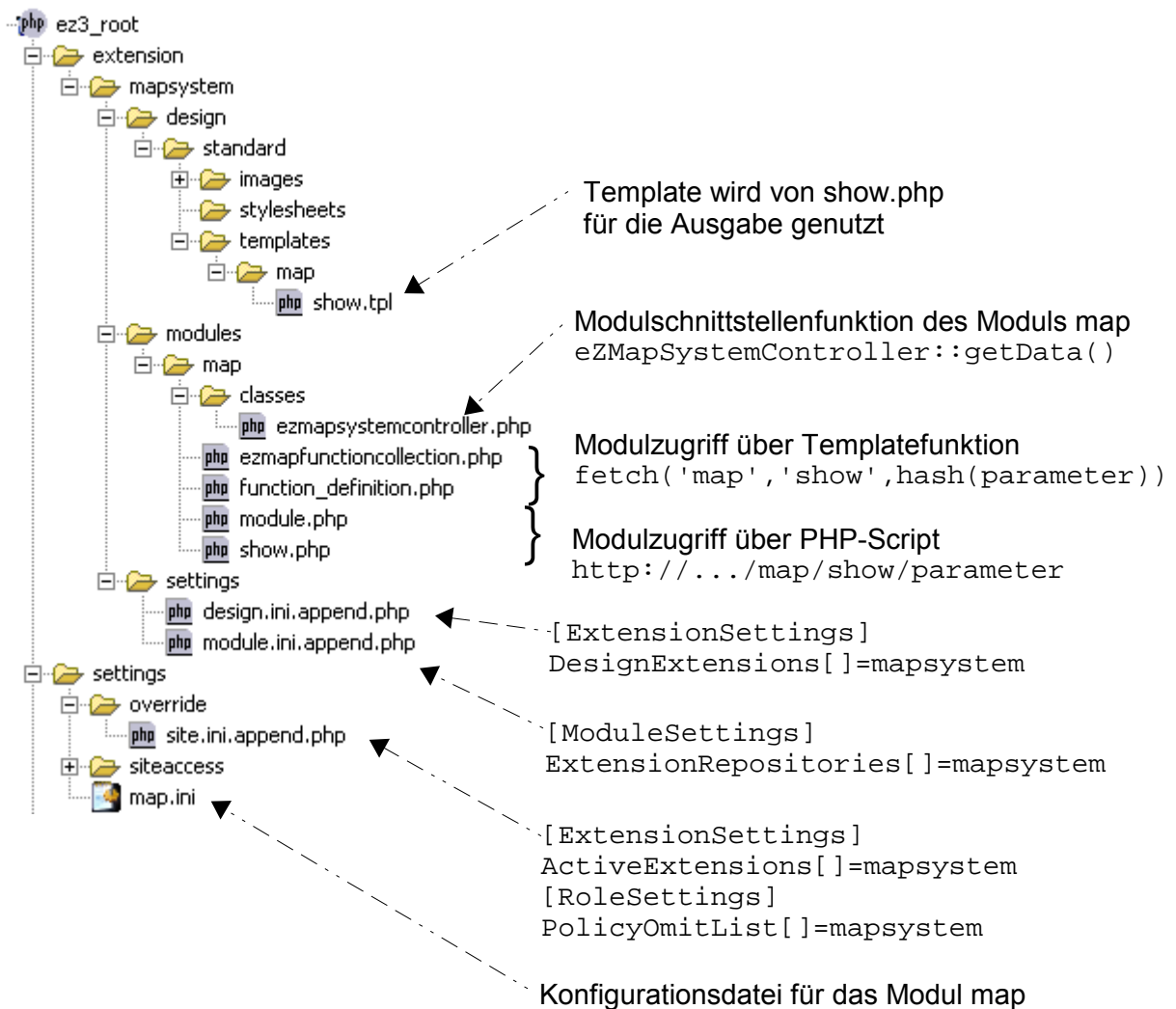


Abbildung 2.2.1 Aufbau und Konfiguration einer Extension am Beispiel von "mapsystem"

Die nachfolgende Liste beschreibt die wichtigsten Konfigurationsdateien, die mindestens erforderlich sind, um eine Extension zu aktivieren und zu nutzen:

- **extension/mapsystem/settings/ module.ini.append.php**

Diese Konfigurationsdatei teilt dem eZ publish System mit, dass die Extension „mapsystem“ verfügbar ist. Daraufhin versucht eZ publish alle Module wie z.B. „map“ , die sich im Verzeichnis „extension/mapsystem/modules/“ befinden, zu laden.

```
[ModuleSettings]
ExtensionRepositories[]=mapsystem
```

- **extension/mapsystem/modules/map/ module.php**

Um den Zugriff per URL auf ein eigenes PHP-Script z.B. „show.php“ im Modul „map“ zu definieren, dient die PHP-Datei „module.php“:

```
<? php
$Module = array( „name“ => „Dynamic Map“ );
$ViewList = array();
$ViewList[„show“] = array( „script“ => „show.php“
                          „params“ => array() );
?>
```

Das gezeigte Beispiel bewirkt, dass über die nachfolgende URL das Script „show.php“ der Extension „mapsystem“ und dem Modul „map“ mit bestimmten Parametern aufgerufen wird:

```
http://.../map/show/?parameter1=value1&parameter2=value2
```

- **extension/mapsystem/modules/map/ show.php**

Das PHP-Script „show.php“ kann die übergebenen Parameter auswerten und bestimmte Aktionen ausführen z.B. externe Funktionen aus Klassenbibliotheken aufrufen und die Ergebnisse entsprechend auswerten. Im nachfolgenden Scriptbeispiel werden zwei übergebene Variablen (parameter1 und parameter2) ausgelesen und als einfacher Text ausgegeben. Die Ausgabe kann auch über ein Template erfolgen. Dies kann in der eZ publish Dokumentation nachgelesen werden.

```

<? php
$Module =& $Params['Module'];
$param1 = $Params['parameter1'];
$param2 = $Params['parameter2'];
$text = „Variable 1=". $param1. „ Variable2". $param2;
$Result = array();
$Result['content'] =& $text;
$Result['path'] = array( array('url' => false,
                             'text'=> 'Exampletext from show.php')); ?>

```

- **extension/mapssystem/settings/ design.ini.append.php**

Soll die Extension ein eigenes Design erhalten, so muss unter dem Verzeichnis der Extension das Verzeichnis „design“ angelegt werden. Dieses ist genauso aufgebaut wie das eZ publish Haupt-Design-Verzeichnis. Folgender Konfigurationseintrag in die „design.ini.append.php“ ist nötig damit eZ publish das Design-Verzeichnis mit durchsucht.

```

[ExtensionSettings]
DesignExtensions[]=mapssystem

```

[eZDoc design]

- **settings/override/ site.ini.append.php**

Damit die Extension „mapssystem“ genutzt werden kann, muss diese aktiviert werden. Folgender Eintrag in der „site.ini.append.php“ aktiviert die Extension global für alle Site Accesses:

```

[ExtensionSettings]
ActiveExtensions[]=mapssystem

```

Falls das Rechtesystem von eZ publish genutzt werden soll, ist folgender Eintrag nötig:

```

[RoleSettings]
PolicyOmitList[]= mapssystem

```

- oder **settings/siteaccess/mysiteaccess/ site.ini.append.php**

Soll die Extension nur für bestimmte Site Accesses einer eZ publish Installation freigeschaltet werden, dann muss der Eintrag folgendermaßen lauten:

```

[ExtensionSettings]
ActiveAccessExtensions[]= mapssystem

```

## 2.3 Modulzugriffsmechanismen

In eZ publish ist es möglich die Modulfunktionalität grundsätzlich über zwei Wege zu nutzen. Neben dem direkten Aufruf über ein PHP-Script (URL mit Parametern) ist es auch möglich, die Modulfunktionalität dem integrierten Template-System bereitzustellen.

- **Zugriff über ein PHP-Script**

Der gebräuchlichste Zugriff auf die Funktionen eines Moduls geschieht über PHP-Scripte. Diesen können beim Aufruf verschiedene Parameter übergeben werden. Die Konfiguration erfolgt über die „module.php“ Datei, die in jedem Modulverzeichnis vorhanden sein muss. In dieser wird definiert über welche URL ein bestimmtes PHP-Script aufgerufen wird. So können z.B. die Übergabeparameter bei entsprechender Konfiguration direkt in die URL integriert sein oder einfach angehängt werden.

```
http://www.example.com/map/show/1/2/3
```

```
http://www.example.com/map/show/?par1=1&par2=2&par3=3
```

- **Zugriff direkt aus einem Template**

Eine andere interessante Möglichkeit ist der Zugriff auf die Modulfunktionalität direkt aus einem Template von eZ publish. Das hat den Vorteil, dass die Funktionalität ohne explizitem URL-Aufruf in das Layout eingebunden werden kann. Bezogen auf das Kartenmodul, könnte über diesen Weg z.B. eine kleine Übersichtskarte in einer Ortsbeschreibung automatisch mit integriert werden, da über die Nodeld des Content-Objektes die Position aus der Datenbank ermittelt und visualisiert werden kann.

Für die Bereitstellung bestimmter Modulfunktionalitäten dient in eZ publish die Datei „function\_definition.php“. In dieser wird definiert, auf welche Funktion einer Funktionsklasse, wie z.B. „ezmapfunctioncollection.php“, und mit welchen Parametern nach folgendem Schema zugegriffen werden kann:

```
fetch('Modulename', 'Funktionsname', hash( 'parameter1' => wert1,  
                                           'parameter2' => wert2,  
                                           ... ));
```

Die Konfiguration der Template-Funktion ist relativ schlecht dokumentiert. Da aber viele Module wie z.B. das sehr komplexe „content“-Modul über diese Zugriffsmöglichkeit verfügen, kann die entsprechende „function\_definition.php“ als Vorlage bzw. als Anregung genutzt werden.

Die Abbildung 2.3.1 versucht einmal den Aufrufprozess der zwei verschiedenen Modulzugriffe darzustellen.

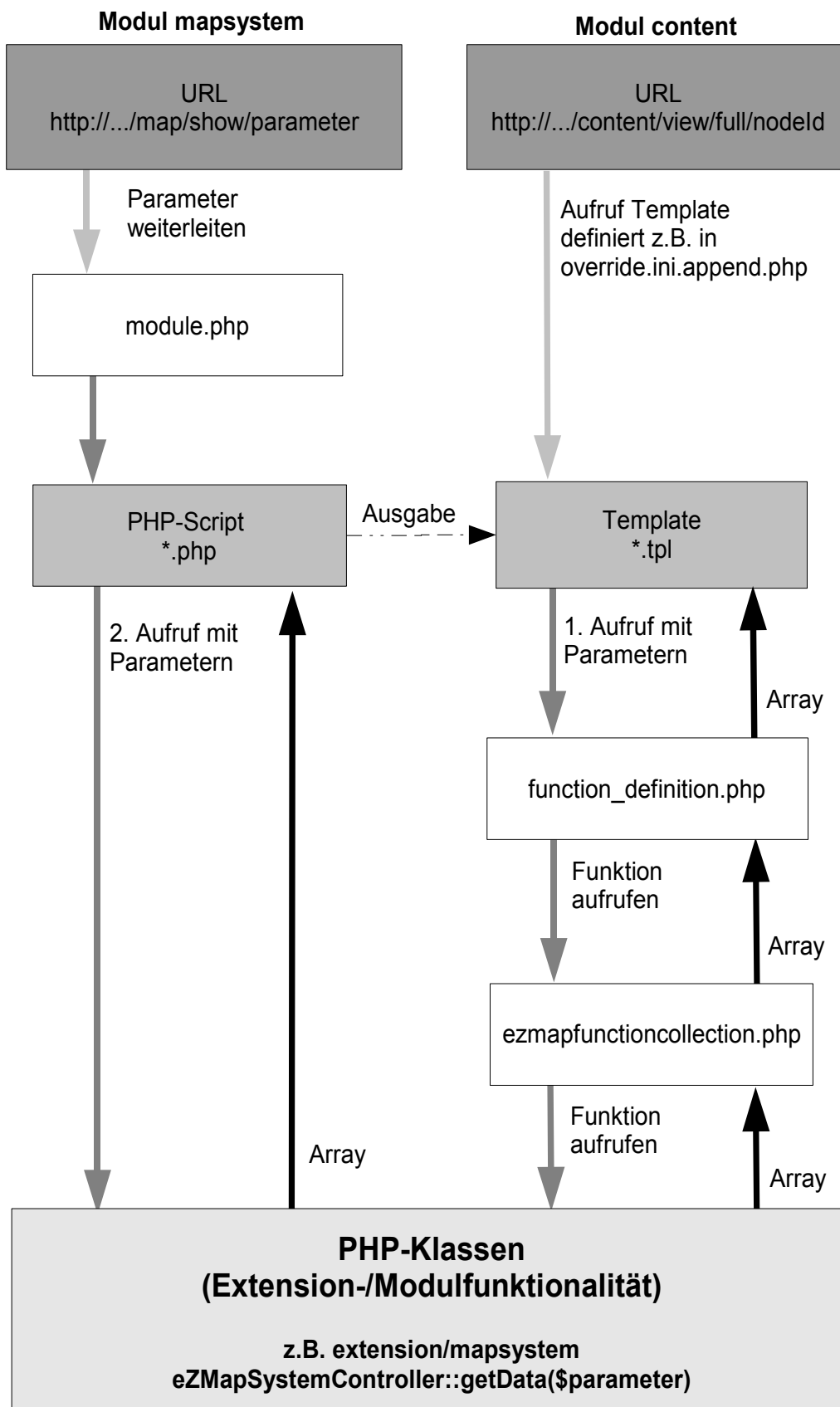


Abbildung 2.3.1 Zugriffsmöglichkeiten auf Modulfunktionalitäten in eZ publish

## 2.4 Zugriffsrechte auf ein Modul

Es ist möglich, Module einer Extension mit speziellen Zugriffsberechtigungen zu versehen. Dazu werden die Modulfunktionalitäten bestimmten selbstdefinierten Funktionsgruppen wie z.B. „read“, „edit“ und „admin“ zugeordnet. Für das Verhalten der Funktionsgruppen ist der Entwickler selbst verantwortlich. So soll z.B. bei „read“ nur der Lesezugriff erlaubt werden. Im Administrations-Interface werden dann bestimmten Nutzergruppen Rechte auf diese Funktionsgruppen zugeordnet.

Die „module.php“ für das Modul „map“ der Erweiterung „mapsystem“ sieht vereinfacht folgendermaßen aus:

```
<?php
$Module = array( "name" => "MyModule" );

$ViewList = array();
$ViewList["show"] = array( 'script' => 'show.php',
                           'functions' => array( 'read' ),
                           'params' => array() );

$ViewList['positionedit'] = array( 'script' => 'positionedit.php',
                                   'functions' => array( 'edit' ),
                                   'params' => array( ) );

$ViewList['cut'] = array( 'script' => 'cut.php',
                          'functions' => array( 'admin' ),
                          'params' => array( ) );

$FunctionList['read'] = array( ); // Karten anzeigen
$FunctionList['edit'] = array( ); // Positionsobjekte verwalten
$FunctionList['admin'] = array( ); // Karte konfigurieren, initialisieren
?>
```

Damit eZ publish überprüft, ob der aktuelle Benutzer die Berechtigung hat, eine bestimmte Modulfunktionalität der Extension „mapsystem“ aufzurufen, muss noch folgender Eintrag in die globale „site.ini.append.php“ eingefügt werden:

```
[RoleSettings]
PolicyOmitList[]= mapsystem
```

[eZDoc Permissions]

## 2.5 INI-Dateien

In eZ publish werden sogenannte INI-Dateien für die Konfiguration des Systems und der einzelnen Module verwendet. In diesen können Variablen definiert werden, die später in Templates oder in PHP-Skripten sehr einfach ausgelesen werden können. Die INI-Dateien kann man dabei in zwei Gruppen einteilen:

1. Die erste Gruppe bilden die **\*.ini** Dateien. Diese beinhalten alle Variablen und weisen diesen Standardwerte zu. Diese Dateien befinden sich alle unter dem Verzeichnis „settings“.
2. Die zweite Gruppe bilden die **\*.ini.append.php** Dateien. Diese Dateien überschreiben Variablenwerte aus den \*.ini Dateien. Dabei kann es mehrere verschiedene \*.ini.append.php eines gleichen Typs z.B. „site.ini.append.php“ geben, die in unterschiedliche Verzeichnissen z.B. im Extension- oder Site Access-Verzeichnis liegen können. Dabei durchsucht eZ publish diese in einer bestimmten Reihenfolge, wie in Abbildung 2.5.1 veranschaulicht wird. Wird nach einer Variablen gesucht, wird der erste Fundort genommen d.h. wird die Variable unter 1. nicht gefunden, dann wird unter 2. weitergesucht usw.

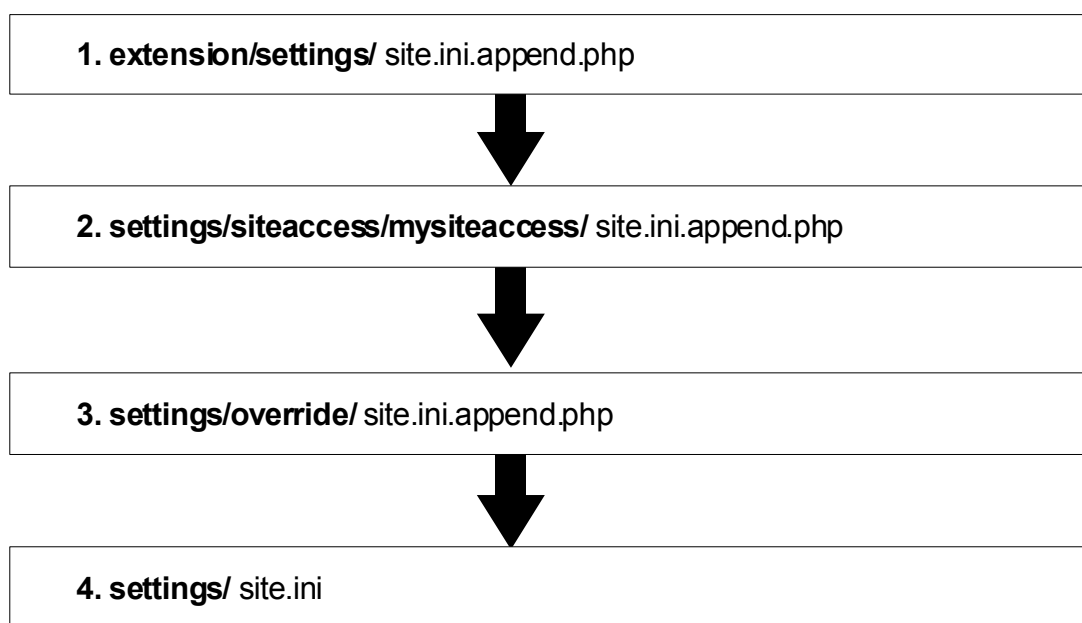


Abbildung 2.5.1 Reihenfolge Parsen von Konfigurationsdateien (INI-Dateien)

Nachfolgendes Beispiel zeigt, wie eine Variable in der „site.ini.append.php“ gesetzt wird:

```
[DatabaseSettings]
Database=mymysqldbname
```

und wie diese in einem PHP-Script wieder ausgelesen werden kann:

```
<?php
$mapINI =& eZINI::instance( 'site.ini' );
$mapTplArr = $mapINI->variable( 'DatabaseSettings', 'Database' );
?>
```

## 2.6 Template-System

Ähnlich wie bei den INI-Dateien bietet das Template-System von eZ publish die Möglichkeit, vorhandene Standardtemplates zu überschreiben. Das Überschreiben kann über zwei Möglichkeiten erfolgen:

1. durch Templates mit gleichem Namen, siehe dazu Abbildung 2.6.1 2., 3. und 4. Die Templates werden dabei immer ersetzt, im Gegensatz zur zweiten Möglichkeit.
2. durch einen Eintrag in der Datei „override.ini.append.php“ in einem Site Access. Diese Möglichkeit ist viel flexibler, da die Überschreibungsregeln abhängig gemacht werden können von Sektionen, Content-Klassen oder auch einzelnen Content-Objekten.

Die nachfolgende Abbildung 2.6.1 zeigt einmal die Reihenfolge, in der eZ publish nach Templates sucht. Wird z.B. an dem Ort 1. das gesuchte Template nicht gefunden, wird unter 2. nachgeschaut usw.

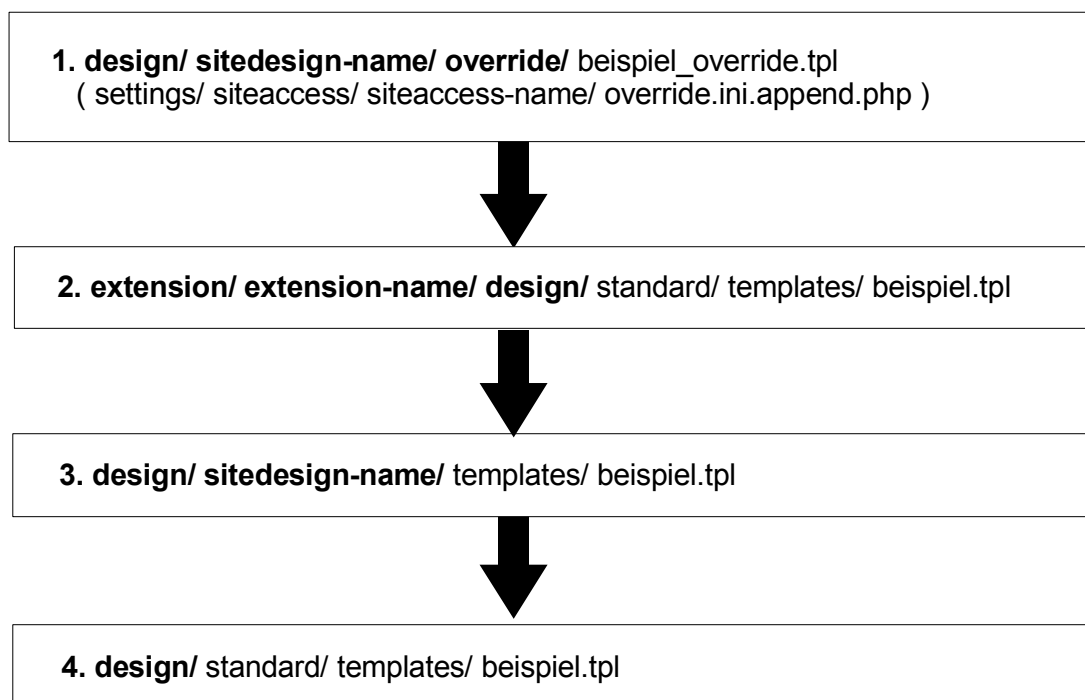


Abbildung 2.6.1 Suchreihenfolge vom Template Parser in eZ publish

## **3 Verzeichnisse**

### **3.1 Tabellenverzeichnis**

Tabelle 1.1.1 eZ publish 3.* Verzeichnisstruktur.....	4
Tabelle 2.1.1 eZ publish 3.* Extension-Verzeichnisstruktur.....	22

### **3.2 Abkürzungsverzeichnis**

PDF - Portable Document Format

### 3.3 Literaturverzeichnis

Als Basis für diese Diplomarbeit wurde die Dokumentation von eZ publish genutzt. Diese ist zu finden im Internet unter [http://ez.no/ez\\_publish/documentation](http://ez.no/ez_publish/documentation). Die eZ publish Community war ebenfalls sehr hilfreich <http://ez.no/community>, insbesondere die Foren. Nachfolgend sind die wichtigsten URLs zu den verwendeten Dokumenten und Dateien aufgelistet:

#### Literaturverzeichnis

- [**eZDoc Admin**] : The admin interface, eZ publish, 28.05.2004/ 27.08.2003,  
URL:[http://www.ez.no/ez\\_publish/documentation/day\\_to\\_day\\_use/the\\_admin\\_interface](http://www.ez.no/ez_publish/documentation/day_to_day_use/the_admin_interface)
- [**eZDoc Appendix A**] : Appendix A eZ publish libraries, eZ Publish, 07.06.2004/  
14.10.2003,  
URL:[http://www.ez.no/ez\\_publish/documentation/incoming/appendices/appendix\\_a\\_ez\\_publish\\_libraries](http://www.ez.no/ez_publish/documentation/incoming/appendices/appendix_a_ez_publish_libraries)
- [**eZDoc Appendix B**] : Appendix B - eZ publish modules, eZ publish, 07.06.2004/  
14.10.2003,  
URL:[http://www.ez.no/ez\\_publish/documentation/incoming/appendices/appendix\\_b\\_ez\\_publish\\_modules](http://www.ez.no/ez_publish/documentation/incoming/appendices/appendix_b_ez_publish_modules)
- [**eZDoc Appendix C**] : Appendix C - eZ publish datatypes, eZ publish, 28.05.2003/  
14.10.2003,  
URL:[http://www.ez.no/ez\\_publish/documentation/incoming/appendices/appendix\\_c\\_ez\\_publish\\_datatypes](http://www.ez.no/ez_publish/documentation/incoming/appendices/appendix_c_ez_publish_datatypes)
- [**eZDoc Appendix D**] : Appendix D - eZ publish content classes, eZ publish, 28.05.2004/  
14.10.2003,  
URL:[http://www.ez.no/ez\\_publish/documentation/incoming/appendices/appendix\\_d\\_ez\\_publish\\_content\\_classes](http://www.ez.no/ez_publish/documentation/incoming/appendices/appendix_d_ez_publish_content_classes)
- [**eZDoc Appendix E**] : Appendix E - eZ publish template operators, eZ publish, 28.05.2004/  
31.01.2004,  
URL:[http://www.ez.no/ez\\_publish/documentation/incoming/appendices/appendix\\_e\\_ez\\_publish\\_template\\_operators](http://www.ez.no/ez_publish/documentation/incoming/appendices/appendix_e_ez_publish_template_operators)
- [**eZDoc Appendix F**] : Appendix F - eZ publish template functions, eZ publish, 28.05.2004/  
31.01.2004,  
URL:[http://www.ez.no/ez\\_publish/documentation/incoming/appendices/appendix\\_f\\_ez\\_publish\\_template\\_functions](http://www.ez.no/ez_publish/documentation/incoming/appendices/appendix_f_ez_publish_template_functions)
- [**eZDoc Basics**] : eZ publish basics, eZ publish, 07.04.2004/ ,  
URL:<http://www.ez.no/content/download/48192/124980/file/basics.pdf>
- [**eZDoc Custom Template**] : Creating and using a custom template, eZ publish,  
28.05.2004/ 24.10.2003,  
URL:[http://www.ez.no/ez\\_publish/documentation/building\\_an\\_ez\\_publish\\_site/the\\_welcome\\_page/creating\\_and\\_using\\_a\\_custom\\_template](http://www.ez.no/ez_publish/documentation/building_an_ez_publish_site/the_welcome_page/creating_and_using_a_custom_template)
- [**eZDoc design**] : Design extension, eZ Crew, 14.09.2004/ ,  
URL:[http://www.ez.no/ez\\_publish/documentation/development/extensions/design\\_extension](http://www.ez.no/ez_publish/documentation/development/extensions/design_extension)
- [**eZDoc extension**] : Introduction to extensions, eZ Crew, 14.09.2004/ ,

URL:[http://www.ez.no/ez\\_publish/documentation/development/extensions/introduction\\_to\\_extensions](http://www.ez.no/ez_publish/documentation/development/extensions/introduction_to_extensions)

**[eZDoc Hello World]** : Hello World, eZ Crew, 14.09.2004/ ,

URL:[http://www.ez.no/ez\\_publish/documentation/development/extensions/module/hello\\_world](http://www.ez.no/ez_publish/documentation/development/extensions/module/hello_world)

**[eZDoc module]** : Building an eZ publish module, eZ Crew, 14.09.2004/ ,

URL:[http://www.ez.no/ez\\_publish/documentation/development/extensions/building\\_an\\_ez\\_publish\\_module](http://www.ez.no/ez_publish/documentation/development/extensions/building_an_ez_publish_module)

**[eZDoc MultiLanguage]** : Multilanguage site, eZ publish, 28.05.2004/ 04.07.2003,

URL:[http://ez.no/ez\\_publish/documentation/configuration/configuration/language\\_and\\_charset/multilanguage\\_site](http://ez.no/ez_publish/documentation/configuration/configuration/language_and_charset/multilanguage_site)

**[eZDoc Override]** : Override templates, eZ publish, 28.05.2004/ 10.03.2004,

URL:[http://www.ez.no/ez\\_publish/documentation/customization/custom\\_design/override\\_templates](http://www.ez.no/ez_publish/documentation/customization/custom_design/override_templates)

**[eZDoc Permissions]** : Module Tutorial Part 1, eZ Crew, 14.09.2004/ ,

URL:[http://www.ez.no/ez\\_publish/documentation/development/extensions/module/module\\_tutorial\\_part\\_1](http://www.ez.no/ez_publish/documentation/development/extensions/module/module_tutorial_part_1)

**[eZDoc Site access]** : Site access, eZ publish, 08.06.2004/ 20.08.2003,

URL:[http://ez.no/ez\\_publish/documentation/configuration/configuration/site\\_access](http://ez.no/ez_publish/documentation/configuration/configuration/site_access)

**[eZDoc Site access Config]** : Site access, eZ publish, 28.05.2004/ 20.08.2003,

URL:[http://ez.no/ez\\_publish/documentation/configuration/configuration/site\\_access](http://ez.no/ez_publish/documentation/configuration/configuration/site_access)

**[NetLexikon 04]** : Net-Lexikon, akademie.de asp GmbH, 26.05.2004/ ,

URL:<http://www.net-lexikon.de>