

Carsten Möhrke

# **Zend Framework**

Das Entwickler-Handbuch

## Auf einen Blick

<b>1</b>	<b>Der Model View Controller .....</b>	<b>25</b>
<b>2</b>	<b>Datenbankzugriff mit Zend_Db .....</b>	<b>59</b>
<b>3</b>	<b>Benutzer- und Rechteverwaltung .....</b>	<b>115</b>
<b>4</b>	<b>Infrastruktur-Klassen .....</b>	<b>137</b>
<b>5</b>	<b>Webservices .....</b>	<b>209</b>
<b>6</b>	<b>Arbeit mit E-Mails und Dateiformaten .....</b>	<b>279</b>
<b>7</b>	<b>Protokolle und Co. ....</b>	<b>335</b>
<b>8</b>	<b>Lokalisierung und Internationalisierung .....</b>	<b>369</b>

# Inhalt

Geleitwort des Fachgutachters .....	11
Einleitung .....	13

## 1 Der Model View Controller ..... 25

1.1 Die Theorie des MVC .....	25
1.2 Die Praxis des MVC .....	26
1.2.1 Der Front Controller .....	29
1.2.2 Der Action Controller .....	30
1.2.3 Einbinden eines Views .....	34
1.2.4 Die Verarbeitungsschritte .....	37
1.2.5 Übergabe von Werten .....	38
1.2.6 Das Model .....	41
1.2.7 Error Handling .....	42
1.2.8 Fortgeschrittene Techniken .....	47
1.2.9 Ein Beispiel .....	50

## 2 Datenbankzugriff mit Zend\_Db ..... 59

2.1 Datenbankunabhängigkeit .....	59
2.2 Nutzung von Zend_Db .....	60
2.2.1 Transaktionen .....	73
2.2.2 Sequenzen und automatisch generierte IDs .....	75
2.2.3 Spezielle Datenbankzugriffsmethoden .....	76
2.3 Datenbankzugriff mit Zend_Db_Table .....	88
2.3.1 Einfügen von Daten .....	93
2.3.2 Aktualisieren von Daten .....	94
2.3.3 Löschen von Daten .....	95
2.3.4 Auslesen von Daten .....	95
2.3.5 Kaskadierende Lösch- und Update-Vorgänge .....	109
2.4 Performanceanalyse mit Zend_Db_Profiler .....	110

## 3 Benutzer- und Rechtemanagement ..... 115

3.1 Rechteverwaltung mit Zend_Acl .....	115
3.1.1 Vererbung von Rechten .....	118
3.1.2 Verfeinern des Rechtesystems .....	120
3.1.3 Manipulieren von Rechten .....	123

## Inhalt

3.2	Benutzerauthentifikation mit Zend_Auth .....	124
3.2.1	Datenbankbasierte Authentifikation .....	124
3.2.2	Dateibasierte HTTP-Authentifikation .....	128
3.3	Session-Verwaltung mithilfe von Zend_Session .....	129
3.3.1	Eine Session starten .....	130
3.3.2	Gültigkeit und Schutz von Session-Daten .....	132
3.3.3	Nutzung eigener Session-Save-Handler .....	133
<b>4</b>	<b>Infrastruktur-Klassen .....</b>	<b>137</b>
4.1	Performance-Optimierung mit Zend_Cache .....	137
4.1.1	Frontends .....	139
4.1.2	Nutzung von Backends .....	149
4.1.3	Manuelle Verwaltung von Cache-Einträgen .....	153
4.2	Prüfen von Werten mit Zend_Validate .....	154
4.2.1	Prüfen auf alphanumerische Daten .....	156
4.2.2	Prüfen von Texten .....	156
4.2.3	Prüfen, ob eine Zahl in einem bestimmten Bereich liegt .....	156
4.2.4	Prüfen von Kreditkartennummern .....	157
4.2.5	Prüfen eines Datums .....	157
4.2.6	Testen von Ziffernfolgen .....	157
4.2.7	Validieren von E-Mail-Adressen .....	157
4.2.8	Testen eines Strings auf Fließkomma-Eigenschaften .....	160
4.2.9	Prüfen, ob eine Zahl über einer Grenze liegt .....	160
4.2.10	Testen von hexadezimalen Zahlen .....	160
4.2.11	Validieren von Hostnames .....	161
4.2.12	Testen von Array-Inhalten .....	163
4.2.13	Validieren von Integer-Werten .....	163
4.2.14	Prüfen von IP-Adressen .....	163
4.2.15	Prüfen, ob eine Zahl unter einer Grenze liegt .....	164
4.2.16	Testen, ob eine Variable leer ist .....	164
4.2.17	Validierung auf Basis eines regulären Ausdrucks .....	164
4.2.18	Testen eines Strings auf seine Länge hin .....	165
4.3	Filtern von Daten mit Zend_Filter .....	165
4.3.1	Alphanumerische Zeichen mit Zend_Filter_Alnum filtern .....	167
4.3.2	Buchstaben filtern .....	168
4.3.3	Extrahieren eines Basenames .....	168
4.3.4	Ziffern mit Zend_Filter_Digits ausfiltern .....	169

4.3.5	Extrahieren von Verzeichnisnamen .....	169
4.3.6	Konvertieren von Sonderzeichen in Entitäten .....	169
4.3.7	Filtern von Integer-Werten .....	170
4.3.8	Absolute Pfade mit Zend_Filter_RealPath extrahieren ....	170
4.3.9	Konvertieren in Kleinbuchstaben .....	170
4.3.10	Konvertieren in Großbuchstaben .....	171
4.3.11	Entfernen von Whitespaces .....	171
4.3.12	Entfernen von HTML-Tags .....	172
4.3.13	Nutzung eigener Filter .....	174
4.4	Formularverarbeitung mit Zend_Filter_Input .....	175
4.5	Schreiben von Logs mit Zend_Log .....	184
4.5.1	Log-Einträge filtern .....	190
4.5.2	Logfile-Einträge formatieren .....	191
4.5.3	Eigene Einträge definieren .....	194
4.6	Konfigurationsverwaltung Zend_Config .....	194
4.6.1	Nutzung von Konfigurations-Arrays .....	195
4.6.2	INI-Dateien .....	197
4.6.3	XML-Dateien .....	200
4.7	Shell-Programmierung mit Zend_Console_Getopt .....	203
4.7.1	Optionen, Flags und Parameter .....	204
4.7.2	Nutzung von Argumenten .....	207

## 5 Webservices ..... 209

5.1	Feeds mit Zend_Feed verarbeiten .....	209
5.1.1	Feeds finden .....	209
5.1.2	Allgemeines zur Verarbeitung von Feeds .....	210
5.1.3	Verarbeiten von RSS-Feeds .....	211
5.1.4	Verarbeiten von Atom-Feeds .....	214
5.1.5	Generieren von Feeds .....	218
5.2	Zugriff auf Amazon mit Zend_Service_Amazon .....	220
5.3	Zugriff auf Flickr mit Zend_Service_Flickr .....	232
5.4	Yahoo!-Suche mit Zend_Service_Yahoo .....	236
5.4.1	Websuche .....	236
5.4.2	News-Suche mit Zend_Service_Yahoo .....	240
5.4.3	Bildersuche mit Zend_Service_Yahoo .....	242
5.5	Zugriff auf Google-Dienste mit Zend_Gdata .....	245
5.5.1	Allgemeines zu Zend_Gdata .....	246
5.5.2	Authentifikation .....	246
5.5.3	Nutzung von Google Calendar .....	255
5.5.4	Nutzung von Google Spreadsheets .....	271

**6 Arbeit mit E-Mails und Dateiformaten ..... 279**

6.1	E-Mails mit Zend_Mail verarbeiten .....	279
6.1.1	E-Mails versenden .....	280
6.1.2	Versand über SMTP-Server .....	286
6.1.3	E-Mails abholen .....	288
6.1.4	Löschen von E-Mails .....	298
6.1.5	Erweiterte Möglichkeiten von IMAP .....	300
6.2	JSON-Daten mit Zend_Json verarbeiten .....	306
6.3	Generieren von PDF-Dokumenten .....	308
6.3.1	Nutzung anderer Schriften .....	313
6.3.2	Mehrzeilige Fließtexte .....	315
6.3.3	Nutzung von Farben .....	320
6.3.4	Zeichnen in PDF-Dokumenten .....	322
6.3.5	Einbinden von Bildern .....	330
6.3.6	Meta-Informationen einfügen .....	331
6.3.7	Einlesen von PDF-Dokumenten .....	333

**7 Protokolle und Co. .... 335**

7.1	Zugriff auf andere Server mit Zend_Http .....	335
7.1.1	Das HTTP-Protokoll .....	335
7.1.2	Einen HTTP-Client erstellen .....	336
7.1.3	Übergabe von Werten .....	338
7.1.4	Uploads .....	340
7.1.5	HTTP-Authentifikation .....	341
7.1.6	Server-Antworten auswerten .....	342
7.1.7	Cookies .....	346
7.1.8	Nutzung von Adaptern .....	351
7.2	URIs mit Zend_Uri verarbeiten .....	352
7.2.1	URIs analysieren .....	353
7.3	Nutzung von XML-RPC mit Zend_XmlRpc .....	355
7.3.1	Allgemeines zu Zend_XmlRpc .....	355
7.3.2	Erstellen eines XML-RPC-Servers .....	356
7.3.3	Erstellen eines XML-RPC-Clients .....	359
7.4	Nutzung von REST mit Zend_Rest .....	361
7.4.1	Zugriff auf offene REST-Schnittstellen .....	362
7.4.2	Implementation eines REST-Servers .....	363

<b>8</b>	<b>Lokalisierung und Internationalisierung .....</b>	<b>369</b>
8.1	Lokalisierung mit Zend_Locale .....	369
8.1.1	Standardtexte und Standardformate lokalisieren .....	371
8.2	Mehrsprachige Oberflächen mit Zend_Translate .....	381
8.2.1	Nutzung von CSV-Dateien .....	384
8.3	Konvertieren von und Rechnen mit Maßeinheiten mittels Zend_Measure .....	385
8.4	Währungsdarstellung mit Zend_Currency .....	389
8.5	Datums- und Zeitangaben mit Zend_Date verarbeiten .....	395
8.5.1	Ableiten eines Zend_Date-Objekts .....	395
8.5.2	Rechnen mit Daten .....	402
8.5.3	Vergleich von Daten .....	404
8.5.4	Prüfen von Datumsinformationen .....	407
	Inhalt der CD-ROM .....	409
	Index .....	411

*You never win a game unless you beat the guy in front of you. The score on the board doesn't mean a thing. That's for the fans. You've got to win the war with the man in front of you. You've got to get your man.*  
– Vince Lombardi, Football Coach

## Einleitung

Zend Framework wurde lange und mit viel Spannung erwartet. Nach einigen Monaten Entwicklungszeit ist die erste Version am 30. Juni 2007 veröffentlicht worden.

Neben Zend Framework existieren auch viele andere etablierte Frameworks am Markt. Warum sollte man also Zend Framework nutzen? Meiner Ansicht nach gibt es eine ganze Reihe Gründe, dies zu tun.

Zuerst einmal ist da natürlich die hohe Qualität des Codes. Der Name Zend bürgt für Qualität. Neben einer robusten Implementation sind auch viele Ansätze des modernen Software Engineerings zu finden. Die Entwicklung ist komplett testgetrieben, sodass die Anzahl der Bugs eher gering ist. Natürlich gibt es hier und da mal einen Bug, wie in jeder anderen Software auch, aber in den meisten Fällen sind diese wenig dramatisch und werden schnell korrigiert. Selbstverständlich gibt es Coding Standards und Namenskonventionen an die sich alle Entwickler halten. Ich würde Ihnen empfehlen, dass Sie diese bei der Entwicklung Ihrer Anwendungen auch zu Grunde legen. Sie können diese hier nachlesen: <http://framework.zend.com/manual/coding-standard.html>.

In den Reihen des Entwicklerteams finden Sie viele Entwickler, die sich in der PHP-Szene seit vielen Jahren einen Namen gemacht haben, erfahren sind und hochwertigen Code erstellen.

Für die Nutzung spricht auch das interessante Lizenz-Modell (<http://framework.zend.com/license>). Es ist wesentlich freier und flexibler als viele andere Lizenzen. Es ist auch ohne Probleme möglich, Code der auf dem Zend Framework basiert kommerziell zu vertreiben.

Ein weiterer Punkt ist die wirklich gute Implementation des Model View Controllers (MVC), der Ihnen gerade bei umfangreichen Applikationen sehr viel

Arbeit abnehmen kann, zu einem deutlich strukturierten Code führt und sehr robust umgesetzt ist.

Aber keine Angst, Sie sind nicht gezwungen auf MVC aufzusetzen. Jede Komponente aus dem Zend Framework kann einzeln genutzt werden, sodass Sie vollkommen flexibel sind.

Eine besondere Stärke des Zend Frameworks sind sicherlich die Klassen zur Internationalisierung bzw. Lokalisierung von Code. Hier finden sich viele sehr leistungsfähige Klassen, die Sie gut unterstützen, wenn Sie mehrsprachige Anwendungen für eine internationale Zielgruppe erstellen. Auch die Klassen, die Ihre Applikation »im Hintergrund« unterstützen, den Zugriff auf eine Datenbank ermöglichen, Daten cachen, Logs schreiben oder E-Mails verschicken, sind wirklich sehr umfangreich, performant und stabil.

Aber genug des Lobes. Ich möchte nicht verschweigen, dass das Zend Framework auch noch ein paar deutliche Schwächen hat. So könnte beispielsweise die eine oder andere Webservice-Klasse mehr Funktionalitäten gebrauchen, und auch die PDF-Klasse ist noch ein wenig schwach auf der Brust. Dennoch sind auch diese Klassen auf einem guten Weg. Sollten Sie zum jetzigen Zeitpunkt allerdings eine dieser Klassen nutzen wollen, so ist es empfehlenswert, vorher zu prüfen, ob die benötigten Funktionalitäten enthalten sind.

Einen Punkt – und das ist meiner Ansicht nach ein absolut unschlagbarer Vorteil – gibt es aber noch. Und zwar finden Sie in der Zend-IDE *Zend Studio for Eclipse* eine komplette Syntaxunterstützung für das Zend Framework. Das heißt, die IDE kennt alle Klassen und Methoden und schlägt Ihnen diese bei der Code Completion vor. Damit aber nicht genug. Möchten Sie ein neues Projekt anlegen, so können Sie dem Tool auch von vornherein mitteilen, dass Sie ein neues MVC-Projekt auf Basis des Zend Frameworks anlegen wollen. Das Zend Studio generiert dann den kompletten »Standard-Code«, damit Sie ihn nicht tippen müssen. Sie können sich sicher vorstellen, dass Sie Ihren Code mit diesen Funktionalitäten deutlich schneller entwickeln können. Arbeiten Sie professionell mit PHP, so sollten Sie auf jeden Fall einen Blick auf *Zend Studio for Eclipse* werfen. Die Investition lohnt sich.

## Informationsquellen

Zwar hoffe ich als Autor natürlich, dass das Buch, das Sie gerade in Händen halten, die meisten Fragen zum Zend Framework beantworten wird. Aber ich weiß natürlich, dass noch Fragen offenbleiben. Bestimmt werden Sie noch eine Frage zu einem Paket haben, das hier nicht erläutert wird. Oder es gibt eine Änderung

im Framework, die dazu führt, dass das Buch nicht mehr ganz aktuell ist. Wo können Sie dann weitere Informationen erhalten?

Zum Ersten ist natürlich die Website des Zend Frameworks eine gute Anlaufstelle. Dort finden Sie eine sehr brauchbare Dokumentation vor. Als Programmierer werden Sie das Problem kennen, dass man nicht gerne dokumentiert. Aber man muss wirklich sagen, dass das Manual des Zend Frameworks eine rühmliche Ausnahme darstellt. Die Dokumentation finden Sie unter der Internetadresse <http://framework.zend.com/manual/>.

Sie sollten hierbei die englische Variante der Dokumentation bevorzugen, weil sie in Teilen einfach aktueller ist, und sich bei anderen Sprachen bei der Übersetzung schon mal ein Fehler einschleichen kann.

Da Programmierer lieber programmieren als schreiben, kann es auch sehr hilfreich sein, sich die Demos anzuschauen, die für einige Pakete vorhanden sind. Diese finden Sie im Download-Archiv des Zend Frameworks im Ordner *demos*.

Auf der Website des Zend Frameworks sehen Sie oben in der Navigation auch einen Link, der mit »Support« beschriftet ist. Hier finden Sie einige Möglichkeiten, wie Sie Unterstützung erhalten oder sich mit anderen Anwendern austauschen können. Sehr interessant ist sicherlich auch das Wiki zum Zend Framework, auf dem Sie eine ganze Menge an Hintergrundinformationen finden. Sie erreichen es unter <http://framework.zend.com/wiki/display/ZFDEV/Home>.

Im Support-Bereich finden Sie einen Link, unter dem Sie sich für diverse Mailing-Listen anmelden können. Diese Listen sind, auch wenn sie englischsprachig sind, sehr zu empfehlen. Zum einen sind diese Listen wirklich sehr aktiv und werden von vielen Benutzern abonniert. Zum anderen sind auch viele Entwickler der Pakete sehr aktiv. Das heißt, Sie bekommen oft sehr schnell eine qualifizierte Antwort. Es lohnt sich also, die Listen zu abonnieren. Sollten Sie eine Frage haben, sollten Sie aber zuvor vielleicht das Archiv der Mailing-Listen durchforschten, ob die Frage nicht schon einmal gestellt wurde.

Auf der Support-Seite finden Sie zudem Links zu mehreren Blogs. Auch wenn Sie nach dem Zend Framework googeln, finden Sie schnell einige Blogs sowie auch Beiträge in Foren. Bitte genießen Sie solche Informationen immer mit Vorsicht. Gerade in der Zeit vor der Version 1.0 des Zend Frameworks haben sich noch viele Änderungen ergeben, die dazu führen, dass die Informationen in den Blogs veraltet sind. Sofern Sie eine solche Informationsquelle nutzen, so prüfen Sie bitte zunächst, wie alt der Beitrag ist. Gleiches gilt auch für viele Tutorials, die Sie in verschiedenster Form im Internet finden.

Ein weiterer guter Anlaufpunkt ist die Website der Firma Zend selbst, die Sie unter [www.zend.de](http://www.zend.de) erreichen. Dort finden Sie neben einigen Artikeln und Tutorials vor allem einige Webinare. Bei Webinaren handelt es sich um Präsentationen bzw. Schulungen, die Sie im Browser betrachten können. Teilweise sind sie aufgezeichnet, sodass Sie sie jederzeit betrachten können, teilweise sind sie aber auch live. Ich denke, dass Webinare einen guten und vor allem entspannten Einstieg in ein Thema liefern können. Daher möchte ich sie Ihnen empfehlen.

Neben den Angeboten, die in engem Zusammenhang mit Zend stehen oder von Zend stammen, gibt es natürlich noch andere Informationsquellen im Web. Interessant ist sicher die Website ZFTutorials, die Sie unter der Adresse <http://www.zftutorials.com> erreichen. Hier werden Tutorials rund um das Zend Framework gesammelt. Auch hier gilt, dass die Daten veraltet ein können.

Eine zweite Website, die ich Ihnen empfehlen möchte, ist das deutschsprachige Zend Framework-Forum, das Sie unter <http://www.zfforum.de> finden. Hier sind sehr engagierte Mitglieder anzutreffen, die teilweise auch zu den Entwicklern des Zend Frameworks gehören, wodurch eine qualitativ hochwertige Hilfe sichergestellt ist.

Neben den bisher vorgestellten Informationsquellen gibt es noch zwei Anlaufstellen, die ich ebenfalls für sehr wichtig halte. Die erste ist der *Zend Framework Issue Tracker*. Dabei handelt es sich um das System, das für das Bug-Reporting genutzt wird. Wenn Sie also einmal ein Problem haben, dann kann es sich durchaus lohnen, hier nachzuschauen, ob es sich eventuell um einen bekannten Bug handelt. Sollten Sie selbst einmal einen Bug finden, wäre es sehr nett, wenn Sie ihn hier eintragen und somit helfen, die Qualität des Zend Frameworks weiter zu verbessern.

Eine weitere interessante Webadresse ist die folgende: <http://framework.zend.com/changelog>. Dabei handelt es sich um das Changelog zum Zend Framework. Hier finden Sie Informationen zu den Veränderungen innerhalb der einzelnen Versionen. Wenn Sie eine neue Version von Zend Framework installieren, so schauen Sie bitte dort nach, was sich geändert hat. Das Zend Framework ist noch jung und ein sehr »lebendiges« System. Es kann also durchaus einmal passieren, dass sich das Verhalten einer Methode oder eine API ändert.

Als letzten Punkt möchte ich auf die Website <http://www.zf-buch.de> verweisen. Mit dieser Website möchte ich Sie als Leser ein wenig unterstützen. Sie finden dort Informationen zum Zend Framework und Ergänzungen sowie Korrekturen zu diesem Buch.

## Installation

Das Zend Framework zu installieren, ist erfreulich einfach. Systemseitig gibt es eigentlich nur die Voraussetzung, dass mindestens PHP 5.1.4 vorhanden sein muss, wobei allerdings die Version 5.2.3 empfohlen wird. Ist diese Voraussetzung erfüllt, müssen Sie einfach nur das Archiv, das Sie unter <http://framework.zend.com> finden, herunterladen und entpacken. In dem dabei entstehenden Ordner finden Sie ein Verzeichnis namens *library* und darin ein Verzeichnis namens *Zend*. In diesem Ordner liegt alles, was Sie benötigen. Sie können den Ordner *library* oder auch nur den Ordner *Zend* an eine beliebige Stelle auf Ihrem Server kopieren, falls Sie einen eigenen Server nutzen. Wenn möglich sollten Sie den Ordner nicht unterhalb des DocumentRoot-Verzeichnisses des Webservers ablegen. Der Webserver benötigt nur Leserechte auf das Verzeichnis; aber es gibt keinen Grund, warum das Verzeichnis von außen ansprechbar sein sollte.

Sollte Ihnen nur gemieteter Webspaces zur Verfügung stehen, ist das auch kein Problem. In dem Fall können Sie den Ordner auch mit im Webspaces ablegen. Sie sollten den Ordner allerdings so sichern, dass er nicht von außen aufgerufen werden kann.

Nachdem Sie den Ordner bereitgestellt haben, müssen Sie nur noch dafür sorgen, dass der Ordner oberhalb des Ordners *Zend* mit im *Include-Path* liegt. Haben Sie das Verzeichnis *library* kopiert müsste dieser Ordner also in den *Include-Path*. Bei dem *Include-Path* handelt es sich um ein oder mehrere Verzeichnisse, die PHP automatisch durchsucht, wenn Sie eine Datei mit `require` o. Ä. einbinden. Um den Ordner mit im *Include-Path* unterzubringen, gibt es verschiedene Möglichkeiten. Die beste Variante ist sicherlich, die Konfigurationsdatei *php.ini* entsprechend zu editieren. Nutzen Sie einen eigenen Server, finden Sie diese Datei meist im Ordner */etc*. Sollten Sie für die Entwicklung auf Xampp, MAMP o. Ä. setzen, so müssen Sie in den entsprechenden Unterverzeichnissen nach der Datei suchen.

In der Datei *php.ini* finden Sie eine Direktive namens *include\_path*. Hier könnte beispielsweise

```
include_path = "usr/share/php"
```

stehen oder Ähnliches. Wenn Sie den Ordner *Zend* beispielsweise nach */usr/share* kopiert haben, müssten Sie diesen Pfad hier ergänzen. Das heißt, die Direktive müsste danach (zumindest unter einem UNIX-Betriebssystem) so aussehen:

```
include_path = ".:usr/share/php:usr/share"
```

## Einleitung

Nutzen Sie Windows und haben Sie den Ordner unterhalb von *d:\Webentwicklung* abgelegt, müssten Sie die ursprünglich vorhandene Angabe

```
include_path = ".;c:\php\includes"
```

so ergänzen:

```
include_path = ".;c:\php\includes;d:\Webentwicklung"
```

Was aber, wenn Sie keinen Zugriff auf die Datei *php.ini* haben? Die einfachste Variante ist dann sicher, dass Sie den Pfad direkt in der PHP-Datei festlegen, die ausgeführt wird. Würde sich der Ordner *Zend* unterhalb von */usr/share* befinden, könnten Sie die folgenden Zeilen am Anfang eines jeden PHP-Scripts ergänzen:

```
$path = '/usr/share';  
set_include_path(get_include_path() . PATH_SEPARATOR . $path);
```

Dabei würde es sich natürlich anbieten, die Zeilen in eine Datei auszulagern, die dann immer mit `require_once` eingebunden wird. Mit diesen Zeilen wird der aktuell gesetzte Pfad ausgelesen, dann um den zusätzlichen Pfad ergänzt und das Ganze wieder gespeichert.

Eine weitere Möglichkeit ist, dass Sie den Pfad mit einer *.htaccess*-Datei setzen, die Sie in dem Verzeichnis ablegen, in dem auch die Applikation liegt. In dieser können Sie dann mit

```
php_value include_path ".:usr/local/lib/php:usr/share"
```

den Pfad setzen. Wichtig ist, dass Sie vorher mithilfe von `get_include_path()` oder `phpinfo()` die aktuellen Pfadeinstellungen auslesen und mit in die Zeile einfügen. Andernfalls würden die anderen Pfade verloren gehen.

Danach können Sie sofort loslegen und das Zend Framework nutzen. Ist Ihnen das alles zu aufwändig, dann können Sie auch einfach zu Zend Core greifen. Dabei handelt es sich um einen Apache Webserver der ein fertig konfiguriertes PHP, inklusive Zend Framework, mitbringt.

## Allgemeines zu diesem Buch

Es gibt ein paar Punkte, die Sie zu diesem Buch wissen sollten, um Dinge besser verstehen zu können. Der erste wichtige Punkt ist sicher, dass die meisten Beispiele nicht auf dem Model View Controller-Pattern basieren. Warum das so ist, fragen Sie? Nun, zum Ersten wäre es einfach zu komplex gewesen, die Pakete zu erläutern und dabei auch ständig die Funktionalitäten mit zu implementieren,

die für ein MVC genutzt werden. Der zweite Punkt ist, dass man vielleicht nicht jede Anwendung mit einem MVC erstellen möchte. Wollen Sie beispielsweise nur Teile des Zend Frameworks in eine Applikation integrieren, ist es sicher sehr hilfreich, dass die Pakete standalone beschrieben sind. Das ist auch der Grund, warum Sie nur sehr selten Beispiele finden, in denen zwei Klassen gemischt werden. Das ist nur in wenigen Fällen gegeben, beispielsweise wenn ein Datenbankzugriff mit `Zend_Db` erfolgt oder ein Datum mit `Zend_Date` konvertiert wird. Trotzdem finden Sie natürlich ein ausführliches Kapitel zum Aufbau und zur Nutzung des MVC-Patterns.

Ein weiterer Punkt, auf den ich hinweisen möchte, ist das Exception Handling. Hier mag man zuerst den Eindruck haben, dass ich geschlampt habe, aber dem ist nicht so. Ich habe bei den meisten Listings sehr bewusst darauf verzichtet, mit `try-catch`-Blöcken zu arbeiten. Der Grund dafür ist, dass dies bei der Vielzahl der Listings sehr viel Platz in Anspruch genommen hätte. Dennoch finden Sie an Stellen, an denen ich es für wichtig gehalten habe, auch entsprechende Blöcke. Wichtig ist in diesem Zusammenhang, dass Sie auf jeden Fall `try-catch`-Blöcke ergänzen sollten, sofern Sie Beispiele aus diesem Buch in Ihren Anwendungen nutzen möchten.

In diesem Zusammenhang ist auch zu beachten, dass die meisten Beispiele darauf ausgelegt sind, Dinge zu erklären. Sie sind daher nicht immer elegant oder gar performant.

Wahrscheinlich arbeiten die meisten von Ihnen nach wie vor im ISO-8859-1/-15-Zeichensatz. Daher sind die Listings alle auf diesen ISO-Zeichensatz ausgelegt. Trotzdem wird in vielen Zusammenhängen der UTF-8-Zeichensatz benötigt. Um das zu verdeutlichen, wird in diesen Listings immer ein UTF-8-Header mitgeschickt, und die Textdaten werden meist manuell mithilfe von `utf8_encode()` und `utf8_decode()` codiert bzw. decodiert.

Falls Sie einen Fehler im Buch finden sollten, so würde ich mich freuen, wenn Sie mir eine kurze E-Mail an die E-Mail-Adresse *zf-buch@netviser.de* schicken würden.

## Allgemeines zur Nutzung

Das Zend Framework ist nicht nur einfach zu installieren, sondern auch einfach zu nutzen. Außerdem ist es sehr klar strukturiert.

## Struktur des Zend Frameworks

Das Zend Framework ist klar strukturiert. Am Namen der Klasse können Sie stets sofort erkennen, wo sich eine Datei befindet. Wollen Sie beispielsweise die Klasse `Zend_Currency` nutzen, so lautet die dazugehörige Klassen-Datei *Zend/Currency.php*. Jede Klasse besitzt eine eigene Datei, die Sie inkludieren bzw. laden können. Benötigt eine Klasse weitere Dateien, was fast immer der Fall ist, finden Sie diese in einem Unterverzeichnis, das dem Namen der Klasse entspricht. So benötigt beispielsweise `Zend_Currency` noch die Klasse `Zend_Currency_Exception`. Diese ist somit in der Datei *Zend/Currency/Exception.php* deklariert. Grundsätzlich gilt also, dass Sie den Namen der Klasse nehmen, die Unterstriche durch Slashes ersetzen und die Endung *.php* anhängen, um den Namen der Datei zu erhalten.

Ein wenig schade ist, dass die meisten Klassen nicht nach Themenbereichen gruppiert sind. Nur wenige Klassen, wie beispielsweise die Webservice-Klassen, sind nach Themenbereichen zusammengefasst. Diese Klassen, wie `Zend_Service_Amazon` oder `Zend_Service_Yahoo`, finden Sie im Unterverzeichnis *Zend/Service*, wie Sie sicher schon vermutet haben.

Erwähnt werden sollte noch, dass es einige Klassen gibt, bei denen in einem der Namensbestandteile ein Großbuchstabe vorkommt, wie beispielsweise bei `Zend_XmlRpc`.

## Laden der Klassen-Dateien

Sie können die Klassen-Dateien, die Sie benötigen, jederzeit mithilfe des Sprachkonstrukts `require_once` einbinden. Dies ist gleichzeitig das Vorgehen, das in diesem Buch an den meisten Stellen verwendet wird.

Allerdings können Sie auch die Klasse `Zend_Loader` nutzen. `Zend_Loader` bietet eine Reihe von Möglichkeiten, um Dateien einzubinden. Um eine Klasse aus dem Zend Framework zu laden, müssen Sie lediglich den Namen der Klasse an die statische Methode `loadClass()` übergeben. Diese lädt die Klassen-Datei und prüft dabei automatisch, ob die Klasse in der Datei enthalten ist. Sollte die Klasse schon vorhanden sein, so wird sie nicht doppelt eingebunden. Kommt es beim Laden zu einem Problem, so wirft die Methode eine `Exception`.

Der ideale Weg, um eine Klasse aus dem Zend Framework zu laden, ist daher:

```
require_once('Zend/Loader.php');
Zend_Loader::loadClass('Zend_Gdata_Calendar');
```

Hiermit würde also automatisch die Datei *Zend/Gdata/Calendar.php* geladen. Wie bereits erwähnt, habe ich in dem Buch üblicherweise mit `require_once()` gearbeitet. Die Entscheidung für `require_once()` resultierte daraus, dass ich hoffe, auf diesem Weg die Transparenz der Beispiele zu erhöhen. Dennoch möchte ich Ihnen empfehlen, `Zend_Loader::loadClass()` zu nutzen, da Sie dann auch in der Lage sind, ein sauberes Exception Handling zu nutzen.

### Exception Handling

Wie schon erwähnt, sind die Beispiele in diesem Buch nicht immer mit einem korrekten Exception Handling versehen. Genau genommen ist das sogar eher selten der Fall. Das sollte Sie aber keinesfalls davon abhalten, dies besser zu machen.

Grundsätzlich gilt, dass alle Fehler immer in einer Exception resultieren. An keiner Stelle geben die Methoden eine Fehlermeldung als Rückgabewert zurück. Hier und da lässt sich sicher darüber streiten, was ein Fehler ist und was nicht. Das heißt, einige Entwickler neigen auch dazu, bei kleineren Problemchen eine Exception auszulösen, wohingegen andere Entwickler das nur bei echten Fehlern tun. Daher sollten Sie immer damit rechnen, dass eine Exception auftreten kann.

Das Zend Framework ist auch bei den Exceptions sehr strukturiert: Jede Klasse bringt ihre eigenen Exceptions mit. Somit können Sie sehr genau erkennen, an welcher Stelle ein Fehler entstanden ist. Der Name der Exception-Klasse ergibt sich (fast) immer aus dem Namen der Hauptklasse, die verwendet wird. Deren Namen wird einfach nur ein `_Exception` angehängt. Einige wenige Klassen verfolgen hier allerdings einen anderen Ansatz, wie beispielsweise die `Gdata`-Klassen.

Zuweilen wirft eine Klasse eine Exception, mit der man nicht gerechnet hat. Wenn Sie also beispielsweise einen RSS-Feed mit `Zend_Feed` auslesen, wird im Hintergrund die Klasse `Zend_Http` für den Verbindungsaufbau genutzt. Somit kann es passieren, dass Sie eine Exception dieser Klasse erhalten, obwohl Sie damit nicht gerechnet haben.

Vor diesem Hintergrund empfiehlt es sich immer, dass Sie zusätzlich zu einem `catch`-Block für die Exceptions einer Klasse auf jeden Fall noch ein `catch` für die Klasse `Exception` nutzen. Da alle Exceptions Kind-Klassen der Klasse `Exception` sind, können Sie dann sicher sein, dass eine Exception, die nicht von einem vorhergehenden `catch` »gefangen« wurde, spätestens dann beachtet wird. Bezogen auf die Verarbeitung eines RSS-Feeds könnte dies so aussehen:

## Einleitung

```
try
{
    // Einlesen des RSS-Feeds
}
catch (Zend_Http_Exception $http_exception)
{
    // Fehler beim Verbindungsaufbau verarbeiten
}
catch (Zend_Feed_Exception $feed_exception)
{
    // Fehler bei der Verarbeitung der Daten
}
catch (Exception $allgemeine_exception)
{
    // Hier landen alle Exceptions mit dem man
    // nicht gerechnet hatte
}
```

Man kann an dieser Stelle sicherlich darüber diskutieren ob es Sinn macht Ausnahmen zu fangen, die man nicht behandeln kann. Wollen Sie beispielsweise auf einen anderen Server zugreifen und dieser ist nicht verfügbar, dann können Sie daran meist auch nichts ändern. Meine persönliche Meinung ist aber, dass man eine Ausnahme immer behandeln sollte. Eine nett formulierte Fehlermeldung macht die meisten Benutzer glücklicher als eine »Unhandled Exception« Meldung von PHP.

## Performance

Es liegt leider in der Natur der Sache, dass die meisten Frameworks eine Anwendung langsamer machen. Jede Software-Schicht, jede Abstraktion führt dazu, dass mehr Klassen und Objekte Verwendung finden. Soll Code robust sein, so müssen viele Abfragen für Eventualitäten enthalten sein. All das kostet Performance. Auch das Zend Framework stellt da keine Ausnahme dar. Um solche Einflüsse zu kompensieren empfiehlt es sich einen Opcode-Cache auf dem Produktiv-Server zu installieren. Dabei handelt es sich um einen Cache, der fertig übersetzten PHP-Code zwischenspeichern kann. Somit können Sie den größten Teil der Geschwindigkeitsverluste die durch die Analyse und Interpretation des Codes entstehen kompensieren. In diesem Bereich gibt es eine ganze Menge empfehlenswerter Lösungen wie eAccelerator, APC oder Zend Platform.

## Sonstige Besonderheiten

Sollten Sie noch nicht sicher im Umgang mit Objekten und Klassen sein, wäre jetzt ein guter Zeitpunkt, sich in die Thematik einzuarbeiten. Das Zend Framework nutzt durchweg modernste Entwicklungsansätze, sodass Interfaces, abstrakte Klassen und Ähnliches keine Fremdworte für Sie sein sollten. Zwar haben Sie mit diesen Dingen nicht unbedingt immer direkt zu tun, aber es kann nicht schaden, die Ideen dahinter verstanden zu haben.

Im Zusammenhang mit der Objektorientierung soll auch erwähnt werden, dass das Zend Framework an den meisten Stellen auf ein »Fluent Interface« setzt. Bei diesem »fließenden« Interface liefern Methoden, die keinen anderen Wert zurückgeben müssen, immer dasjenige Objekt zurück, aus dem heraus sie aufgerufen wurden. Dies hat zur Folge, dass Sie mehrere Methoden direkt hintereinander aufrufen können. War Ihnen das zu abstrakt? Betrachten Sie hierzu die folgende Klasse:

```
class Rechner
{
    private $_divident;
    private $_divisor;

    public function setzeWerte ($l, $r)
    {
        if ($r == 0)
        {
            throw new Exception("Division durch Null");
        }
        $this->_divident = $l;
        $this->_divisor = $r;
        return $this;
    }

    public function dividiere()
    {
        return ($this->_divident / $this->_divisor);
    }
}
```

Die Methode `setzeWerte()` muss keinen Wert zurückgeben, da hier nur Parameter für die Berechnung gesetzt werden. In einem konventionellen, prozeduralen Ansatz würde die Methode sicher `true` zurückgeben, falls die Zuweisung erfolgreich war, und `false`, wenn es zu einem Problem kam. Da das Zend Framework aber mit Exceptions arbeitet, ist das nicht nötig. Bei einem Problem wird eine

## Einleitung

Exception geworfen und andernfalls `$this` zurückgegeben. Das hat zur Folge, dass Sie die Klasse wie folgt nutzen können:

```
$rechner = new Rechner();  
echo $rechner->setzeWerte(1,3)->dividiere();
```

In vielen Fällen können Sie so eine recht stattliche Anzahl von Methoden miteinander verknüpfen. Das kann die Lesbarkeit verbessern, kann aber auch schnell verwirren. Überlegen Sie also gut, wie viele Methodenaufrufe Sie miteinander verknüpfen wollen.

Ein weiterer Punkt, auf den ich hinweisen möchte, ist die *Standard PHP Library (SPL)*. Sie ist ein PHP-Feature, das mit PHP 5 eingeführt wurde, und meiner Ansicht nach leider viel zu wenig Beachtung findet. Die SPL definiert eine ganze Reihe von Interfaces, welche für eigene Klassen genutzt werden können. In vielen Klassen des Zend Frameworks finden Sie eine Implementation des Interfaces `Iterator`. Eine Klasse, die dieses Interface implementiert, kann beispielsweise direkt in einer `foreach`-Schleife genutzt werden. Enthält ein Objekt zum Beispiel verschiedene andere Objekte, könnte die Schleife bei jeder Iteration eines dieser Objekte auslesen.

Sollten Sie nicht mit der SPL vertraut sein, so würde ich Ihnen empfehlen, dass Sie sich ein wenig mit ihr befassen. Eine Einleitung finden Sie in der PHP-Dokumentation unter <http://www.php.net/spl> bzw. unter <http://www.php.net/~helly/php/ext/spl>. Insbesondere das Interface `Iterator` sollten Sie sich dort anschauen. Hilfreich ist auch, wenn Sie verstehen, wozu die Interfaces `ArrayAccess` und `RecursiveIterator` sowie die Klasse `RecursiveIteratorIterator` dienen.

*No one has ever drowned in sweat.*  
– Lou Holtz, Football Coach

## 1 Der Model View Controller

Das Kernstück des Zend Frameworks sind die Klassen zur Umsetzung des MVC-Entwurfsmusters. Die Abkürzung MVC steht für Model View Controller und beschreibt, wie eine Anwendung strukturiert werden kann. Das Zend Framework bringt hierbei eine große Menge Funktionalitäten mit, die Ihnen die Erstellung einer Applikation auf Basis eines MVC ermöglichen.

Zur Umsetzung eines MVC werden mehrere Klassen miteinander kombiniert, sodass in diesem Kapitel – anders als in anderen Kapiteln – mehrere Klassen gleichzeitig genutzt und erläutert werden.

Sie werden feststellen, dass das Thema recht komplex ist. Daher kann dieses Kapitel nicht alles erläutern. Dennoch denke ich, dass die wichtigen Inhalte hier abgebildet sind. Es kann aber auch nicht schaden, wenn Sie noch ein wenig in der Dokumentation stöbern. Sie finden dort noch viele hilfreiche Dinge wie beispielsweise die Helper.

Bevor ich auf den praktischen Teil eingehe, möchte ich zunächst die Theorie erläutern.

### 1.1 Die Theorie des MVC

Wie schon erwähnt, ist die Idee des Model View Controllers, eine Anwendung in verschiedene Teile, nämlich das Model, den View und den Controller, aufzugliedern.

Lassen Sie mich mit dem letzten Teil, nämlich dem Controller beginnen. Der Controller ist derjenige Teil der Applikation, der auf Benutzereingaben reagiert, die Eingaben verarbeitet und dafür sorgt, dass die korrekten Seiten dargestellt werden.

Die eigentliche Darstellung der Daten wird vom View übernommen. Der View ist hierbei als eine abstrakte Komponente zu verstehen, da er sich aus mehreren

Teilen zusammensetzt. Neben der intern genutzten View-Klasse wird auch noch ein Template, also eine Design-Vorlage, genutzt, die für die Darstellung der Daten verwendet wird.

Der Dritte im Bunde, das Model, ist für die Datenhaltung und -verwaltung verantwortlich. Hierbei handelt es sich um eine Komponente, die dafür zuständig ist, die genutzten Daten unter anderem zu lesen und zu speichern. Gerne wird hierbei auch von einem Datenmodell gesprochen, wobei das vielleicht ein wenig verwirrend ist, da es sich nicht um ein Modell im Sinne eines Entity-Relationship-Modells, sondern um eine konkrete Klasse handelt.

In den meisten Fällen finden Sie hier auch die Geschäftslogik. Die Geschäftslogik ist dafür zuständig Objekte und Vorgänge aus der realen Welt in die Programmierung umzusetzen. Das heißt, im Fall eines Shops könnte sich hier eine Klasse finden die dafür zuständig ist Bestellungen anzulegen und zu verwalten. Die Geschäftslogik im Model anzusiedeln macht Sinn, da sie sehr eng mit den Daten verknüpft ist. Beim Anlegen einer Bestellung müssen die Informationen beispielsweise in eine Datenbank geschrieben werden. Die Business-Logik stellt also einen weiteren Abstraktionslevel dar. Durch dieses Konzept haben Sie den Vorteil, dass der komplette Zugriff auf die Daten gekapselt ist. Im Controller müssen Sie also nur noch die Eingaben des Benutzers entgegen nehmen und sie an die Methoden der Model-Klassen übergeben. Durch diese klare Struktur wird der Code deutlich wartbarer und übersichtlicher.

Sollten Sie sich nun fragen, wie man Model und Controller trennscharf definiert, so ist das berechtigt. Die Idee hinter der Aufteilung ist, die Daten und Funktionalitäten zu kapseln. Das heißt, das Model muss so strukturiert sein, dass alles, was die Speicherung und das Auslesen betrifft, hierin enthalten ist. Möchten Sie die Daten in einer Datenbank abspeichern, so muss das Escapen von Sonderzeichen ein Teil des Models sein. Auch Methoden zum Berechnen von Preisen, der Verwaltung von Lagerbeständen und Ähnliches sind hier anzusiedeln, da diese Funktionalitäten zur Geschäftslogik gehören.

Die Aufbereitung der Daten für die Darstellung hingegen ist Teil des Controllers.

## 1.2 Die Praxis des MVC

Eine Applikation auf Basis eines MVC umzusetzen, erscheint auf den ersten Blick ein wenig komplex, aber keine Angst, so schlimm ist's nicht.

Eine MVC-Applikation besitzt immer einen zentralen Einstiegspunkt. Es gibt also exakt eine Datei, die bei jedem Aufruf angesprochen wird. Hierbei handelt es sich

um den Front-Controller, der auch Bootstrap-File genannt wird. Bei genauer Betrachtung ist die Bezeichnung Front Controller vielleicht ein wenig verwirrend. Die Datei bietet die Funktionalität eines Front-Controllers, nutzt aber gleichzeitig auch ein Objekt der Klasse `Zend_Controller_Front`. Da Datei und Objekt sehr eng miteinander verknüpft sind, kann man mit dieser kleinen Ungenauigkeit leben. Die Datei ist allerdings nicht so komplex, wie Sie vielleicht gerade befürchten. Der Front Controller ist im Endeffekt nur dafür zuständig, die Anfrage entgegenzunehmen und sie an einen »Action-Controller« weiterzugeben, in dem die eigentliche Logik enthalten ist. Sie können sich den Front Controller also wie einen Manager vorstellen, der dafür zuständig ist, die Applikation zu verwalten. Rein technisch ist das zwar nicht ganz korrekt, aber an dieser Stelle soll das so erst einmal reichen.

Wie Sie vielleicht schon ahnen, wird im Hintergrund eine Menge »Magie« genutzt, welche die einzelnen Komponenten verknüpft. Das heißt, das Framework ist in der Lage, automatisch die notwendigen und korrekten Komponenten einzubinden. Dazu muss Ihre Applikation sich an eine vordefinierte Verzeichnisstruktur halten. Die Dokumentation des Zend Frameworks schlägt dafür diese Verzeichnisstruktur vor:

```
application/  
  controllers/  
  models/  
  views/  
    scripts/  
    helpers/  
    filters/  
html/
```

Die Ordner *html* und *application* befinden sich in diesem Beispiel auf einer Ebene, wobei nicht erforderlich ist. Es könnte sich beispielsweise auch um die Ordner */var/application* und */var/www/html* handeln. Das Directory *html* ist das eigentliche Document-Root-Verzeichnis des Servers. Hier würden normalerweise die HTML- bzw. PHP-Dateien abgelegt. Der Ordner muss übrigens nicht unbedingt *html* heißen. Er könnte genauso gut *htdocs* oder *public\_html* heißen.

Der Ordner *application* sollte nicht direkt unterhalb des Document-Root-Verzeichnisses des Servers liegen. Somit ist sichergestellt, dass niemand unberechtigten Zugriff auf eine der Dateien hat. Der Name *application* ist übrigens frei gewählt. Sie könnten hier auch einen beliebigen anderen Namen nutzen. Die Namen der Verzeichnisse, die darunter genutzt werden, sollten Sie allerdings übernehmen. Im Unterverzeichnis *controllers* werden die Controller-Klassen abgelegt, im Ordner *views/scripts* die Templates für die Darstellung und in *models* die Dateien, die für den Datenzugriff erforderlich sind.

Im Ordner *html* befindet sich unter Umständen nur eine einzige Datei, der Front Controller. Abhängig von der Applikation wären hier natürlich noch Grafiken, CSS-, JavaScript-Dateien oder Ähnliches zu finden.

Bevor ich auf die Controller eingehe, stellt sich noch die Frage, wie Sie sicherstellen können, dass die Anfragen auch wirklich alle beim Front Controller landen. Um das zu gewährleisten, sollten Sie auf die Rewrite-Engine Ihres Webservers zurückgreifen. Alle aktuellen Webserver unterstützen die Möglichkeit des URL-Rewritings, wobei unter Umständen, wie beim IIS, zusätzliche Module notwendig sind. Beim URL-Rewriting wird die URL, die ein Browser aufgerufen hat, einfach so umgeschrieben, dass ein bestimmtes Ziel angesprochen wird. In diesem Fall würde also jeder eingehende Aufruf auf die Datei *index.php* »umgebogen«. Genau genommen handelt es sich nicht um jeden Aufruf. Grafiken, statische HTML-Seiten, CSS-Dateien und alles andere, was nicht unmittelbar mit der Applikation zu tun hat, kann direkt ausgeliefert werden.

Leider sind die Webserver alle unterschiedlich zu konfigurieren. Daher werde ich hier nur auf den Apache-Webserver eingehen. Für andere Webserver ziehen Sie bitte das Manual Ihres Webservers zu Rate.<sup>1</sup>

Im Fall von Apache können Sie die Rewrite-Rule entweder in der entsprechenden Konfigurationsdatei Ihres Servers angeben oder Sie nutzen eine *.htaccess*-Datei, was wahrscheinlich einfacher und flexibler ist. Der Inhalt der Datei könnte beispielsweise so aussehen:

```
RewriteEngine on
RewriteRule !\.(js|ico|gif|jpg|png|css)$ index.php
```

Diese beiden Zeilen müssten unter dem Namen *.htaccess* im Document-Root-Verzeichnis des Servers gespeichert werden. Hiermit werden zwei Dinge definiert. In der ersten Zeile wird die Rewrite-Engine des Servers eingeschaltet. Üblicherweise ist das kein Problem. Sollte das dafür notwendige Modul `mod_rewrite` nicht verfügbar sein, müssten Sie es nachinstallieren oder sich mit Ihrem Administrator bzw. Provider in Verbindung setzen.

In der zweiten Zeile finden Sie die eigentliche Regel, die für das Rewriting zuständig ist. Im Endeffekt handelt es sich bei dem ersten Teil (`!\.(js|ico|gif|jpg|png|css|css|htm)$`) um einen regulären Ausdruck, der auf alle URLs zutrifft, die nicht auf eine der enthaltenen Endungen enden. Sollten Sie noch weitere Dateiformate nutzen, die nicht in der Liste enthalten sind, beispielsweise

---

<sup>1</sup> Die Dokumentation zum Zend Framework beinhaltet auch einige Informationen zu anderen Webservern: <http://framework.zend.com/manual/en/zend.controller.router.html>

RSS oder WSDL, so müssten Sie diese noch ergänzen. Der zweite Teil der Regel (`index.php`) ist das Ziel, auf das umgeleitet wird, also die Datei `index.php`.

Sollten Sie das MVC-Projekt nicht im Root-Verzeichnis des Servers entwickeln, kann es passieren, dass Sie noch die Direktive `RewriteBase` ergänzen müssen. Hinter `RewriteBase` geben Sie dann bitte den absoluten Pfad des Verzeichnisses an, das Sie nutzen.

### 1.2.1 Der Front Controller

Somit ist nun sichergestellt, dass jede Anfrage beim Front Controller landet. Der Front Controller kann im Prinzip sehr einfach aufgebaut sein. Diese beiden Zeilen würden eigentlich schon reichen:

```
require_once 'Zend/Controller/Front.php';
Zend_Controller_Front::run('/var/www/application/controllers');
```

Das wäre eigentlich schon alles, was Sie benötigen. Der statischen Methode `run()` wird in diesem Fall einfach nur der Pfad zum Verzeichnis des Controllers übergeben. Der Rest ist Magie!

Allerdings würde ich Ihnen eine etwas umfangreichere Variante vorschlagen, da Sie dann über bessere Möglichkeiten zur Konfiguration verfügen. Diese Variante könnte so aussehen:

```
require_once 'Zend/Controller/Front.php';
// Controller-Instanz auslesen
$fc = Zend_Controller_Front::getInstance();
// Verzeichns setzen
$fc->setControllerDirectory('/var/www/app1/controllers');
// Nutzung von Views unterdrücken
$fc->setParam('noViewRenderer', true);
// So einstellen, dass Ausnahmen geworfen werden
$fc->throwExceptions(true);
// Error-Handler ausschalten
$fc->setParam('noErrorHandler', true);
$fc->dispatch();
```

**Listing 1.1** Ein einfacher Front Controller

Diese Einstellungen sehen auf den ersten Blick vielleicht ein wenig komplex aus, dafür bieten sie aber ein hohes Maß an Flexibilität.

Durch den Aufruf der Methode `Zend_Controller_Front::getInstance()`, die übrigens keine Parameter akzeptiert, wird die aktuelle Instanz des Front Control-

lers ausgelesen, der als Singleton-Pattern implementiert ist. Damit stellt das Framework sicher, dass immer nur ein Objekt des Front Controllers abgeleitet wird.

Da der Front Controller sich darum kümmern soll, dass der korrekte Action Controller eingebunden wird, muss er wissen, wo die Action Controller zu finden sind. Sie teilen ihm dies mittels der Methode `setControllerDirectory()` mit.

Um die ersten Beispiele möglichst einfach zu halten, wird die Nutzung eines Views mithilfe von `$fc->setParam('noViewRenderer', true)` unterdrückt. Die Methode `setParam()` kann natürlich noch einiges mehr für Sie tun, wie Sie später noch sehen werden.

Damit wäre die grundsätzliche Konfiguration abgeschlossen. Gerade für die Entwicklung ist es sinnvoll, noch einige zusätzliche Einstellungen festzulegen.

Da wäre zunächst der Umgang mit Exceptions. Die Model View Controller-Implementation des Zend Frameworks ist darauf ausgelegt, möglichst robust zu sein. Das heißt, dass sie eine Exception erst einmal fängt, um ein möglichst gutes Error-Handling zu ermöglichen. Allerdings führt das bei der Entwicklung und dem Debugging dazu, dass Fehler unter Umständen schwer zu finden sind. Daher ist es hilfreich, dem Controller mitzuteilen, dass Exceptions tatsächlich geworfen werden sollen, was mit `$fc->throwExceptions(true)` geschieht. In der nächsten Zeile wird mithilfe von `$fc->setParam('noErrorHandler', true)` sichergestellt, dass das interne Error-Handling unterdrückt wird. Die eigentliche Verarbeitung des Aufrufs übernimmt die Methode `dispatch()`. Etwas verallgemeinert formuliert bindet sie nun den entsprechenden Action Controller ein.

### 1.2.2 Der Action Controller

Nachdem Sie nun einen Überblick über den Front Controller erhalten haben, stellt sich die Frage, wie der Action Controller aufgebaut sein muss. Im nächsten Listing sehen Sie eine Minimalversion eines Action Controllers:

```
require_once 'Zend/Controller/Action.php';

class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
        echo "Hallo, ich bin die Index-Action
            aus dem Index-Controller";
    }
}
```

**Listing 1.2** Der erste Action Controller

Die Namen der Klasse und der Methode habe ich mir übrigens nicht einfach nur ausgedacht. Warum beide mit `Index` bzw. `index` anfangen, erfahren Sie gleich noch. An dieser Stelle möchte ich lediglich darauf hinweisen, dass der Name einer Controller-Klasse immer mit einem Großbuchstaben anfangen und auf `Controller` enden muss. Des Weiteren muss ein Action Controller immer die Klasse `Zend_Controller_Action` erweitern.

Die Namen der Methoden, die von außen ansprechbar sein sollen, müssen mit einem Kleinbuchstaben beginnen und auf `Action` enden.

Dieser Controller muss nun in einer Datei namens `IndexController.php` im Controller-Verzeichnis abgelegt werden, das dem Front Controller mit der Methode `setControllerDirectory()` mitgeteilt wurde. Der Front Controller bindet automatisch die Datei ein, leitet ein Objekt der Klasse ab und ruft die Methode `indexAction()` auf. Das Ergebnis sehen Sie in Abbildung 1.1.



**Abbildung 1.1** Die erste Ausgabe der Anwendung

Wie Sie sehen, wird der Index-Controller aufgerufen, ohne dass ich dem Front Controller mitgeteilt habe, dass er dies tun soll. Aber warum ist das so? Solange Sie dem Front Controller nichts anderes mitteilen, ruft er automatisch die Index-Action im Index-Controller auf. Auch die dazugehörige Datei wird dementsprechend automatisch eingebunden, sodass Sie sich um nichts kümmern müssen. Diese gesamte Funktionalität kann nur dann gewährleistet werden, wenn alle Komponenten korrekt benannt sind.

Natürlich werden Sie nicht nur den Index-Controller aufrufen wollen. Dann müssten Sie ja die komplette Logik in den Index-Controller einbauen, was ihn ziemlich komplex werden ließe.

Daher hier ein zweiter Controller – nennen wir ihn `FooController`:

```
require_once 'Zend/Controller/Action.php';

class FooController extends Zend_Controller_Action
{
    public function indexAction()
    {
        echo "Hier ist die indexAction aus dem FooController";
    }
}
```

Dieser Controller muss in einer Datei mit dem Namen *FooController.php* gespeichert werden. Um ihn anzusprechen, können Sie den Front Controller auf zwei Wegen aufrufen:

```
http://127.0.0.1/Foo
http://127.0.0.1/index.php/Foo
```

Die Angabe 127.0.0.1 müssten Sie eventuell durch den Namen oder die IP-Adresse des Rechners ersetzen, den Sie ansprechen wollen. In allen Beispielen dieses Kapitels werde ich immer die 127.0.0.1 nutzen.

Über beide URLs wird nun dem Action Controller mitgeteilt, dass der Controller *FooController* genutzt werden soll. Sollte die erste Variante bei Ihnen in einem »404-Fehler« resultieren, dann liegt das daran, dass das URL-Rewriting bei Ihnen nicht funktioniert. Der zweite Aufruf müsste aber funktionieren. Sollten beide Varianten eine Exception zur Folge haben, prüfen Sie bitte, ob die Pfade alle korrekt gesetzt und die Klasse und die Methode korrekt benannt sind.

Falls Sie noch eine zusätzliche Action ergänzen, die beispielsweise `barAction` heißen könnte, haben Sie schon deutlich mehr Möglichkeiten:

```
public function barAction()
{
    echo "Hier ist die barAction aus dem FooController";
}
```

Diese neue Action können Sie folgendermaßen ansprechen:

```
http://127.0.0.1/Foo/bar
http://127.0.0.1/index.php/Foo/bar
```

Wobei Sie natürlich auch *FOO*, *foo* oder *fOO* schreiben könnten. Das Zend Framework normalisiert die Schreibweise so, dass der Aufruf immer beim korrekten Controller landet.

Wie Sie sich nun sicher schon denken, ist der erste Parameter, der nach dem Slash übergeben wird, der Name des Controllers, der genutzt werden soll, und der zweite ist der Name der Action, die ausgeführt werden soll. Geben Sie weder Controller nach Action an, wird beides implizit durch `index` ersetzt, sodass die Anfrage beim `IndexController` landet und an die `indexAction` weitergereicht wird. Geben Sie nur einen Controller an, wird dieser Controller genutzt und `indexAction` in diesem Controller angesprochen. Daher sollten Sie auf jeden Fall immer eine Methode namens `indexAction` in jedem Controller haben. Natürlich können Sie den `IndexController` auch explizit ansprechen, indem Sie den Front Controller so aufrufen:

```
http://127.0.0.1/index/index
http://127.0.0.1/index.php/index/index
```

Gar nicht so schwer, oder? Bei sehr großen Anwendungen können Sie die Controller auch noch in Module untergliedern, um Ihre Anwendungen noch weiter zu strukturieren. Sie können also mehrere Verzeichnisse für die Controller nutzen.

In dem Fall übergeben Sie über die URL als ersten Wert den Namen des Moduls, gefolgt von dem Namen des Controllers und der Action. Auch in diesem Fall setzen Sie die Namen der Verzeichnisse mit der Methode `setControllerDirectory()`. Hier bekommt sie allerdings ein Array übergeben und nicht nur den Namen eines Verzeichnisses:

```
$fc->setControllerDirectory(
    array(
        'default' => '/var/www/appl/controllers',
        'cms'     => '/var/www/cms/controllers',
        'admin'   => '/var/www/admin/controllers'
    ));
```

Hierbei ist wichtig, dass es einen Eintrag namens `default` gibt. Das Verzeichnis, das damit angegeben wird, ist das Default-Verzeichnis, das beispielsweise angesprochen wird, wenn Sie das Modul nicht explizit angeben. Bei den Verzeichnissen ist zu beachten, dass die Controller immer in einem Verzeichnis namens `controllers` liegen sollten. Anders formuliert: Sie können die Verzeichnisebenen dazwischen (`appl`, `cms`, `admin`) nutzen, um Ihre Anwendung zu strukturieren.

Das Interessante dabei ist, dass der Front Controller erkennt, ob Sie den Modulnamen angegeben haben oder nicht. Die folgenden URLs würden alle den Index-Controller im Default-Modul ansprechen:

```
http://127.0.0.1/default/index/index
http://127.0.0.1/index
http://127.0.0.1
```

Wollen Sie ein anderes Modul ansprechen, müssen Sie den Namen des Moduls explizit angeben:

```
http://127.0.0.1/cms/
http://127.0.0.1/cms/index
http://127.0.0.1/cms/index/index
```

Das Ganze funktioniert so, dass der Front Controller die übergebene URL analysiert und schaut, ob als erster Parameter der Name eines Moduls übergeben wurde. Ist das nicht der Fall, wird automatisch das Default-Modul genutzt. Diese Vorgehensweise hat natürlich zur Folge, dass Sie im Default-Modul keinen Controller nutzen sollten, der den selben Namen hat wie ein Modul. Andernfalls könnte der Aufruf zweideutig sein, oder Sie müssen sicherstellen, dass dieser Controller nur mit vorangestelltem Modulnamen aufgerufen wird.

In allen Beispielen werde ich allerdings auf die Nutzung von Modulen verzichten und immer nur mit einem Controller-Verzeichnis arbeiten.

### 1.2.3 Einbinden eines Views

Nachdem Sie nun wissen, wie der Controller eingebunden wird, stellt sich die interessante Frage, wie die View-Komponente ins Spiel kommt. Dazu müssen Sie im Front Controller zunächst die Nutzung der Views aktivieren. Ändern Sie dazu die Zeile, in der der Parameter `noViewRenderer` gesetzt wird, so ab, dass ihr der Wert `false` zugewiesen wird:

```
$fc->setParam('noViewRenderer', false);
```

Versuchen Sie jetzt einen der Controller aufzurufen, resultiert das in einer `Zend_View_Exception`, die Ihnen mitteilt, dass das entsprechende Template nicht gefunden wurde.

Das bedeutet, dass Sie nun ein Template anlegen müssen. Bei den Templates handelt es sich um ganz normale HTML-Seiten, die auch PHP enthalten dürfen. Bei genauerer Betrachtung muss hier sogar ein wenig PHP enthalten sein. Ein solches Template wird direkt vom Front Controller, genau genommen vom Dispatcher, der im Hintergrund die Arbeit erledigt, mit eingebunden. Das Template, auf dem die folgenden Beispiele aufbauen, sieht so aus:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
      charset=ISO-8859-1">
    <title>Mein erstes Template</title>
  </head>
  <body>
    <?php
    ?>
  </body>
</html>
```

**Listing 1.3** Grundlegendes Template

Wie Sie sehen, ist es wirklich eine ganz normale HTML-Datei. Der PHP-Abschnitt in der Mitte wird gleich noch mit Inhalt gefüllt. Abgespeichert werden muss das Template im Ordner *views/scripts*, der sich unterhalb des Verzeichnisses *appl* befindet. Wie und wo genau Sie die Datei abspeichern, hängt davon ab, für welchen Controller und welche Action sie gedacht ist. Standardmäßig geht das Zend Framework davon aus, dass Sie für jeden Controller ein eigenes Unterverzeichnis anlegen, und dass die Datei den Namen der Action bekommt, für die das Template zuständig ist. Wenn also das Template für den *IndexController* und die *indexAction* gedacht ist, so würde es mit dem Namen *index.phtml* im Ordner *index* unterhalb von *views/scripts* gespeichert. Der komplette Pfad würde also */var/www/appl/views/scripts/index/index.phtml* lauten. Ein Template für die *barAction* aus dem *FooController* würde also unter */var/www/appl/views/scripts/foo/bar.phtml* gespeichert. Aber keine Angst, Sie können ein Template auch für mehrere Actions nutzen, sodass Sie nicht in einer Flut von Templates ertrinken. Wie das funktioniert, erfahren Sie später.

Zwar können Sie die Seite jetzt schon aufrufen, aber es gibt noch ein kleines Problem. Das fällt allerdings erst dann auf, wenn Sie sich den Quelltext anschauen, der im Browser ankommt. Dieser lautet folgendermaßen:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
      charset=ISO-8859-1">
    <title>Mein erstes Template</title>
  </head>
  <body>
  </body>
```

```
</html>
Hallo, ich bin die Index-Action
    aus dem Index-Controller
```

Grundsätzlich sieht das ja nicht schlecht aus, nur hätte der Text nicht hinter dem schließenden `</html>`-Tag landen sollen, sondern im Body der Seite. Das Problem ist, dass das System natürlich nicht wissen kann, wo die Ausgabe landen soll. Eine Ausgabe mit `echo` oder `print` ist daher keine gute Idee, falls Sie die Daten in einem Template ausgeben wollen.

Zu Beginn des Kapitels hatte ich angedeutet, dass der View im MVC aus mehreren Komponenten besteht. Nun kennen Sie das Template, aber im Hintergrund gibt es auch noch ein Objekt der Klasse `Zend_View`, das für die Ausgabe zuständig ist. Mithilfe dieses Objekts können die Action Controller auch mit dem Template »kommunizieren«, ihm also Daten übergeben.

Und das geht so: Innerhalb der Action, die ausgeführt wird, lesen Sie eine Referenz auf das `Zend_View`-Objekt aus. In diesem können Sie nun Eigenschaften mit Werten belegen. Dieses Objekt steht dann auch im Template zur Verfügung und kann über `$this` angesprochen werden. Die Methode `indexAction` im `Index-Controller` müssten Sie folgendermaßen umstellen:

```
public function indexAction()
{
    $view = $this->initView();
    $view->ausgabe = "Hallo, ich bin die Index-Action
                    aus dem Index-Controller";
}
```

**Listing 1.4** Zuweisen eines Textes

Der Action Controller legt das View-Objekt in der Eigenschaft `view` ab. Die Methode `initView()` liest diese Eigenschaft aus und gibt Ihnen eine Referenz auf das View-Objekt zurück. Zwar könnten Sie auch direkt auf die Eigenschaft zugreifen, aber die Nutzung von `initView()` hat den Vorteil, dass die Eigenschaft korrekt initialisiert wird, sofern dort noch kein Objekt der Klasse `Zend_View` enthalten ist.

Dem so ausgelesenen Objekt können Sie dann direkt die Werte zuweisen, wie Sie das in diesem Beispiel sehen. Der Name der Eigenschaft ist frei gewählt. Alternativ können Sie auch eine andere Syntax nutzen, die für Außenstehende vielleicht besser zu verstehen ist. Und zwar haben Sie die Möglichkeit, einer Eigenschaft mithilfe der Methode `assign()` einen Wert zuzuweisen. Bezogen auf dieses Beispiel würde das so aussehen:

```
$view = $this->initView();  
$view->assign('ausgabe', 'Hallo, ich bin die Index Action  
    aus dem Index Controller');
```

Wählen Sie einfach die Variante, die Ihnen am besten gefällt. In beiden Fällen geschieht dasselbe. Wobei die Aussage nicht ganz richtig ist. Der Methode `assign()` können Sie auch ein assoziatives Array übergeben. In dem Fall werden die Schlüssel des Arrays als neue Eigenschaften genutzt, denen dann die dazugehörigen Werte zugewiesen werden.

In beiden Fällen gilt allerdings, dass die Namen der Eigenschaften, die Sie vergeben, nicht mit einem Unterstrich (`_`) beginnen dürfen. Versuchen Sie einen solchen Namen zu nutzen, resultiert das in einer Exception der Klasse `Zend_View_Exception`. Der Hintergrund ist, dass die internen Eigenschaften alle mit einem Unterstrich beginnen und keinesfalls überschrieben werden dürfen.

Im Template selbst könnten Sie den Inhalt der Eigenschaft einfach mit `echo $this->ausgabe;` ausgeben. Allerdings sollten Sie natürlich immer sicherstellen, dass die Daten, die Sie ausgeben, nicht problematisch sein können. Das heißt, es muss sichergestellt sein, dass kein HTML-Code enthalten ist, der dafür sorgen könnte, dass die Darstellung leidet, oder dass der Browser, der die Daten anzeigt, mit JavaScript kompromittiert wird. Dafür ist in der Klasse `Zend_View` die Methode `escape()` deklariert. Die bessere Variante zur Ausgabe der Daten ist also diese:

```
echo $this->escape($this->ausgabe);
```

Die Methode `escape()` ersetzt HTML-Sonderzeichen (`<`, `>`, `&`, `"`) durch Entitäten. Sollten die Daten schon escaped sein, weil sie beispielsweise aus `Zend_Filter_Input` übernommen wurden, dürfen sie hier natürlich nicht noch einmal codiert werden.

Damit haben Sie die erste kleine Implementation eines MVC umgesetzt.

#### 1.2.4 Die Verarbeitungsschritte

Nachdem Sie nun einen guten Überblick haben, wie ein MVC funktioniert, möchte ich einen kleinen Rückschritt machen und noch ein paar Worte dazu verlieren, was im Hintergrund passiert.

Wie schon erwähnt, ist der Front Controller derjenige Teil, der sich um den gesamten Ablauf der Verarbeitung kümmert. Das heißt, er nimmt die eingehende Anfrage entgegen und sorgt dafür, sie an den Router weiterzugeben. Der Router analysiert die Anfrage und prüft, welcher Action Controller zu nutzen ist. Mit

dieser Information ausgestattet ist der Dispatcher dann der Nächste in der Reihe. Er bindet den Action Controller ein und ruft die entsprechende Methode auf. Danach wird die Antwort an den Client geschickt.

Damit alle Beteiligten miteinander kommunizieren können, werden die Anfrage (der Request) und die Antwort (die Response) in eigenständigen Objekten abgelegt. Auf das Request-Objekt kann von allen Komponenten zugegriffen werden, wohingegen das Response-Objekt nur von einem Action Controller aus genutzt werden kann.

Zugegebenermaßen ist das nur ein kurzer Abriss, wie die Verarbeitungsschritte aussehen. Ich meine aber, dass diese Information erst einmal ausreichend ist, um das meiste zu verstehen. Weitere Informationen dazu können Sie der Dokumentation unter der Adresse <http://framework.zend.com/manual/en/zend.controller.basics.html> entnehmen.

### 1.2.5 Übergabe von Werten

Die meisten Webanwendungen geben nicht nur Daten aus, sondern benötigen auch Informationen vom Benutzer. Meist kann der Anwender Daten in ein Formular eingeben, die dann von der Anwendung verarbeitet werden. Dabei stellt sich die Frage, wie Ihre Anwendung an diese Daten kommt.

Die einfachste Möglichkeit ist sicher, wenn Sie in altbekannter Weise direkt auf `$_GET`, `$_POST` etc. zugreifen. Auch in Zeiten des Model View Controllers funktioniert das ohne Probleme.

Allerdings sieht das Zend Framework an dieser Stelle auch eine elegantere Möglichkeit vor.

#### Das Request-Objekt

Alle Daten, die mit der clientseitigen Anfrage im Zusammenhang stehen, werden im Request-Objekt abgelegt. Dieses Request-Objekt ist im Action Controller bekannt und kann mit der Methode `getRequest()` ausgelesen werden. Das Objekt der Klasse `Zend_Controller_Request_Http` stellt verschiedene Möglichkeiten zur Verfügung, wie Sie auf die Daten zugreifen können, die mit dem Request übertragen wurden. Die Methoden, deren Name immer mit `get` beginnt, haben alle ein einheitliches Call-Interface. Sie können also alle auf dieselbe Art und Weise aufgerufen werden.

Wollen Sie einen Wert auslesen, der mit der Methode `POST` aus einem Formular übergeben wurde, können Sie die Methode `getPost()` nutzen. Sie greift im Hintergrund auf das superglobale Array `$_POST` zu und bekommt den Namen des Schlüssels übergeben, den Sie auslesen wollen. Wollen Sie also auf `$_POST['name']` zugreifen, so würden Sie im Action Controller die folgenden Zeilen benötigen:

```
$request = $this->getRequest();
$name = $request->getPost('name');
```

Die Nutzung der Methode `getPost()` hat zwei Vorteile. Erstens nutzt `getPost()` – und alle anderen `get`-Methoden tun das auch – intern ein `isset()`, sodass Sie sich keine Gedanken machen müssen, ob ein bestimmter Wert vorhanden ist. Ist ein Wert in dem entsprechenden superglobalen Array (in diesem Fall also `$_POST`) nicht vorhanden, gibt die Methode `NULL` zurück.

Der zweite Vorteil ist, dass Sie der Methode einen Default-Wert übergeben können, der zurückgegeben wird, falls der gesuchte Wert nicht im Array enthalten ist. Die Zeile

```
$name = $request->getPost('name', 'kein Name übergeben');
```

speichert in `$name` also entweder den Inhalt von `$_POST['name']` oder, falls der Schlüssel in dem Array nicht vorhanden ist, »kein Name übergeben«.

In Tabelle 1.1 finden Sie die verschiedenen Methoden, die für den Zugriff auf die superglobalen Arrays deklariert sind.

Methode	Superglobales Array
<code>getPost()</code>	<code>\$_POST</code>
<code>getQuery()</code>	<code>\$_GET</code>
<code>getServer()</code>	<code>\$_SERVER</code>
<code>getCookie()</code>	<code>\$_COOKIE</code>
<code>getEnv()</code>	<code>\$_ENV</code>

**Tabelle 1.1** Methoden zum Auslesen von superglobalen Arrays

Für `$_SESSION` ist keine Zugriffsmethode deklariert, da zur Verwaltung von Sessions die Klasse `Zend_Session` genutzt werden sollte.

Wenn Sie eine dieser Methoden aufrufen, ohne ihr einen Parameter zu übergeben, erhalten Sie übrigens das komplette superglobale Array zurück.

Eine besondere Rolle kommt noch der Methode `getParam()` zu. Sie liest einen Parameter aus, welcher dem Action Controller übergeben wurde. Wenn Sie nun fragen, was ein Parameter ist, dann kann ich nur antworten: Das kommt darauf

an. Es gibt drei Möglichkeiten, was die Methode Ihnen zurückgeben kann. Die erste Variante ist ein Parameter, der über die URL übergeben wurde. Wenn Sie Ihre Applikation also folgendermaßen

```
http://127.0.0.1/index/index/ort/Bielefeld/plz/33602
```

ansprechen, werden hier die Parameter `ort` und `plz` übergeben. Dabei bekommt `ort` bekommt den Wert »Bielefeld« zugewiesen und `plz` erhält »33602«. Das heißt, `$request->getParam('ort');` liefert »Bielefeld« zurück. Der Vollständigkeit halber sollte ich an dieser Stelle erwähnen, dass Sie auch die Parameter `module`, `controller` und `action` abfragen können. Der erste enthält in diesem Beispiel `default` und die beiden anderen `index`.

Sie können also Werte über die URL übergeben, wobei diese dann immer paarweise zusammengefasst werden. Der erste Wert nach dem Namen der Action ist der Name des ersten Parameters, worauf der dazugehörige Wert folgt. Dann kommt der Name des zweiten Parameters, der dazugehörige Wert usw.

Nun hatte ich eingangs erwähnt, dass es nicht ganz einfach zu beantworten ist, was ein Parameter ist. Das liegt daran, dass die Methode `getParam()` erst nach einem Wert schaut, der über die URL übergeben wurde, anschließend `$_GET` prüft, ob ein entsprechender Schlüssel vorhanden ist, und dann noch in `$_POST` danach sucht. Erst wenn in allen Fällen keine Daten zu finden sind, gibt die Methode `NULL` bzw. den Default-Wert, den Sie als zweiten Parameter übergeben haben, zurück. Die Methode liefert dabei jeweils den ersten Treffer, der gefunden wird.

Der Zugriff über `getParam()` empfiehlt sich meiner Ansicht nach nur dann, wenn Sie auch wirklich einen Wert über die URL übergeben. Sie sollten die Methode nicht nutzen, falls Sie Werte auslesen wollen, die mit `GET` oder `POST` übergeben wurden. Ebenso sollten Sie auch eine Namensgleichheit zwischen URL-Parametern und Namen von Formularfeldern vermeiden, da die übergebenen Werte sonst eventuell nicht eindeutig sind.

Wichtig ist, dass Sie immer beachten, dass in der URL die Angabe von Controller und Action nicht fehlen darf.

Parameter, die Sie über die URL übergeben, können beispielsweise sehr hilfreich sein, wenn es sich um Daten handelt, die Bestandteil eines Links sein sollen. Das könnte zum Beispiel die Übergabe einer ID oder der Name eines Produkts sein. Dies kann den angenehmen Nebeneffekt haben, dass Sie das Suchmaschinen-Ranking Ihrer Anwendung verbessern. Ein suchmaschinenfreundlicher Link auf eine Produktseite zu einem Apple iPod könnte in einem Shop dann beispielsweise so aussehen:

```
http://www.example.com/produkte/anzeigen/name/Apple%20iPod%20nano/  
ean/8859091650494
```

Es gibt übrigens auch noch die Methode `getParams()`, die Ihnen ein Array zurückgibt. Dieses Array beinhaltet dann alle Werte, die über die URL, GET und POST übergeben wurden. Auch hierbei gilt, dass bei Namensgleichheit der Wert erhalten bleibt, der zuerst gefunden wird. Die URL »überschreibt« GET und GET »überschreibt« POST.

Zu jeder der `get`-Methoden gibt es auch ein Gegenstück, das mit `set` beginnt und Ihnen die Möglichkeit liefert, einen Wert zu setzen.

Übrigens ist es auch möglich, die übergebenen Werte direkt in Form von Eigenschaften des Request-Objekts auszulesen. Ich hätte also auch einfach auf `$request->ort` zugreifen können. Allerdings würde ich davon abraten, da das noch weniger eindeutig ist. In diesem Fall wird zuerst geprüft, ob es einen URL-Parameter mit dem Namen gibt. Danach werden die Arrays `$_GET`, `$_POST`, `$_COOKIE`, `REQUEST_URI`, `PATH_INFO`, `$_SERVER` und `$_ENV` abgefragt, ob es einen Schlüssel mit dem Namen gibt. Wobei `REQUEST_URI` und `PATH_INFO` natürlich keine Arrays sind. Es handelt sich dabei um Eigenschaften, die vom System auf verschiedenen Wegen gefüllt werden können. Der Hintergrund ist, dass `$_SERVER['REQUEST_URI']` bzw. `$_SERVER['PATH_INFO']` nicht immer korrekt belegt sind.

Damit kennen Sie auch schon die grundlegenden Funktionalitäten des MVC. Zugegebenermaßen fehlt noch das M, nämlich das Model.

### 1.2.6 Das Model

Eigentlich bedürfte das Model keiner eigenen Überschrift, da Sie es völlig frei definieren können. Hier gibt es keine wirklichen Vorgaben. Das hat den Grund darin, dass es hier keinerlei »Magie« gibt, die das Model automatisch einbindet oder automatisch Daten daraus bezieht. Das heißt, Sie können eine eigene Klasse erstellen, diese manuell im Action Controller einbinden und wie gewohnt nutzen.

Dennoch möchte ich ein paar Punkte hierzu anmerken. Innerhalb der Verzeichnisstruktur ist das Verzeichnis *models* zum Speichern der Model-Klassen bzw. -Dateien gedacht. Natürlich könnten Sie diese auch an einem anderen Ort speichern, allerdings es ist doch sehr hilfreich, wenn Sie sich an diese Empfehlung halten. Eine klare Struktur hilft, das Dritte Ihren Code verstehen und sorgt auch dafür, dass Sie nicht lange suchen müssen, um die Komponenten zu finden.

Der zweite Punkt, den ich Ihnen empfehlen möchte, ist, dass Sie eine eigene Exception-Klasse für Ihr Model nutzen. Diese muss nicht sonderlich umfangreich sein. Eine Variante wie

```
Model_Exception extends Exception
{ }
```

reicht schon völlig aus. Damit haben Sie dann die Möglichkeit, diejenigen Exceptions, die zum Beispiel beim Datenbankzugriff oder Ähnlichem entstehen, in eine eigene Exception zu überführen. Damit meine ich ein Konstrukt wie dieses:

```
try
{
    // Datenbankzugriff
}
catch (Zend_Db_Exception $e)
{
    throw new Model_Exception ($e->getMessage(), $e->getCode());
}
```

Dies gibt Ihnen den Vorteil, sich bei der Nutzung des Models innerhalb des Action Controllers keinerlei Gedanken darum machen zu müssen, welche Klassen innerhalb des Models genutzt werden.

### 1.2.7 Error Handling

Was Sie bisher über den MVC wissen, bringt Sie schon recht weit und Sie können schon eine ganze Menge damit umsetzen. Was passiert aber, wenn ein Fehler auftritt?

Die erste Stelle, an der ein Fehler auftreten kann, betrifft die Situation, dass ein Controller oder eine Action angesprochen werden, die nicht existieren. In der momentanen Konfiguration wirft das System eine `Zend_Controller_Dispatcher_Exception`, wenn Sie versuchen, einen Controller anzusprechen, den es nicht gibt, und eine `Zend_Controller_Action_Exception`, wenn es zwar den Controller, aber nicht die entsprechende Action gibt.

Hier sind verschiedene »Schrauben« vorhanden, an denen Sie »drehen« können, um das System möglichst zuverlässig zu machen.

Zunächst ist zu überlegen, was passieren soll, wenn ein Controller angesprochen wird, den es nicht gibt. Um solche Fälle möglichst elegant zu lösen, empfiehlt es sich, im Front Controller den Parameter `useDefaultControllerAlways` mit dem Wert `true` zu belegen:

```
$fc->setParam('useDefaultControllerAlways', true);
```

Sollte der Front Controller nun mit einem Action Controller aufgerufen werden, den es nicht gibt, wird die Anfrage automatisch an den Index-Controller weitergeleitet. Das ist aber nur die halbe Miete. Der Aufruf `http://127.0.0.1/michgibtsnicht` stellt kein Problem dar. Er landet bei der `indexAction` von `IndexController`. Der Aufruf `http://127.0.0.1/michgibtsnicht/michauchnicht` resultiert aber nach wie vor in einer Exception, weil es die Methode `michauchnichtAction` nicht gibt. Um dieses Problem zu lösen, können Sie die magische Methode `__call()` nutzen. Sie ist seit PHP 5 verfügbar und wird immer dann aufgerufen, wenn eine Methode einer Klasse aufgerufen werden soll, die nicht implementiert ist. Falls also eine unbekannte Action angesprochen werden soll, wird `__call()` aufgerufen. Man könnte in der Methode auch einfach eine Exception werfen, aber das würde Sie nicht wirklich weiter bringen, da Sie nur eine Exception gegen eine andere getauscht hätten. Es ist aber auch möglich, die Anfrage an eine andere Methode weiterzuleiten, die Anfrage also zu »forwarden«. Dafür ist die Methode `_forward()` deklariert. Eine solche Implementation könnten Sie wie folgt umsetzen:

```
class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
        // normal implementierte indexAction
    }

    // Wird aufgerufen wenn die gesuchte
    // Methode nicht deklariert ist
    public function __call($methode, $parameter)
    {
        // Endet der Name der aufgerufenen Methode auf Action?
        if ('Action' === substr($methode, -6))
        {
            // Dann ein Forward auf indexAction
            $this->_forward('index');
        }
        else
        {
            throw new Exception("Methode $methode existiert nicht");
        }
    }
}
```

**Listing 1.5** Nutzung der magischen Methode `__call()`

Die Methode `__call()` prüft, ob der Name der Methode, die aufgerufen werden soll, auf `Action` endet. Ist das der Fall, erfolgte der Aufruf vom Dispatcher und daraus resultiert, dass jemand eine falsche URL genutzt hat. Sollte der Aufruf nicht auf `Action` enden, wird es sich um einen Programmierfehler handeln, der natürlich nach wie vor in einer Exception resultieren sollte.

Mit der Methode `_forward()` können Sie die Anfrage auch an einen anderen Action Controller und sogar an ein komplett anderes Modul weiterreichen. Der erste Parameter ist dabei der Name der Action, wie Sie gesehen haben. Als zweiten Parameter können Sie den Namen des Controllers angeben. Auch hierbei gilt, dass nur der eigentliche Name angegeben wird. Also beispielsweise `foo` für den `FooController`. Als dritten Parameter können Sie dann noch den Namen eines Moduls angeben.

So schön diese Technik auch sein mag – ich muss an dieser Stelle dennoch eine deutliche Warnung anbringen. Nutzen Sie Parameter, die über die URL übergeben werden, kann diese Technik schnell verwirren. Bei der folgenden Vorgehensweise würden beide URLs bei der Methode `indexAction()` aus dem `IndexController` landen:

```
http://127.0.0.1/index/index/ort/Bielefeld/plz/33602
```

```
http://127.0.0.1/ort/Bielefeld/plz/33602
```

Die erste URL ist eindeutig und unproblematisch. Die zweite hingegen würde beim Auslesen der Daten mit `getParams()` diese Informationen liefern:

```
array(4) {
  ["controller"]=>
  string(3) "ort"
  ["action"]=>
  string(9) "Bielefeld"
  ["plz"]=>
  string(5) "33602"
  ["module"]=>
  string(7) "default"
}
```

Es wäre eventuell möglich, die Werte im Request-Objekt mithilfe von `setParam()` neu zu setzen, um eine Verarbeitung möglich zu machen. Wenn Sie das tun, sollten Sie dabei aber im Hinterkopf behalten, dass es mehrere Möglichkeiten gibt, warum der Aufruf bei `__call()` gelandet ist.

Neben der Methode `_forward()` ist auch die Methode `_redirect()` deklariert. Hiermit können Sie ein echtes Redirect auf eine andere URL realisieren. Sie bekommen als ersten Parameter die Ziel-URL übergeben. Diese können Sie absolut angeben (was zu empfehlen ist) oder auch relativ. Das heißt, Sie können hier auch lediglich

```
$this->_redirect('/index/index');
```

nutzen. Wichtig ist dabei, dass Sie bei einer relativen Angabe mit einem Slash am Anfang arbeiten sollten, weil Sie sonst schnell in einer Redirect-Endlosschleife landen.

Ein Redirect hat gegenüber einem Forward den Vorteil, dass der Client den Statuscode 302 (Moved Temporarily) mitgeteilt bekommt. Haben Sie die URLs auf Ihrem Server umgestellt, und wollen Sie den Suchmaschinen mitteilen, dass die Struktur sich dauerhaft geändert hat, so übergeben Sie `_redirect()` als zweiten Parameter `array ('code'=>'301')`. Damit wird dann der HTTP-Code 301 (Moved Permanently) mitgesendet.

Nun kann es aber immer noch passieren, dass ein Fehler bei der Ausführung des Codes auftritt.

Auch in diesem Zusammenhang gibt es wieder zwei »Schrauben«, die man justieren kann. Zum ersten ist da die Methode `throwExceptions()`. Diese haben Sie schon kurz am Anfang des Kapitels kennengelernt. Normalerweise unterdrückt die MVC-Implementation die Ausgabe von Exceptions. Für die Entwicklung ist es allerdings sicherlich hilfreich, dass Exceptions geworfen werden. Für den produktiven Einsatz sollten Sie das natürlich wieder ändern. Am besten übergeben Sie der Methode den Wert `false`:

```
$fc->throwExceptions(false);
```

Alternativ können Sie die Zeile auch einfach löschen. Tritt jetzt eine Exception auf, bleibt das Browserfenster einfach leer, was natürlich auch nicht so schön ist. Um das zu verhindern, können Sie das Error-Handling-Plug-in nutzen. Und zwar habe ich das Plug-in Front Controller mit dieser Zeile ausgeschaltet:

```
$fc->setParam('noErrorHandler', true);
```

Übergeben Sie hier `false` (beachten Sie die doppelte Verneinung), ist das Plug-in wieder aktiv. Wenn jetzt ein Fehler auftritt, wird automatisch die `errorAction` im `ErrorController` aufgerufen. Dabei handelt es sich um einen ganz normalen Controller, den Sie selbst implementieren müssen. Außerdem müssen Sie auch ein Template für ihn anlegen. Dieses muss unter dem Namen `error.phtml` im Ordner `error` unterhalb von `views/scripts` abgespeichert werden.

Damit haben Sie nun die Möglichkeit, eine schicke Fehlermeldung für den Benutzer auszugeben. Nur würden Sie als Betreiber der Website nie merken, dass es ein Problem gibt, weil die Exceptions alle vom System gefangen werden. Genau genommen speichert das Error-Handling-Plug-in sie im Response-Objekt. Das ist normalerweise dazu da, die Antwort für den Client zu verwalten. Allerdings können Sie das Response-Objekt auch aus dem Action bzw. aus dem Error Controller heraus auslesen und dann die dort enthaltenen Exceptions verarbeiten. In einer ganz einfachen Variante könnte ein solcher Error Controller so aussehen:

```
require_once "Zend/Controller/Action.php";

class ErrorController extends Zend_Controller_Action
{

    public function errorAction()
    {
        $view = $this->initView();
        $response = $this->getResponse();
        $exceptions = $response->getException();
        $ausgabe = '';
        foreach ($exceptions as $exception)
        {
            $ausgabe[]=$exception->getMessage();
        }
        $view->ausgabe = $ausgabe;
    }
}
```

**Listing 1.6** Ein einfacher Error-Controller

Das Response-Objekt wird hier mit der Methoden `getResponse()` ausgelesen. Die Exceptions, die darin enthalten sind, stellt die Methode `getException()` zur Verfügung. Da es schnell passieren kann, dass mehrere Fehler auftreten, liefert die Methode ein Array mit Exceptions zurück.

Der Body-Abschnitt des dazugehörigen Templates könnte beispielsweise so aussehen:

```
<body>
    Die folgenden Fehler traten auf:<br>
<ul>
<?php
foreach ($this->ausgabe as $ausgabe)
{
```

```
        echo "<li>".$this->escape($ausgabe)."</li>";
    }
    ?>
</ul>
</body>
```

**Listing 1.7** Das Template error.phtml

Zugegebenermaßen ist das keine Variante, die Sie für eine produktive Anwendung nutzen sollten, weil Sie sonst ja auch gleich direkt die Exception ausgeben könnten. Es geht an dieser Stelle viel mehr darum, Ihnen die Funktionsweise zu zeigen. Für eine produktive Anwendung würde es sich anbieten, die Fehlermeldungen in eine Log-Datei zu schreiben, den Administrator per E-Mail zu benachrichtigen und für den Benutzer eine »kundenfreundliche« Fehlermeldung auszugeben. In diesem Zusammenhang sollten Sie auch einen Blick auf die Klasse `Zend_Log` werfen.

Hinweisen möchte ich Sie auch noch darauf, dass Sie im Manual des Zend Frameworks unter der Adresse <http://framework.zend.com/manual/en/zend.controller.plugins.html> eine etwas ausgefeiltere Variante zur Fehlerbehandlung finden.

### 1.2.8 Fortgeschrittene Techniken

In den folgenden Abschnitten finden Sie noch einige weitergehende Techniken, die Sie beim Umgang mit den Komponenten unterstützen.

#### Action-Controller

Die Klassen, die Sie als Action Controller nutzen, sollten den Konstruktor nicht überschreiben. Sollten Sie den Konstruktor dennoch einmal überschreiben wollen, so beachten Sie bitte, dass Sie den Konstruktor des Eltern-Objekts mit

```
parent::__construct($this->getRequest(),
                    $this->getResponse(), $this->getInvokeArgs())
```

aufrufen. Allerdings haben Sie auch eine andere Möglichkeit, Standardaufgaben auszuführen. Dazu können Sie die Methode `init()` implementieren. Diese wird vom Konstruktor automatisch ausgeführt:

```
public function init()
{
    // View initialisieren etc.
}
```

Wie Sie schon gesehen haben, können Sie aus einem Action Controller heraus auf das Response-Objekt zugreifen, welches für die Antwort an den Client zuständig ist. Grundsätzlich können Sie die meisten Punkte, die mit der Antwort verbunden sind, auch beeinflussen.

Ich möchte an dieser Stelle nicht auf den kompletten Funktionsumfang des Response-Objekts eingehen; Sie können ihn unter der Adresse <http://framework.zend.com/manual/en/zend.controller.response.html> nachlesen. Erwähnen möchte ich aber, wie Sie einen Header versenden. Das kann in einigen Fällen sehr hilfreich sein. Einen Header können Sie mit der Methode `setHeader()` übergeben. Sie bekommt als ersten Parameter den Namen des Headers übergeben und als zweiten den dazugehörigen Wert. Da ein Header nur dann gesetzt werden kann, wenn noch keine Daten zum Client geschickt wurden, sollten Sie vor dem Setzen des Headers mit `canSendHeaders()` prüfen, ob dieser Schritt möglich ist. Falls Sie aus einer Methode in einem Action Controller einen Header senden wollen, könnte das wie folgt umgesetzt werden:

```
public function indexAction()
{
    $response = $this->getResponse();
    if (true === $response->canSendHeaders())
    {
        $response->setHeader('Content-Type',
                            'text/html; charset=utf-8');
    }
    // Weiterer Code
}
```

Mit diesen Zeilen würde der Antwort ein UTF-8-Header mitgeschickt. Die Methode `canSendHeaders()` kann übrigens auch sofort eine Exception werfen, wenn die Header nicht mehr gesetzt werden können. Falls Sie das wünschen, übergeben Sie ihr `true`. Die Methode `setHeader()` akzeptiert auch noch einen booleschen Wert als weiteren Parameter. Übergeben Sie dort `true`, wird ein Header mit gleichem Namen, der eventuell schon gesetzt ist, überschrieben. Standardmäßig ist das allerdings nicht der Fall.

Bei der Einführung der Templates hatte ich erwähnt, dass Sie nicht unbedingt für jede Action ein eigenes Template anlegen müssen. Dabei bin ich Ihnen aber bisher die Antwort schuldig geblieben, wie Sie das realisieren. Ich empfehle Ihnen, dass Sie zumindest für jeden Action Controller ein Template behalten. Das verbessert die Übersichtlichkeit der Anwendung. Möchten Sie aber von einer Action ein anderes Template nutzen, ist das kein Problem. Um so etwas zu beeinflussen, ist ein sogenannter Helper, nämlich `ViewRenderer`, vorgesehen. Mit ihm haben

Sie die Möglichkeit, verschiedene Parameter zu verändern, welche die Darstellung und die Nutzung der Templates betreffen. Zuerst benötigen Sie das entsprechende Objekt, das Sie mit der statischen Methode `getStaticHelper()` aus der Klasse `Zend_Controller_Action_HelperBroker` auslesen. Sobald Sie das Objekt haben, können Sie die Methode `setRender()` aufrufen und ihr den Namen eines anderen Templates, genau genommen den Namen einer anderen Action, übergeben.

```
public function indexAction()
{
    $viewRenderer = Zend_Controller_Action_HelperBroker::
        getStaticHelper('viewRenderer');
    $viewRenderer->setRender('allgemein');
}
```

**Listing 1.8** Nutzung eines anderen Templates

In diesem Beispiel wird also nicht mehr das Template *index.phtml* genutzt, sondern *allgemein.phtml*.

### Plug-ins

Den Begriff Plug-in haben Sie in diesem Kapitel ja schon gelesen. Bei Plug-ins handelt es sich um Methoden, die an einem bestimmten Punkt der Verarbeitung automatisch ausgeführt werden. Mit Plug-ins können Sie eine ganze Menge Dinge erledigen lassen, die Sie sonst mehrfach manuell ausführen müssten. Einige Plug-ins bringt das Zend Framework schon mit, andere können Sie selbst implementieren. Diejenigen, die Sie selbst implementieren können, werden an einer bestimmten Stelle in den Verarbeitungsprozess eingehängt oder »eingehakt«. Daher spricht man an dieser Stelle auch von Hooks. Ist eine dieser Stellen erreicht, wird das entsprechende Plug-in automatisch ausgeführt. Ich möchte hier nicht auf alle Stellen eingehen, an denen Sie ein Plug-in aufrufen lassen können. Zwei Stellen finde ich aber sehr hilfreich. Das ist zum ersten `preDispatch` und zum anderen `postDispatch`. Hierbei handelt es sich um den Zeitpunkt direkt vor sowie direkt nach dem Dispatching. Oder anders formuliert, vor und nach dem Ausführen der Action aus dem Controller.

Müssten Sie beispielsweise bei jeder Seite einen UTF-8-Header mitsenden, würde es sich anbieten, dies mit dem `preDispatch`-Plug-in zu tun. Die Plug-ins werden alle in einer Klasse implementiert, die innerhalb des Front Controllers beim System angemeldet werden sollte.

Ein Plug-in, das jede Seite mit einem UTF-8-Header versieht, könnte so aussehen:

```
require_once 'Zend/Controller/Front.php';
require_once 'Zend/Controller/Plugin/Abstract.php';

class MeinePlugins extends Zend_Controller_Plugin_Abstract
{
    public function preDispatch(Zend_Controller_Request_Abstract
                                $request)
    {
        $response = $this->getResponse();
        if (true === $response->canSendHeaders())
        {
            $response->setHeader('Content-Type',
                                'text/html; charset=utf-8');
        }
    }
}

// normale Implementation des Front-Controllers

$fc->registerPlugin(new MeinePlugins());
$fc->dispatch();
```

**Listing 1.9** Implementation eines Plug-ins

Die Plug-ins werden alle in einer Klasse zusammengefasst. Diese Klasse muss die abstrakte Klasse `Zend_Controller_Plugin_Abstract` implementieren. Jedes Plug-in muss das Request-Objekt als Parameter akzeptieren. Am Namen des Plug-ins erkennt der Plug-in-Broker, an welcher Stelle es ausgeführt werden muss. Wie bereits gesagt, wird das `preDispatch`-Plug-in stets vor dem Dispatching und somit vor der Verarbeitung der Action ausgeführt.

Nachdem die Klasse entsprechend implementiert ist, muss nur noch ein Objekt von ihr abgeleitet werden, das mithilfe der Methode `registerPlugin()` beim Plug-in-Broker angemeldet wird.

Wollten Sie noch ein `postDispatch`-Plug-in nutzen, würde dies einfach mit der oben dargestellten Klasse implementiert.

### 1.2.9 Ein Beispiel

Da sicher das eine oder andere beim MVC ein wenig abstrakt ist, folgt jetzt noch ein kleines Beispiel. Es ist wirklich nur eine ganz kleine Anwendung, die noch einmal die Prinzipien verdeutlichen soll.

Und zwar dient die folgende »Anwendung« dazu, Telefonnummern zu verwalten. Die Möglichkeiten beschränken sich allerdings darauf, neue Telefonnummern anzulegen und bestehende anzuzeigen. Die dazu verwendete Tabelle hat den folgenden Aufbau:

```
CREATE TABLE `telefonliste` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `name` varchar(100) DEFAULT NULL,  
  `telefon` varchar(100) DEFAULT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

**Listing 1.10** Datenbanktabelle

### Der Front Controller

Der Front Controller der Anwendung macht im Endeffekt das, was Sie schon kennengelernt haben. Hier gibt es keine Besonderheiten:

```
require_once 'Zend/Controller/Front.php';  
// Controller-Instanz auslesen  
$fc = Zend_Controller_Front::getInstance();  
// Verzeichnis setzen  
$fc->setControllerDirectory('/var/www/appl/controllers');  
// Nutzung von Views unterdrücken  
$fc->setParam('noViewRenderer', false);  
// So einstellen, dass Ausnahmen geworfen werden  
$fc->throwExceptions(false);  
// Error Handler ausschalten  
$fc->setParam('noErrorHandler', false);  
$fc->setParam('useDefaultControllerAlways', true);  
  
$fc->dispatch();
```

**Listing 1.11** Der Front Controller

In diesem Front Controller wird zwar das Error Handling eingeschaltet, aber ich werde darauf verzichten, hier den Error Controller vorzustellen.

### Index Controller

Interessanter, aber auch unspektakulär ist der Index Controller umgesetzt. Er soll in diesem Beispiel nur dafür genutzt werden, ein statisches Hauptmenü auszugeben. Daher muss die Action nichts machen. Trotzdem muss die Action deklariert werden:

```

require_once 'Zend/Controller/Action.php';

class IndexController extends Zend_Controller_Action
{
    // Die Action muss nichts machen
    public function indexAction()
    { }

    // Falsche Aufrufe abfangen
    public function __call($methode, $parameter)
    {
        if ('Action' === substr($methode, -6))
        {
            $this->_forward('index');
        }
        throw new Exception("Methode $methode existiert nicht");
    }
}

```

**Listing 1.12** Der Index Controller

Auch wenn der Controller selbst nicht sonderlich viel tut, muss das Template ein wenig mehr Inhalt haben. Der Body-Bereich des Templates sieht folgendermaßen aus:

```

<body>
<h2>Willkommen zur Telefonliste!</h2>
Was darf's sein?
<ul>
    <li><a href='/neu'>Neue Nummer anlegen</a></li>
    <li><a href='/anzeigen'>Alle Nummern anzeigen</a></li>
</ul>
</body>

```

**Listing 1.13** Template für den Index-Controller

Hier sind also nur zwei statische Links auf die beiden Controller der Anwendung enthalten. Eine Action habe ich nicht angegeben; hier wird einfach die Index-Action genutzt.

### Der Controller »Neu«

Der Controller »Neu« ist dafür zuständig, neue Telefonnummern zu speichern. Es gibt hier also zwei Aktionen, die zu erledigen sind: Das Ausgeben des Formulars und das Speichern der Daten. Daher hat der Controller auch zwei entsprechende Methoden zuzüglich der Methode `__call()`. Auf deren Darstellung habe ich hier

verzichtet. Dafür finden Sie aber die Methode `init()`, die dafür zuständig ist, das Model zu initialisieren:

```
require_once "Zend/Controller/Action.php";
require_once '/appl/models/AdressenDbModel.php';

class NeuController extends Zend_Controller_Action
{
    private $model = null;
    // initialisiert das Model
    public function init()
    {
        $this->model = new AdressenModel();
    }
    // Initialisiert die Ausgabe des Formulars
    public function indexAction()
    {
        $view = $this->initView();
        $view->valueName = '';
        $view->disableName = false;

        $view->valueTelefon = '';
        $view->disableTelefon = false;

        $view->showSubmit = true;
        $view->meldung = '';
    }
    public function speichernAction()
    {
        $request = $this->getRequest();
        $view = $this->initView();
        $viewRenderer = Zend_Controller_Action_HelperBroker::
            getStaticHelper('viewRenderer');

        $viewRenderer->setRender('index');
        // Feld fuer Namen initialisieren
        $name = $request->getPost('name');
        $view->valueName = $name;
        $view->disableName = true;

        $telefon = $request->getPost('telefon');
        $view->valueTelefon = $telefon;
        $view->disableTelefon = true;
        $view->showSubmit = false;
        $view->meldung = 'Vielen Dank, die folgenden
```

```

        'Daten wurden gespeichert!';
        $daten = array ('name' => $name,
                       'telefon' => $telefon);
        $this->model->datenSpeichern ($daten);
    }
}
}

```

**Listing 1.14** Controller zum Anlegen neuer Daten

Sicher hatten Sie hier etwas anderen Code erwartet. Es wäre ein einfaches gewesen zwei Templates zu nutzen – eins mit einem Formular und das andere mit einer Ausgabe die dem Benutzer mitteilt, dass die Daten gespeichert wurden. Da ich aber nur ein Template nutzen wollte wird das Template durch die Aktionen initialisiert. Für die beiden Eingabefelder gibt es dazu jeweils zwei Eigenschaften. Im Fall des Feldes für den Namen sind das `valueName` und `disableName`. Mit der ersten Eigenschaft kann ein Wert für das Value-Attribut übergeben werden und mit der zweiten kann gesteuert werden ob das Feld aktiv ist. Wird hiermit `true` übergeben, dann wird das Attribut `disabled` in das Tag eingefügt.

Des Weiteren ist eine Eigenschaft namens `showSubmit` vorgesehen mit deren Hilfe Sie den Submit-Button ausblenden können.

In diesem Beispiel wird das Formular also zunächst zur Eingabe der Daten genutzt. Nach dem Abschicken sorgt die Methode `speichernAction()` dafür, dass die Daten an das Model übergeben werden. Darüber hinaus werden sie auch noch einmal im Formular ausgegeben, wobei die Felder nicht aktiv sind und der Submit-Button ausgeblendet wird. So kann der Benutzer noch einmal sehen was er eingegeben hatte.

Die Nutzung des Formulars, wie ich sie umgesetzt habe, ist nicht sonderlich elegant. Sollten Sie öfter mit Formularen zu tun haben, so werfen Sie einmal einen Blick auf die View-Helper und `Zend_Filter_Input`.

Wie Sie hier aber schön erkennen, muss dieser Teil der Anwendung sich nicht weiter um die Daten kümmern. Er übergibt sie einfach an das Model, das die Arbeit leistet.

Der Body-Bereich des Templates, das diese beiden Methoden nutzen, ist folgendermaßen aufgebaut:

```

<body>
  <form action="/neu/speichern" method="post">
    <?php
      echo $this->meldung;
    </?php
  </form>

```

```

?>
<table>
  <tr><td>Name</td>
    <td><input type="text" name="name"
      value ='<?php echo $this->valueName; ?>'
      <?php if ($this->disableName) {echo 'disabled';} ?>
    >
    </td>
  </tr>
  <tr><td>Telefon</td>
    <td><input type="text" name="telefon"
      value ='<?php echo $this->valueTelefon; ?>'
      <?php if ($this->disableTelefon) {echo 'disabled';} ?>
    >
    </td>
  </tr>
  <tr><td colspan="2" align="center">
    <?php if (true === $this->showSubmit)
    {
      echo "<input type='submit' value='Speichern'>";
    }
    ?>
  </td>
</tr>
</table>
</form>
</body>

```

**Listing 1.15** Template zum Anlegen der Daten



**Abbildung 1.2** Eingabe der Daten

Nun fehlt noch der Teil, der die Daten wieder ausgibt.

**Der Controller »Anzeigen«**

Dieser Controller bekommt die Daten vom Model übergeben und stellt sie dar. Die Daten werden als Array übergeben, sodass sie im Template mit einer for-each-Schleife ausgegeben werden können:

```
require_once "Zend/Controller/Action.php";
require_once '/appl/models/AdressenDbModel.php';

class AnzeigenController extends Zend_Controller_Action
{
    private $model = null;
    public function init()
    {
        $this->model = new AdressenModel();
    }

    public function indexAction()
    {
        $view = $this->initView();
        $daten = $this->model->datenLesen();

        $view->daten = $daten;
    }
}
```

**Listing 1.16** Der Controller zum Anzeigen der Daten

Auch hier ist nicht mehr viel zu erläutern. Das Template ist ähnlich einfach aufgebaut:

```
<body>
<?php
echo "<table border='1'>";
foreach ($this->daten as $zeile)
{
    echo "<tr><td>";
    echo $this->escape($zeile['name']);
    echo "</td><td>";
    echo $this->escape($zeile['telefon']);
    echo "</td></tr>";
}
echo "</table>";
?>
</body>
```

**Listing 1.17** Das Template für die Ausgabe



Homer Simpson	+1 (3735) 4810278
Monty Burns	+1 (3735) 373533
Patrick Star	+1 (921) 98151
Spongebob Squarepants	+1 (921) 953222

**Abbildung 1.3** Die Darstellung der Daten im Browser

### Das Model

Zu guter Letzt fehlt noch das Model, das in den beiden Controllern ja schon genutzt wurde. Ich habe hier auf Zend\_Db zurückgegriffen, um die Komponente möglichst einfach und schnell umsetzen zu können. Auch wenn Sie sich vielleicht noch nicht mit Zend\_Db auskennen, denke ich, dass die beiden Methoden doch gut zu verstehen sind:

```
require_once('Zend/Loader.php');
Zend_Loader::loadClass('Zend_Db');

class AdressenModel
{
    private $db = null;
    public function __construct()
    {
        // Daten für den Zugriff auf die Datenbank
        $optionen = array(
            'host'      => '127.0.0.1',
            'username' => 'root',
            'password' => '',
            'dbname'   => 'test'
        );
        // Objekt ableiten
        $this->db = Zend_Db::factory('mysqli',$optionen);
    }

    /**
     * Speichert Daten in der Datenbank
     *
     * @param array $daten
     * @return true
     */
}
```

## 1 | Der Model View Controller

```
    */
    public function datenSpeichern (array $daten)
    {
        $sql = "INSERT INTO telefonliste (name, telefon)
                VALUES (?, ?)";
        // Befehl vorbereiten
        $stmt = $this->db->prepare($sql);
        // Befehl ausführen
        $stmt->execute($daten);
    }

    /**
     * Liest alle Daten aus Tabelle
     * und gibt sie als Array zurück
     *
     * @return array
     */
    public function datenLesen ()
    {
        $sql = 'SELECT * FROM telefonliste';
        return $this->db->fetchAll($sql);
    }
}
```

**Listing 1.18** Das Model zum Zugriff auf die Datenbank

Sie sehen, an dem Model ist wirklich nichts Besonderes. Es ist einfach nur eine normale Klasse.

*Prinzipien sind der jämmerlichste Grund,  
den es gibt, um sich unbeliebt zu machen.*  
– George Bernhard Shaw

## 5 Webservices

### 5.1 Feeds mit Zend\_Feed verarbeiten

Im Zeitalter der Blogs sind Feeds für viele Menschen eine wichtige Informationsquelle. Bei Feeds handelt es sich um Informationen, die auf Basis von XML zur Verfügung gestellt werden. Informationen heißt an dieser Stelle, dass es meist Daten wie Blogbeiträge, Teaser von Zeitungsartikeln oder Ähnliches sind. In den meisten Fällen handelt es sich um eine Überschrift sowie einen kurzen Text, die teilweise mit einem Bild ergänzt werden. Üblicherweise sind auch ein Link auf den eigentlichen Artikel sowie ein Publikationsdatum enthalten. Feeds haben gegenüber dem eigentlichen Blog den Vorteil, dass man sie gut in Feedreadern zusammenfassen kann. Dabei handelt es sich um Programme oder Websites, die verschiedene Feeds einlesen, aufbereiten und darstellen, sodass man sich schnell einen Überblick über die neuen Einträge verschaffen kann.

Feeds werden in verschiedenen Formaten angeboten, wobei die Zend-Klassen mit RSS- und Atom-Feeds umgehen können.

Bitte beachten Sie bei der Arbeit mit Feeds, dass Sie immer ein Exception Handling implementieren sollten. Bei der Arbeit mit solchen Informationsangeboten kann es nämlich schnell passieren, dass Sie nicht auf eine Ressource zugreifen können.

#### 5.1.1 Feeds finden

In den meisten Fällen wird es sicher so sein, dass Sie die URL eines Feeds kennen bzw. von einer Website übernehmen können. Allerdings gibt es auch oft die Möglichkeit, die URLs der Feeds, die zur Verfügung stehen, automatisiert zu übernehmen. Dazu müssen die URLs der Feeds mithilfe des Tags `<link>` auf der eigentlichen Webseite vermerkt sein. Zend\_Feed ist in der Lage, diese Tags zu finden und die dort enthaltenen URLs zu extrahieren.

Die gefundenen Feed-URLs werden dann direkt eingelesen. Die Methode gibt Ihnen ein Array mit Feed-Objekten zurück. Leider ist momentan keine Möglichkeit vorgesehen, die URL des Feeds auszulesen.

Die Methode `findFeeds()` ist statisch deklariert, sodass Sie nicht erst ein Objekt ableiten müssen:

```
require_once 'Zend/Feed.php';
try {
    $feeds = Zend_Feed::findFeeds('http://www.mygadgetblog.de');
}
catch (Exception $e)
{
    die ($e->getMessage());
}
if (0 === count($feeds))
{
    echo "Es wurden keine Feeds gefunden";
}
else
{
    //Hier können die Feeds verarbeitet werden
}
```

**Listing 5.1** Automatisches Finden von Feeds

### 5.1.2 Allgemeines zur Verarbeitung von Feeds

Unabhängig davon, mit welcher Art von Feed Sie es zu tun haben, gibt es zwei Möglichkeiten, die Feeds einzulesen. Die erste Variante, die auch nachfolgend genutzt wird, ist ein Objekt der entsprechenden Klasse abzuleiten und dem Konstruktor die URL des Feeds zu übergeben. Die zweite Variante ist, dass Sie die statische Klasse `import()` aus der Klasse `Zend_Feed` aufrufen und ihr die gewünschte URL übergeben. Sie ermittelt dann selbstständig die Art des Feeds und liefert ein Objekt der entsprechenden Klasse zurück. Auch wenn `import()` komfortabler erscheint, stellt sich dabei unter Umständen doch das Problem, dass nicht ganz klar ist, ob ein Atom- oder ein RSS-Feed eingelesen wird. Daher gebe ich in den folgenden Beispielen dem expliziten Ableiten der Objekte den Vorzug.

Ist der Feed eingelesen, wird er mithilfe der PHP-DOM-Funktionen aufbereitet. Um auf ein Element, also den Textinhalt eines XML-Tags, zuzugreifen, nutzen Sie einfach den Namen des Tags als Methodennamen. Der Inhalt wird dann direkt als String zurückgegeben:

```
$feed->title()
```

Ist das Element nicht vorhanden, gibt der Methodenaufruf `null` zurück.

Attribute von Tags lesen Sie aus, indem Sie den Namen des Attributs als Array-Schlüssel an den Namen des Elements anhängen. In dem Fall wird der Name des Arrays nicht als Methode, sondern nur als Array-Name genutzt. Wollen Sie den Wert des `href`-Attributs eines Elements namens `link` auslesen, würde das so aussehen:

```
$feed->link['href']
```

Auch wenn Sie nur Atom- oder nur RSS-Daten verarbeiten wollen, sollten Sie beide Abschnitte lesen, da die dortigen Informationen sich auf beide Fälle beziehen.

### 5.1.3 Verarbeiten von RSS-Feeds

RSS ist das Format für Feeds, das sicher die größte Verbreitung hat. Auch wenn RSS als veraltet gilt, so ist es auf absehbare Zeit nicht wegzudenken. Das Problem ist, dass bei einem RSS-Feed, schon alleine aufgrund der Vielzahl der unterschiedlichen Unterformate, nicht ganz genau gesagt werden kann, welche Elemente vorhanden sind und welche nicht. Daher kann ich an dieser Stelle nicht dafür garantieren, dass alle Elemente, die in dem Beispiel genutzt werden, auch später bei einem Feed verfügbar sind, den Sie einlesen wollen. Am einfachsten und sichersten ist es, wenn Sie einen Blick auf die XML-Daten werfen, die der Feed zur Verfügung stellt. Dann kann der Feed sich zwar später noch ändern, aber Sie haben zunächst einmal einen guten Anhaltspunkt.

Die Verarbeitung eines RSS-Feeds könnte so umgesetzt werden:

```
<?php
require_once 'Zend/Feed/Rss.php';

header('Content-Type: text/html; charset=utf-8');

try {
    // Feed einlesen
    $feed = new Zend_Feed_Rss(
        'http://www.mygadgetblog.de/index.php/feed/');
}
catch (Exception $e)
{
    die ($e->getMessage());
}
// Titel des Blogs bzw. Feeds ausgeben
echo '<p>Titel des Blogs: '.$feed->title();
```

```

// Anzahl der Einträge ermitteln
echo '<br>Anzahl der Einträge: '.$feed->count().'\</p>';
echo '<table>';
// Über die Beiträge iterieren
foreach ($feed as $item)
{
    echo '<tr><td>';
    if ($item->link() != null)
    {
        echo '<a href="'. $item->link().' ">';
        echo $item->title();
        echo '</a>';
    }
    else
    {
        echo $item->title();
    }
    echo '</td></tr>';
    echo '<tr><td>';
    if ($item->description() != null)
    {
        echo $item->description();
    }
    else
    {
        echo 'Keine Beschreibung vorhanden';
    }
    echo '</td></tr>';
    echo '<tr><td>&nbsp;</td></tr>';
}
echo '</table>';

```

**Listing 5.2** Verarbeiten eines RSS-Feeds

Wie Sie in diesem Listing sehen, ist die Nutzung der Klasse denkbar einfach. Nach dem Einlesen des Feeds stehen die Informationen, die unterhalb des `channel`-Elements deklariert wurden, direkt zur Verfügung. Somit kann also beispielsweise der Titel des Feeds durch den Aufruf der Methode `title()` ausgelesen werden. Die Methode `count()`, die die Anzahl der Einträge liefert, ist übrigens in der Klasse definiert. Es handelt sich dabei nicht um ein Element aus dem Feed.

Um alle Einträge auszulesen, können Sie das Objekt direkt in einer `foreach`-Schleife verwenden, da die Klasse das SPL-Interface `Iterator` implementiert. Bei dem Objekt, das Sie bei jedem Durchlauf erhalten, handelt es sich um ein `item`-Element aus den XML-Daten. Alle Kind-Elemente, die unterhalb von `item` dekla-

riert sind, stehen dann wieder als Methoden zur Verfügung. Bitte beachten Sie, dass Sie auf jeden Fall auf die Methoden zugreifen sollten. Greifen Sie auf die Eigenschaften zu, so würde das zwar auch funktionieren, solange im Feed ein XML-Element mit dem entsprechenden Namen vorhanden ist. Ist dieses nicht vorhanden, liefert der Zugriff auf die Eigenschaft ein leeres DOM-Element zurück. Die Methode hingegen gibt `null` zurück, was in einer `if`-Abfrage leichter zu testen ist. Dabei sollten Sie beachten, dass die Methode nur dann `null` zurückgibt, wenn das Element nicht vorhanden ist. Ist das Element leer, gibt die Methode einen leeren String zurück. In dem obigen Beispiel wird diese Eigenschaft genutzt.

In einem normalen Vergleich werden ein leerer String und `null` als gleich eingestuft. Die `if`-Abfragen im Körper der Schleife testen somit, ob das Element, das abgefragt werden soll, im Feed vorhanden bzw. möglicherweise leer war. Der Titel eines Elements wird sicher immer vorhanden sein, weswegen ich an der Stelle auf eine weitere Abfrage verzichtet habe. Die XML-Daten aus dem Feed, die hier genutzt wurden, haben die nachfolgende Struktur:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0"
  xmlns:content="http://purl.org/rss/1.0/modules/content/"
  xmlns:wfw="http://wellformedweb.org/CommentAPI/"
  xmlns:dc="http://purl.org/dc/elements/1.1/">

  <channel>
    <title>myGadgetBlog</title>
    <link>http://www.mygadgetblog.de</link>
    <item>
      <title>LED am Einsatzwagen: so muss das :)</title>
      <link>http://www.mygadgetblog.de/index.php/led-am-
        einsatzwagen-so-muss-das/</link>
      <description>Und weil ja bald Weihnachten...</description>
    </item>
    <item>
      <title>Fotos vom Philips Fizz sehr beliebt</title>
      <link>http://www.mygadgetblog.de/index.php/fotos-vom-philips-
        fizz-sehr-beliebt/</link>
      <description>Offenbar habe ich das Foto .... </description>
    </item>
    <!-- weitere Items -->
  </channel>
</rss>
```

Diese XML-Daten sind stark gekürzt und sollen nur helfen, den PHP-Code besser verständlich zu machen. In einem echten Feed sind natürlich auch noch Datums- und Sprachinformationen und Ähnliches vorhanden.

Wie der fertig aufbereitete Feed im Browser dargestellt wird, sehen Sie in Abbildung 5.1.

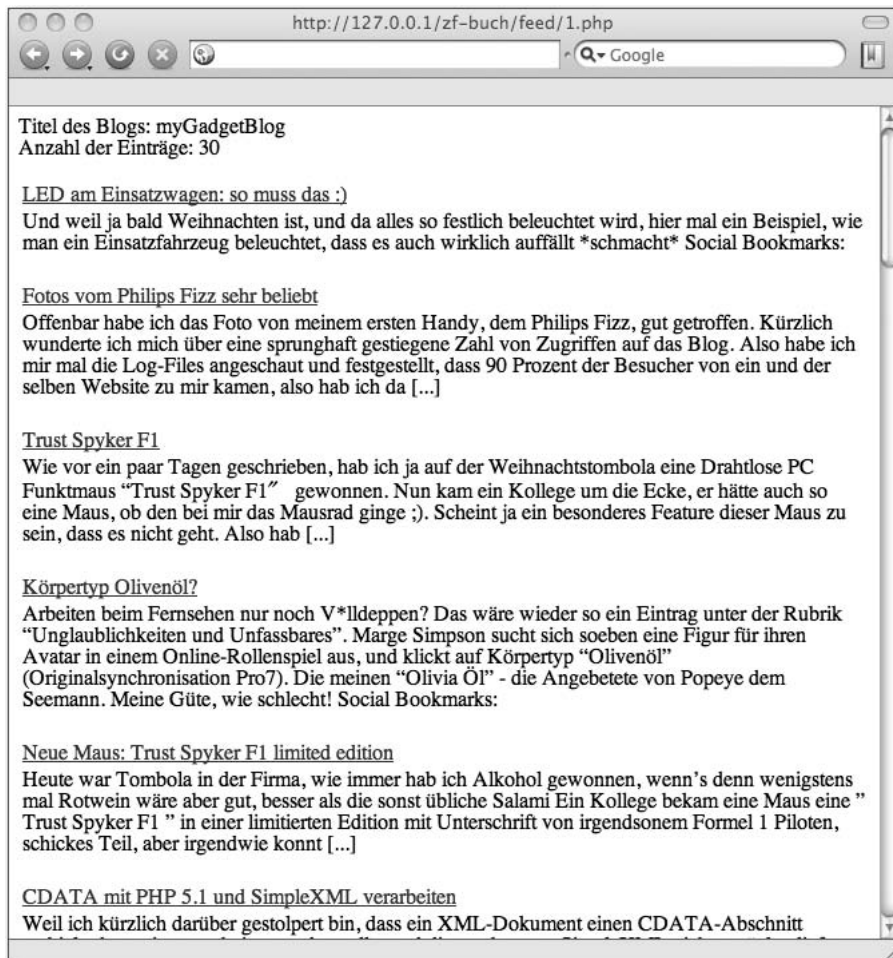


Abbildung 5.1 Ausgabe des eingelesenen Feeds

#### 5.1.4 Verarbeiten von Atom-Feeds

Möchten Sie Daten aus einem Atom-Feed verarbeiten, ist das genau so einfach wie die Nutzung von RSS. Lediglich die Elemente haben andere Namen. Die Verarbeitung eines Atom-Feeds könnte beispielsweise so aussehen:

```

<?php
require_once 'Zend/Feed/Atom.php';

header('Content-Type: text/html; charset=utf-8');

try {
    // Feed einlesen
    $feed = new Zend_Feed_Atom(
        'http://www.adminblogger.de/blog/feed/atom/');
}
catch (Exception $e)
{
    die ($e->getMessage());
}
// Titel ausgeben
echo '<p>Titel: '.$feed->title();
echo ' - '.$feed->tagline();
// Anzahl der Einträge ermitteln
echo '<br>Anzahl der Einträge: '.$feed->count().'\</p>';
echo "&<table>";
// Über die Beiträge iterieren
foreach ($feed as $item)
{
    echo '<tr><td>';
    if ($item->link() != null &&
        $item->link['href'] != '')
    {
        echo '<a href="'. $item->link['href'].'">';
        echo $item->title();
        echo '</a>';
    }
    else
    {
        echo $item->title();
    }
    echo '</td></tr>';
    echo '<tr><td>';
    echo 'Autor: '.$item->author->name();
    echo '</td><tr>';
    echo '<tr><td>';
    if ($item->summary() != null)
    {
        echo $item->summary();
    }
    else

```

```

    {
        echo 'Keine Beschreibung vorhanden';
    }
    echo '</td></tr>';
    echo '<tr><td>&nbsp;</td></tr>';
}
echo '</table>';

```

**Listing 5.3** Einlesen eines Atom-Feeds

Wie Sie sehen, ist die grundsätzliche Struktur identisch mit der bei der Verarbeitung eines RSS-Feeds. Daher möchte ich auch nur einige Kleinigkeiten genauer erläutern, wie beispielsweise die folgende `if`-Abfrage:

```

if ($item->link() !== null &&
    $item->link['href'] != '')

```

In Atom gibt es zwar ein Element namens `link`, aber die eigentliche Ziel-URL ist nicht als Text enthalten, sondern als Wert des Attributs `href`. Das heißt, der erste Teil der Abfrage prüft, ob das Element vorhanden ist. Wichtig ist hier, dass der »Nicht-Identitätsoperator« genutzt wird, da die Methode auf jeden Fall einen Leerstring zurückgibt, da `link` ein leeres Element ist. Mit dem zweiten Teil der Abfrage wird sichergestellt, dass das Attribut `href` vorhanden ist und einen Wert enthält. Wie am Anfang des Kapitels erwähnt, erfolgt der Zugriff auf Attribute dadurch, dass der Name des Elements nicht als Methode, sondern als Eigenschaft genutzt wird, die ein Array enthält. Der Schlüssel ist in diesem Fall der Name der Eigenschaft, die ausgelesen werden soll. Sollte das Attribut nicht vorhanden sein, liefert dieser Zugriff einen Leerstring zurück, generiert aber keine Notice oder einen anderen Fehler.

Der nächste interessante Punkt ist der Zugriff auf den Namen des Autors. Hier nutzt Atom zwei ineinander verschachtelte Elemente, nämlich `author` und `name`. In diesem Beispiel erfolgt der Zugriff über:

```

echo 'Autor: '.$item->author->name();

```

Das äußere Element `author` wird als Eigenschaft und nicht als Methode genutzt. Korrekterweise hätte man hier mit einer `if`-Abfrage vorher testen müssen, ob das Element vorhanden ist. Die `if`-Abfrage müsste in dem Fall so aufgebaut sein:

```

if ($item->author() !== null &&
    $item->author->name() !== null)

```

Mit dieser Abfrage wird also zunächst geprüft, ob das Element `author` vorhanden ist. Mit der zweiten Abfrage wird dann sichergestellt, dass das verschachtelte Element `name` vorhanden ist.

In diesem Fall wäre ein Zugriff auf `author()` ausreichend gewesen, da die Werte aller Elemente, die sich unterhalb von `author` befinden, automatisch mit ausgelesen werden. Zum Vergleich hier noch einmal ein Ausschnitt aus den gekürzten XML-Daten:

```
<?xml version="1.0" encoding="UTF-8"?>
<feed version="0.3" xmlns="http://purl.org/atom/ns#" xmlns:dc="http://purl.org/dc/elements/1.1/"
  xml:lang="en">
  <title>adminblogger.de</title>
  <tagline>Geschichten aus dem Leben eines Linux-SysAdmins</tagline>
  <entry>
    <author>
      <name>Marcel</name>
    </author>
    <title type="text/html" mode="escaped">
      Licht an (Aber richtig)
    </title>
    <link rel="alternate" type="text/html"
      href="http://www.adminblogger.de/blog/2007/12/08/licht-an-
      aber-richtig/" />
    <summary type="text/plain" mode="escaped">
      Anstatt zum Lichtabschalten aufzurufen, wie es Google...
    </summary>
  </entry>
  <entry>
    <author>
      <name>Marcel</name>
    </author>
    <title type="text/html" mode="escaped">
      Neues Licht f&#252;r Auto</title>
    <link rel="alternate" type="text/html"
      href="http://www.adminblogger.de/blog/2007/12/08/neues-licht-
      fuers-auto-osram-nightbreaker-seat/" />
    <summary type="text/plain" mode="escaped">
      F&#252;r mein rollendes Gef&#228;hrt, ...
    </summary>
  </entry>
</feed>
```

Die Ausgabe des Scripts sehen Sie in Abbildung 5.2.

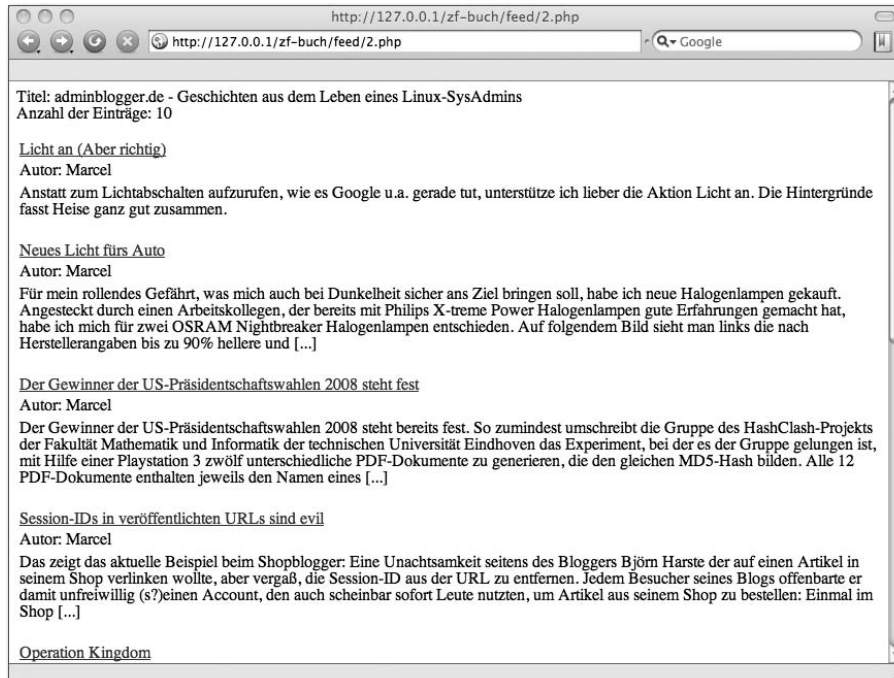


Abbildung 5.2 Ausgabe eines Atom-Feeds

### 5.1.5 Generieren von Feeds

Das Schöne an `Zend_Feed` ist, dass Sie auf eine sehr einfache Art und Weise selbst Feeds generieren können, ohne dass Sie sich um XML oder die Struktur des Dokuments kümmern müssen.

Die Daten, die im Feed enthalten sein sollen, übergeben Sie in Form eines assoziativen Arrays an die statische Methode `importArray()`. Als zweiten Parameter teilen Sie ihr dann noch mit, welches Format Ihr Feed haben soll. Das heißt, Sie können hier `'atom'` oder `'rss'` angeben, wobei `'atom'` der Default-Wert ist.

Das Format des Arrays ist leider zu komplex, um es hier komplett zu erläutern. Daher werde ich nur die wichtigsten Elemente benutzen. Die komplette Struktur können Sie der Dokumentation unter der Adresse <http://framework.zend.com/manual/de/zend.feed.importing.html> entnehmen.

Dort finden Sie auch Informationen dazu, welche Array-Schlüssel genutzt werden müssen und welche optional sind. Bitte beachten Sie, dass einige Elemente bei einem Feed-Format benötigt werden und bei einem anderen nicht. Abhängig vom Format werden einige Elemente sogar ignoriert.

In dem Array werden zunächst allgemeine Daten zum Feed deklariert, wie der Titel, die URL und Ähnliches. Die Beiträge selbst finden sich in einem indizierten Array, das unterhalb des Schlüssels `entries` zu finden ist. Jeder Beitrag muss über einige erforderliche Einträge verfügen. Das sind primär die Schlüssel `title`, `link` und `description`. Der erste deklariert den Titel des Beitrags, der zweite legt fest, unter welcher URL er aufgerufen werden kann, und mit `description` können Sie einen kurzen Abriss des Inhalts geben.

Wenn Sie das Array an die Klasse übergeben haben, erhalten Sie ein Objekt einer entsprechenden `Zend_Feed`-Klasse zurück. Rufen Sie `send()` auf, werden die Daten in XML umgewandelt, mit den notwendigen Headern versehen und direkt an den Client gesendet. Möchten Sie die Daten lieber abspeichern oder anderweitig nutzen, rufen Sie stattdessen `saveXml()` auf. Mit dieser Methode erhalten Sie die kompletten XML-Daten als String zurück.

```
require_once 'Zend/Feed.php';

// Erster Eintrag
$eintrag1 = array(
    'title' => 'Erster Eintrag',// Titel
    'link' => '127.0.0.1/blog/1',//URL zum kompletten Beitrag
    'description' => 'Mein erster kleiner Eintrag zum Testen'
);
// Zweiter Eintrag
$eintrag2 = array(
    'title' => 'Zend_Feed ist toll',
    'link' => '127.0.0.1/blog/2',
    'description'=> 'Ja, Zend_Feed ist wirklich kinderleicht'
);
$rss_daten =array(
    'title'    => 'Mein Blog', //Titel des Feeds
    'link'     => '127.0.0.1/blog', // URL des Blogs
    'charset' => 'UTF-8', // Zeichensatz
    'entries' => array( // Array für die Beiträge
        $eintrag2,
        $eintrag1
    )
);

try
{
    // RSS-Feed-Objekt ableiten
    $feed = Zend_Feed::importArray($rss_daten,'rss');
}
```

```

catch (Zend_Feed_Exception $e)
{
    die ($e->getMessage());
}
// Daten an Client senden
$feed->send();

```

Die Ausgabe dieses Scripts sehen Sie in Abbildung 5.3

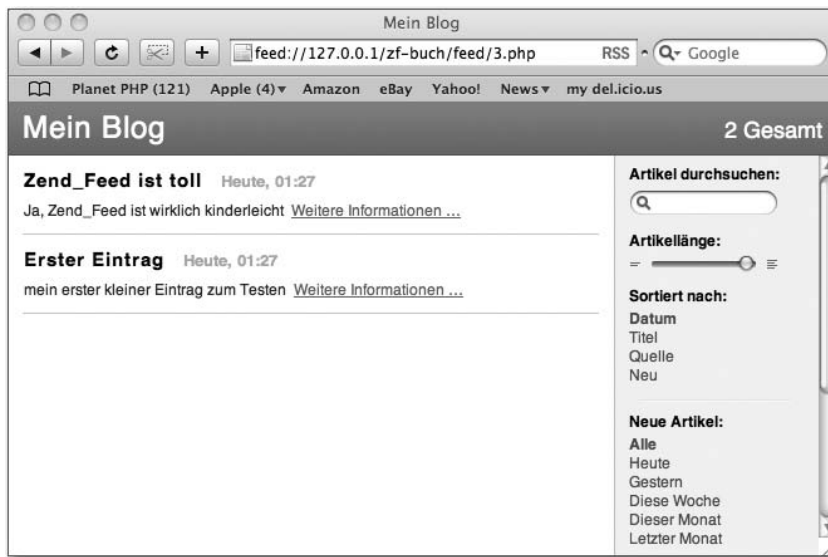


Abbildung 5.3 Selbst generierter Feed

## 5.2 Zugriff auf Amazon mit Zend\_Service\_Amazon

Amazon ist sicher einer der bekanntesten Online-Händler. Wie viele andere große Internet-Plattformen bietet auch Amazon die Möglichkeit, mithilfe einer API (Application Programmers Interface) auf das Online-Angebot zuzugreifen. Damit haben Sie die Möglichkeit, Produkte von Amazon auf Ihrer eigenen Homepage anzubieten und das Angebot in Ihr Layout zu integrieren.

Um Zugriff auf die Amazon-API zu haben, müssen Sie zunächst einen Account anlegen. Diesen können Sie unter <http://aws.amazon.com> beantragen. AWS ist übrigens die Abkürzung für Amazon Webservices. Unter AWS fasst der Anbieter alle Schnittstellen zusammen, die er zur Verfügung stellt.

Nachdem Sie einen Account angelegt haben, erhalten Sie eine E-Mail. In dieser finden Sie einen Link, unter dem Sie Ihre »Access Key ID« herunterladen können.

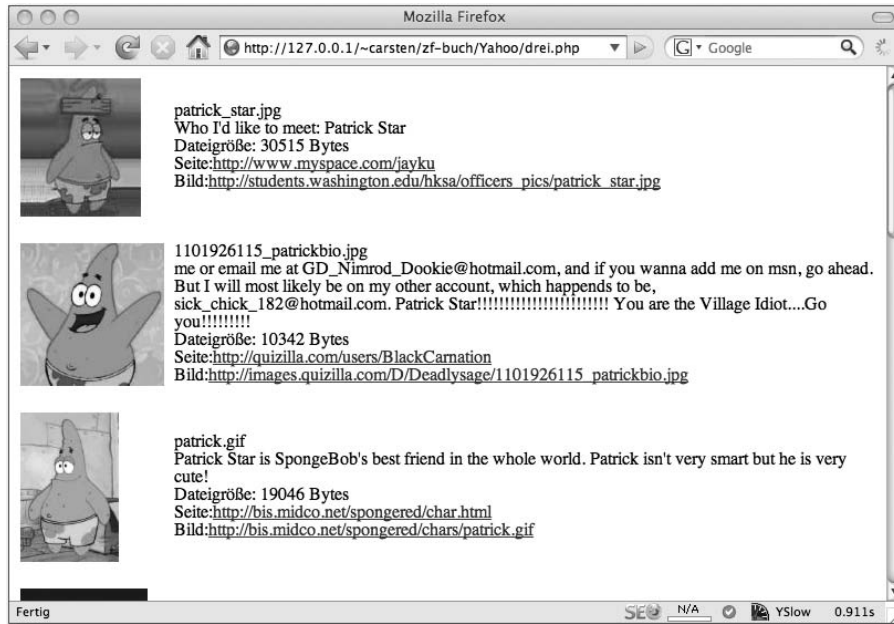


Abbildung 5.9 Ergebnis der Bildersuche

## 5.5 Zugriff auf Google-Dienste mit Zend\_Gdata

Der Suchmaschinen-Gigant Google bietet neben der eigentlichen Suchmaschine noch viele andere Dienste im Internet an. So bietet Google Dienste wie eine Tabellenkalkulation, eine Textverarbeitung oder auch einen Terminplaner an. Des Weiteren gehören inzwischen auch andere Dienste zum Google-Konzern, bei denen das vielleicht nicht auf Anhieb ersichtlich ist, wie beispielsweise YouTube.

Viele dieser Dienste können über eine einheitliche Schnittstelle, der Google Data API, angesprochen werden. Diese API, die auch kurz GData genannt wird, basiert auf dem Atom Publishing Protocol. GData unterstützt zurzeit noch nicht alle Google-Angebote, sodass es beispielsweise leider noch nicht möglich ist, darüber die eigentliche Suchmaschine zu nutzen.

Momentan unterstützt das Zend Framework den Zugriff auf die APIs von: Google Calendar, Google Spreadsheets, Picasa, Youtube, Google Base und Google Documents. Die Zugriffsmöglichkeiten sind unterschiedlich umfangreich implementiert. Ich habe mich hier dafür entschieden, mich auf die Vorstellung der Calendar- und Spreadsheets-Pakete zu beschränken. Diese kennen schon recht viele

Möglichkeiten. Haben Sie die Nutzung dieser Pakete verstanden, so können Sie sich sicherlich schnell in die Nutzung der anderen Pakete einarbeiten.

Resultierend aus der Tatsache, dass die Dienste alle über dieselbe API angesprochen werden, gibt es natürlich viele Gemeinsamkeiten bei den entsprechenden Paketen des Zend Frameworks. Daher möchte ich Ihnen zunächst die Methoden und Techniken vorstellen, die bei allen Google-Klassen Verwendung finden, und dann auf die Authentifikation eingehen.

### 5.5.1 Allgemeines zu Zend\_Gdata

Google Data basiert auf dem Atom Publishing Protocol. Dabei handelt es sich um ein XML-basiertes Protokoll, bei dem die Daten über das HTTP-Protokoll ausgetauscht werden.

Der Zugriff auf die verschiedenen Dienste wird über Feeds realisiert. Das heißt, wenn Sie auf eine bestimmte Informationsressource zugreifen wollen, ist dafür immer eine Feed-URL definiert, wie beispielsweise *<http://www.google.com/calendar/feeds/default/owncalendars/full>*. In einer solchen URL sind verschiedene Informationen enthalten. In diesem Fall wird der Zugriff auf die Kalender eines Benutzers angestrebt.

Abhängig davon, welche Informationen Sie benötigen, müssen Sie sich gegenüber dem System authentifizieren. Wenn Sie beispielsweise öffentliche Daten aus einem Blog auslesen, ist keine Anmeldung am System notwendig. Möchten Sie aber einen neuen Eintrag in Ihrem Kalender anlegen, so müssen Sie sich natürlich zuvor anmelden. Übrigens wird auch das Schreiben von Daten über solche URLs verwaltet.

Die Nutzung der verschiedenen Google-Dienste ist in `Zend_Gdata` – da die gleichen Basisklassen genutzt werden – recht ähnlich implementiert. Vor diesem Hintergrund werde ich diejenigen Funktionalitäten, die in allen Paketen vorkommen, lediglich bei Google Calendar erläutern. Auch wenn Sie sich vielleicht nur mir Spreadsheets beschäftigen wollen, sollten Sie vorher das Kapitel zu Google Calendar lesen.

### 5.5.2 Authentifikation

Die Authentifikation gegenüber der Google-API kann auf zwei Wegen erfolgen. Zum ersten können Sie die »übliche« Vorgehensweise mithilfe von Benutzernamen bzw. E-Mail-Adresse und Passwort nutzen. Die zweite Möglichkeit ist die Anmeldung mithilfe von Googles Account Authentication.

Für beide Vorgehensweisen ist jeweils eine eigene Klasse definiert, in der es eine Methode namens `getHttpClient()` gibt, mit welcher ein neuer, authentifizierter Client abgeleitet werden kann.

### Authentifikation via Client-Log-in

Eine Authentifikation mithilfe eines Client-Log-ins bietet sich immer dann an, wenn unterschiedliche Benutzer eine Applikation nutzen sollen, die sich mithilfe ihrer E-Mail-Adresse und ihres Passwortes anmelden können. In den folgenden Beispielen wurden E-Mail-Adresse und Passwort direkt in den Code integriert, um die Strukturen einfach zu halten.

Ein Client-Log-in, mit dem Sie sich bei Google Calendar anmelden können, sieht im einfachsten Fall so aus:

```
$email = 'zf-buch@netviser.de';
$password = 'totalgeheim';

$client = Zend_Gdata_ClientLogin::getHttpClient(
    $email,$password,'cl');
$cal = new Zend_Gdata_Calendar($client);
```

In diesem kleinen Beispiel wird ein neuer Client generiert, indem der statischen Methode `getHttpClient()` die E-Mail-Adresse, also sozusagen der Log-in-Name, und das Passwort übergeben werden. Der dritte Parameter, der String `'cl'`, legt fest, dass eine Verbindung mit dem Calendar-Dienst aufgebaut werden soll. Hier können Sie eine der Konstanten aus Tabelle 5.8 nutzen, wobei Sie allerdings auch auf den universellen String `xapi` zurückgreifen können. Damit können Sie sich zu allen Services verbinden.

Abkürzung	Service
cl	Calendar
blogger	Blogger
gbase	Google Base
wise	Google Spreadsheet
apps	Google Apps
lh2	Picasa Web Albums
youtube	YouTube
xapi	Standard Client

**Tabelle 5.8** Abkürzungen für Google-Dienste

Allerdings ist diese Vorgehensweise nicht perfekt. Neben den »üblichen« Exceptions, die auftreten können, besteht bei dieser Form der Authentifikation noch

das Problem, dass eine Exception der Klasse `Zend_Gdata_App_CaptchaRequiredException` geworfen werden kann. Das ist dann der Fall, wenn Google die Nutzung eines CAPTCHA verlangt. Die Nutzung eines CAPTCHA verlangt Google immer dann, wenn zu viele Log-in-Versuche in zu kurzer Zeit stattgefunden haben. Sollte dieser Fall eintreten, so können Sie mit der Methode `getCaptchaUrl()` die URL des CAPTCHA-Bildes auslesen, und die Methode `getCaptchaToken()` liefert ein Token, welches Sie bei der nächsten Anfrage zusammen mit der Benutzereingabe an den Server senden müssen, damit diese verifiziert werden kann.

```
require_once 'Zend/Gdata/ClientLogin.php';
require_once 'Zend/Gdata/Calendar.php';

$email = 'zf-buch@netviser.de';
$password = 'totalgeheim';
$service = 'cl';
$client = null;
$source = Zend_Gdata_ClientLogin::DEFAULT_SOURCE;

// Wurden Daten aus dem Formular übergeben?
if (true === isset ($_POST['captcha']))
{
    // Daten aus Formular übernehmen
    $login_captcha = $_POST['captcha'];
    $login_token = $_POST['token'];
}
else
{
    // Keine Daten erhalten => null übergeben
    $login_captcha = null;
    $login_token = null;
}

try
{
    $client = Zend_Gdata_ClientLogin::getHttpClient($email,
        $password, $service, $client, $source,
        $login_token, $login_captcha);
}
catch (Zend_Gdata_App_CaptchaRequiredException $e)
{
    // Google verlangt ein CAPTCHA => Formular ausgeben
    echo 'Bitte geben Sie den Code ein, den Sie auf dem
        Bild sehen.<br>';
}
```

```

echo "<form method='post' action='$_SERVER[PHP_SELF]'">";
// CAPTCHA-Bild ausgeben
echo '<br>';
echo 'Code: <input type="text" name="captcha"><br>';
// Token in einem versteckten Feld übergeben
echo '<input type="hidden" value="'. $e->getCaptchaToken().
      '" name="token" ><br>';
echo '<input type="submit" value="Abschicken">';
echo '</form>';
}
catch (Zend_Gdata_App_AuthException $e)
{
    // Hier werden Authentifikationsfehler abgefangen
    die ('Bei der Authentifikation trat folgender Fehler auf: ',
        $e->getMessage());
}
catch (Exception $e)
{
    // Hier werden sonstige Exceptions abgefangen
    die ('Der folgende Fehler ist aufgetreten: '. $e->getMessage());
}

$cal = new Zend_Gdata_Calendar($client);

```

**Listing 5.11** Authentifikation via Client-Log-in

In Listing 5.11 finden Sie eine etwas komplexere Lösung zum Ableiten eines Clients. Die Methode `getHttpClient()` wird auch hier wieder genutzt, wobei in diesem Beispiel alle Parameter genutzt werden und nicht nur die obligatorischen. Mit dem vierten Parameter könnten Sie ein Objekt der Klasse `Zend_Http_Client` übergeben. Das bietet sich dann an, wenn Sie bereits ein Objekt dieser Klasse besitzen. Übergeben Sie an dieser Stelle `null` oder lassen Sie den Parameter ganz wegfallen, leitet die Methode selbst ein Objekt der Klasse ab.

Bei dem fünften Parameter, in diesem Beispiel also die Variable `$source`, handelt es sich um die »Kennung« der Applikation. In diesem Beispiel wird der Wert der Konstante `DEFAULT_SOURCE` übergeben, welche standardmäßig mit dem String `Zend-ZendFramework` belegt ist. An dieser Stelle können und sollten Sie allerdings die Kennung Ihrer Applikation übergeben. Entsprechend der Google-Dokumentation sollte die Kennung nach folgendem Schema aufgebaut sein: `'firma-applikation-versionsID'`.

An diesen Parameter schließen sich die Parameter an, mit denen das Token und der Code vom CAPTCHA-Bild übergeben werden. Diese sind natürlich nur dann

erforderlich, wenn Google nach einem CAPTCHA verlangt. Mit anderen Worten: Beim vorhergehenden Aufruf der Methode wurde eine Ausnahme des Typs `Zend_Gdata_App_CaptchaRequiredException` geworfen. In dem Fall wird dann dasjenige Formular ausgegeben, welches in dem `catch`-Block definiert wird. Die URL des Bildes und das Token werden mithilfe der Methoden `getCaptchaUrl()` und `getCaptchaToken()` ausgelesen und in das Formular integriert. Wie ein solches CAPTCHA-Formular aussehen kann, sehen Sie in Abbildung 5.10.



**Abbildung 5.10** CAPTCHA bei einem Client-Log-in

Wird das Formular abgeschickt, werden die Daten aus dem Formular für den nächsten Verbindungsaufbau genutzt.

Der zweite `catch`-Block dient dazu, Authentifikationsfehler, also wenn die Kombination aus Log-in und Passwort nicht gültig ist, abzufangen. Leider liefert die Google-Dokumentation momentan noch keine exakten Informationen, welche Codes hier zurückgeliefert werden. Sie enthält lediglich die Information, dass die Fehlermeldung eine Erläuterung des Fehlers enthält.

Das Beispiel aus Listing 5.11 ist natürlich ein klein wenig realitätsfremd, da die E-Mail-Adresse und das Passwort hart in die Anwendung codiert sind. Bei einer Anwendung im »echten Leben« würden diese Daten sicher eher aus einem Formular oder einer Konfigurationsdatei übernommen.

### Authentifikation via Account-Authentication

Die Account-Authentication nutzt eine gänzlich andere Vorgehensweise, die auf den ersten Blick ein wenig befremdlich erscheinen mag. Die Idee ist allerdings gut und bringt einige Vorteile mit sich. Sie wird übrigens auch als `AuthSub` bezeichnet, da Google eine solche Authentifikation dann durchführt, wenn ein `AuthSub-Request` erfolgte.

Ihre Applikation leitet den Benutzer zunächst auf eine Seite von Google, wobei die Applikation ihre eigene URL mit übergibt. Auf der Google-Seite wird der Benutzer dann gefragt, ob er der Applikation Zugriff auf die Daten gewähren möchte, und loggt sich ein. Sind die Daten korrekt, schickt Google den Benutzer zurück zur ursprünglichen Anwendung, wobei dieser ein Token übergeben wird. Bei diesem Token handelt es sich um ein »One-Time-Use-Token«, das Sie nur für einen Request nutzen können. Da Sie in den meisten Fällen aber mehrere Anfragen benötigen werden, sollten Sie die eine Anfrage, die Ihnen zur Verfügung steht, dazu nutzen, um aus dem One-Time-Use-Token ein Session-Token zu machen, das dann für die gesamte Sitzung gültig ist. Mithilfe dieses Session-Tokens kann die Anwendung sich bei allen folgenden Requests authentifizieren.

Diese Vorgehensweise bietet sich immer dann an, wenn viele verschiedene Benutzer eine Anwendung nutzen sollen. Insbesondere dadurch, dass das Log-in auf einer Seite von Google erfolgt, haben die Benutzer auch ein größeres Gefühl der Sicherheit.

So viel zur Theorie. Die URL, die Ihre Applikation aufrufen muss, damit sich der Benutzer authentifizieren kann, setzt sich aus verschiedenen Informationen zusammen, die Sie an die statische Methode `getAuthSubTokenUri()`, die in der Klasse `Zend_Gdata_AuthSub` definiert ist, übergeben sollten. Als ersten Parameter übergeben Sie die URL Ihrer Applikation, damit Google den Benutzer auch wieder zurückschicken kann. Der zweite Parameter ist die URL der Google-Anwendung, die genutzt werden soll, da Google natürlich wissen muss, für welchen Dienst authentifiziert werden soll.

Danach folgen zwei optionale Parameter, die entweder 0 oder 1 als Wert haben. Der erste von beiden legt fest, ob ein sicheres Token genutzt werden soll. Diese Möglichkeit steht allerdings nur dann zur Verfügung, wenn Sie Ihre Applikation vorher bei Google registriert haben.<sup>8</sup> Somit wird an dieser Stelle üblicherweise eine 0 zu finden sein. Der zweite Wert teilt Google mit, ob das One-Time-Use-Token in ein Session-Token konvertiert werden darf. Hier wird also üblicherweise eine 1 stehen.

Der Methodenaufruf liefert Ihnen die URL zurück, auf die der Benutzer weitergeleitet werden muss. Das kann über einen Link oder direkt durch die Funktion `header()` geschehen.

Nachdem der Benutzer sich angemeldet und zugestimmt hat, die Applikation zu authentifizieren, wird er zurückgeleitet, wobei das One-Time-Use-Token über die

---

<sup>8</sup> Möchten Sie Ihre Applikation registrieren lassen, finden Sie hier weitere Informationen: <http://code.google.com/apis/accounts/RegistrationForWebApps.html>

URL übergeben wird. Das wiederum übergeben Sie an die statische Methode `getAuthSubSessionToken()`, die es in ein Session-Token konvertiert. Ein solcher Log-in-Mechanismus kann dann so aussehen:

```
<?php
require_once('Zend/Gdata/AuthSub.php');
require_once('Zend/Gdata/Calendar.php');

session_start();

// Ist die Applikation noch nicht authentifiziert?
if (false === isset($_SESSION['token']))
{
    // Wurde ein Token über die URL übergeben?
    if (false === isset($_GET['token']))
    {
        // Kein Token über URL => dann Link ausgeben
        // URL der Kalender-Applikation
        $url_kalender = 'http://www.google.com/calendar/'.
            'feeds/default/private/full';
        // URL der eigenen Applikation
        $url_applikation = 'http://'. $_SERVER['SERVER_NAME'].
            $_SERVER['REQUEST_URI'];

        $sicheres_token = 0;
        $session_token_ok = 1;

        // Link ausgeben den der User anklicken kann
        $google_url = Zend_Gdata_AuthSub::getAuthSubTokenUri(
            $url_applikation, $url_kalender,
            $sicheres_token, $session_token_ok);
        echo "Klicken Sie <a href='$googleUri'>hier</a>
            um die Applikation zu authentifizieren.";
        exit();
    }
    else
    {
        // Konvertieren des One-Time-Use-Tokens in ein Session-Token
        // und Speichern in der Session
        try
        {
            $_SESSION['token'] = Zend_Gdata_AuthSub::
                getAuthSubSessionToken($_GET['token']);
        }
        catch (Zend_Gdata_App_AuthException $e)
        {
```

```

        die ('Konnte One-Time-Use-Token nicht in
            Session-Token konvertieren
            <br>Grund: '.$e->getMessage());
    }
}
}

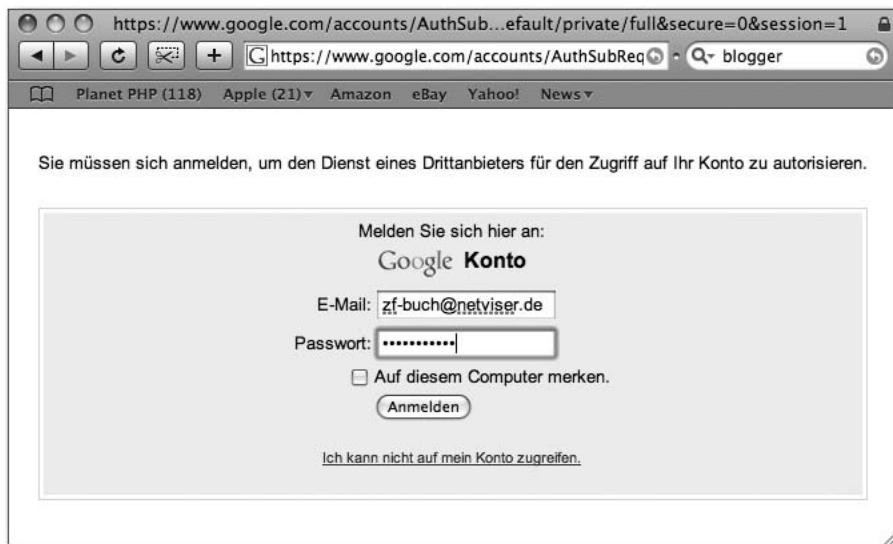
// Client mit Token initialisieren
$client = Zend_Gdata_AuthSub::getHttpClient($_SESSION['token']);

// Calendar-Client ableiten
$cal = new Zend_Gdata_Calendar($client);

```

**Listing 5.12** Authentifikation mit AuthSub

In Abbildung 5.11 und Abbildung 5.12 sehen Sie die Google-Seiten, die der Benutzer in diesem Beispiel zu sehen bekommt.

**Abbildung 5.11** Log-in bei Google

Um den Benutzer bzw. die Applikation wieder auszuloggen, müssen Sie Google mitteilen, dass das Token ungültig gemacht werden soll. Dazu ist die statische Methode `AuthSubRevokeToken()` deklariert, der Sie einfach das Token übergeben können:

```

Zend_Gdata_AuthSub::AuthSubRevokeToken($_SESSION['token']);
unset($_SESSION['token']);

```



Abbildung 5.12 Information über die Weiterleitung

Aus Gründen der Übersichtlichkeit werde ich in den folgenden Kapiteln immer eine Authentifikation via Client-Log-in verwenden, wobei ich auf die Prüfung einer CAPTCHA-Exception verzichte.

### Exception Handling bei Zend\_Gdata

Die Gdata-Pakete verhalten sich beim Exception Handling ein wenig anders als die anderen Klassen im Zend Framework. Aufgrund der Vielzahl der Klassen wären so viele unterschiedliche Exceptions auch schlecht handhabbar.

Momentan nutzen die Gdata-Pakete die folgenden Exceptions:

- ▶ `Zend_Gdata_App_AuthException`  
Diese Exception-Klasse wird immer dann genutzt, wenn das Log-in nicht erfolgreich war, weil es nicht möglich war, sich mit der Kombination aus Benutzernamen und Passwort anzumelden.
- ▶ `Zend_Gdata_App_CaptchaRequiredException`  
Nutzen Sie ein Client-Log-in, kann es passieren, dass Google bei zu vielen Log-in-Versuchen in zu kurzer Zeit die Nutzung eines CAPTCHA verlangt. In diesem Fall wirft `Zend_Gdata` die obige Exception.

- ▶ `Zend_Gdata_App_HttpException`  
Eine Exception dieses Typs wird geworfen, wenn ein Fehler bei der Kommunikation mit Google auftritt. Das kann beispielsweise dann passieren, wenn Sie eine falsche URL nutzen.
- ▶ `Zend_Gdata_App_InvalidArgumentException`  
Dieser Typ von Exception begegnet Ihnen dann, wenn Sie bei einer Abfrage falsche Parameter spezifizieren.
- ▶ `Zend_Gdata_App_BadMethodCallException`  
Diese Exception wird genutzt, wenn eine HTTP-Methode genutzt wurde, die der aktuell genutzte Dienst nicht unterstützt. Dieser Typ von Exception sollte nicht auftreten, solange Sie die vordefinierten Methoden nutzen.

### 5.5.3 Nutzung von Google Calendar

Mit Google Calendar bietet Google einen sehr umfangreichen Terminmanager an, der über das Internet genutzt werden kann. Dank modernster Technik ist er ähnlich komfortabel wie die Terminverwaltung von Outlook und ähnlichen Tools. Sollten Sie sich noch nicht mit Google Calendar beschäftigt haben, wäre das jetzt ein guter Zeitpunkt. Es dürfte es Ihnen leichter machen, die nachfolgenden Funktionen zu verstehen. Einen neuen Account können Sie unter <http://www.google.com/calendar> anlegen.

Aufgrund der Komplexität der API kann ich sie in diesem Zusammenhang nicht komplett erläutern. Das Kapitel wird Ihnen aber einen guten Einstieg bieten. Sollten Sie weitere Informationen benötigen, dann ist die Dokumentation der Google-API eine gute Anlaufstelle. Sie finden sie unter der Adresse [http://code.google.com/apis/calendar/developers\\_guide\\_protocol.html](http://code.google.com/apis/calendar/developers_guide_protocol.html).

#### Auslesen von Events

Das Auslesen von Terminen, die im Kalender eingetragen sind, ist recht einfach. Nach der Authentifikation müssen Sie der Klasse nur mitteilen, dass Sie die Termine auslesen wollen. Dabei erhalten Sie ein Feed-Objekt zurück. In diesem Fall handelt es sich um ein Objekt der Klasse `Zend_Gdata_Calendar_EventFeed`. Hierbei handelt es sich um eine Kind-Klasse der Klasse `Zend_Gdata_Feed`. Alle Funktionalitäten, die für Sie wichtig sind, sind auch in dieser Eltern-Klasse realisiert. Auch die Klasse der anderen Feeds, die Sie an anderer Stelle zurückerhalten, sind von dieser Klasse abgeleitet, sodass Sie eine weitestgehend einheitliche API haben.

Die Klasse `Zend_Gdata_Feed` implementiert das SPL-Interface `Iterator`, sodass Sie das entsprechende Objekt direkt in einer `foreach`-Schleife nutzen können. Innerhalb der Schleife stehen dann die einzelnen Objekte für die einzelnen Einträge zur Verfügung. In diesem Fall handelt es sich dabei um die Klasse `Zend_Gdata_Calendar_EventEntry`. Auch hier gilt, dass es eine Elternklasse gibt, die einen Großteil der Methoden deklariert. In diesem Fall heißt sie `Zend_Gdata_Entry`.

In der Schleife laufen Sie nun über die einzelnen Einträge aus dem Kalender. Wie bei einem Feed üblich, handelt es sich dabei um eigenständige XML-Knoten, die weitere Elemente beinhalten. Um die dort enthaltenen Informationen auszulesen, gibt es mehrere Möglichkeiten. Möglichkeit eins ist, dass Sie das `Entry`-Objekt nutzen und das fragliche XML-Element einfach als Eigenschaft anhängen. Das könnte beispielsweise so aussehen:

```
foreach ($eintraege as $eintrag)
{
    echo $eintrag->title."<br>";
}
```

Diese Schleife würde also von jedem `Entry`-Objekt die Eigenschaft `title` bzw. den Inhalt des XML-Elements `title` ausgeben. Die andere Möglichkeit ist, dass Sie die Methode `getTitle()` aufrufen, die Ihnen auch den Inhalt zurückgibt. Die Methode `getTitle()` gehört zu den magischen Methoden, die das System kennt. Es gibt eine ganze Menge dieser magischen Methoden, die oft auch genutzt werden, um neue Objekte abzuleiten. Diese Factorys beginnen dann jeweils mit `new`, wie Sie noch sehen werden.

Nun stellt sich die interessante Frage, woher man überhaupt weiß, dass es das Element `title` gibt. Die Frage ist leider nur mithilfe der Google-Dokumentation zu beantworten. Dort finden Sie für die verschiedenen Dienste Informationen, wie die XML-Nachrichten aufgebaut sind. Sie finden Beispiele wie dieses:

```
<entry>
  <id>http://www.google.com/calendar/feeds/jo@gmail.com/private
  /full</id>
  <published>2006-03-30T22:00:00.000Z</published>
  <updated>2006-03-28T05:47:31.000Z</updated>
  <title type='text'>Lunch with Darcy</title>
  <content type='text'>Lunch to discuss future plans.</content>
  <link rel='self' type='application/atom+xml'
  href='http://www.google.com/calendar/feeds/jo@gmail.com/private-
  magicCookie/full/entryID'></link>
  <author>
```

```

        <name>Jo March</name>
        <email>jo@gmail.com</email>
    </author>
    <gd:transparency
value='http://schemas.google.com/g/2005#event.opaque'>
</gd:transparency>
    <gd:eventStatus
value='http://schemas.google.com/g/2005#event.confirmed'>
</gd:eventStatus>
    <gd:when startTime='2006-03-30T22:00:00.000Z'
        endTime='2006-03-30T23:00:00.000Z'></gd:when>
    <gd:where></gd:where>
</entry>

```

Dieses Beispiel aus der Google-Dokumentation habe ich deutlich gekürzt, aber die wichtigsten Elemente können Sie erkennen.

Ein wenig problematisch ist die Tatsache, dass einige Elemente in Form von Arrays bereitgestellt werden und andere nicht. Leider konnte ich weder in der Dokumentation von Google noch in der des Zend Frameworks einen Hinweis finden, wann ein Element ein Array ist und wann nicht. Hier kann ich Sie leider nur darauf verweisen, dass Sie dies ausprobieren. So werden die Elemente `<author>` oder `<gd:when >` beispielsweise als Array mit einem Element vorgehalten. Um nun einen Kalender komplett auszulesen, könnten Sie folgendermaßen vorgehen:

```

require_once 'Zend/Gdata/ClientLogin.php';
require_once 'Zend/Gdata/Calendar.php';
require_once 'Zend/Gdata/Calendar/EventQuery.php';

require_once 'Zend/Date.php';

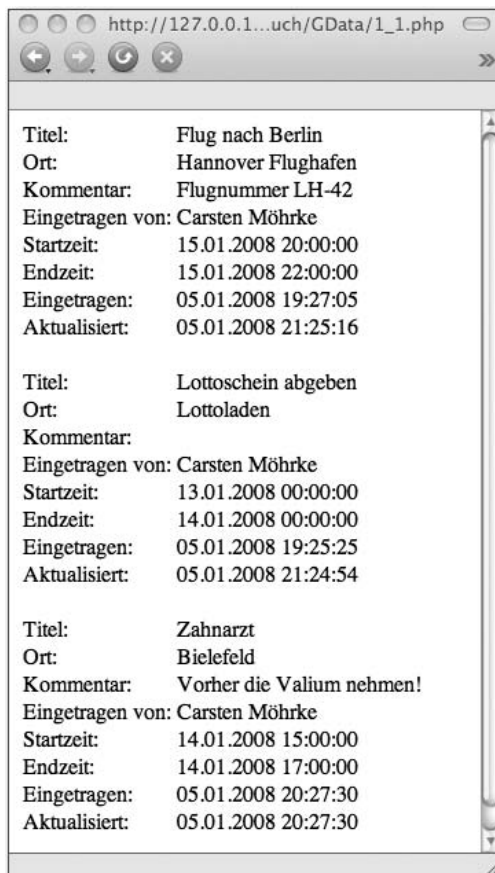
header ('Content-Type: text/html; charset=utf-8');

$email = 'zf-buch@netviser.de';
$password = 'totalgeheim';
$service = 'cl';
$client = null;
$source = Zend_Gdata_ClientLogin::DEFAULT_SOURCE;
try
{
    $client = Zend_Gdata_ClientLogin::getHttpClient($email,
        $password, $service, $client, $source);
}
catch (Exception $e)

```



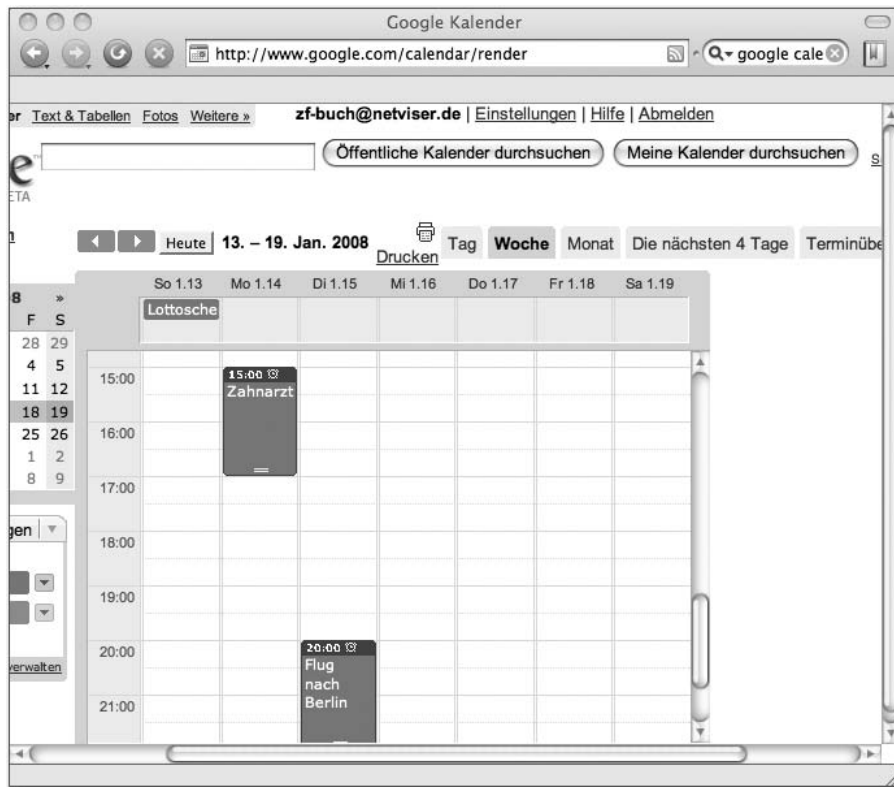
Da die Uhrzeiten in einem ISO 8601-kompatiblen Format vorliegen, habe ich hier der Einfachheit halber zur Klasse `Zend_Date` gegriffen, um sie zu konvertieren. Die Ausgabe der Kalenderdaten sehen Sie in Abbildung 5.13.



**Abbildung 5.13** Ausgabe der Kalenderdaten

Wie Sie sehen, ist der Zugriff auf die Daten nicht sonderlich kompliziert. Die einzelnen Elemente der Einträge liegen in Form von Eigenschaften bzw. Objekten vor. Einige der Eigenschaften sind allerdings etwas komplexer als andere. So enthält das Element `when` beispielsweise zwei Informationen in Form von Attributen: Die Start- und die Endzeit des Termins. Diese komplexeren Elemente sind innerhalb von `Zend_Gdata` als eigene Klassen umgesetzt. Klassen, die für alle Dienste relevant sind, finden Sie im Unterverzeichnis `Zend/Gdata/Extension`. Klassen, die nur für bestimmte Dienste genutzt werden, finden Sie im jeweiligen Unterverzeichnis des Dienstes. Das heißt, die Klassen für Calendar finden Sie

unter *Zend/Gdata/Calendar/Extension*. Die Tatsache, dass es sich dabei um eigene Klassen handelt, ist eigentlich nicht weiter wichtig. Ich wollte es nur erwähnen haben, damit Sie in den Klassen ein wenig stöbern, um noch weitere Funktionalitäten zu entdecken.



**Abbildung 5.14** Der Kalender bei Google

Ein wichtiger Punkt sind gelöschte Termine. Diese finden im letzten Beispiel noch keine Beachtung. Termine, die Sie gelöscht haben, werden nicht sofort gelöscht. Sie werden zunächst nur als gelöscht markiert. Auch wenn »gelöschte« Termine bei normalen Abfragen nicht ausgegeben werden, so kann es bei bestimmten Abfragen doch passieren, dass sie wieder auftauchen. Möchten Sie sicherstellen, dass solche Termine nicht ausgegeben werden, müssen Sie zusätzlich die Eigenschaft `eventStatus` abfragen. Ist hier der Wert `http://schemas.google.com/g/2005#event.cancelled` enthalten, ist der Termin gelöscht. Ein normaler Termin enthält den Wert `http://schemas.google.com/g/2005#event.confirmed`.

Nach diesem Einstieg möchte ich einen kleinen Schritt zurückgehen. Bei dem vorhergehenden Beispiel habe ich einfach nur einen Kalender ausgelesen, habe dabei aber nicht beachtet, welcher Kalender das ist.

### Auslesen verschiedener Kalender

Haben Sie einen Account bei Google Calendar, so können Sie mehrere Kalender anlegen. Somit sind Sie in der Lage, private und geschäftliche Termine zu trennen oder beispielsweise einen eigenen Kalender für Geburtstage zu führen. Fragt man Google Calendar so ab, wie oben beschrieben, erhalten Sie den »Standard-Kalender«, den Google per Default für Sie anlegt. Darüber hinaus legt Google standardmäßig aber noch einen Kalender für Geburtstage an. Um diesen abzufragen, benötigen Sie eine andere URL für den Feed. Diese URL können Sie, sobald Sie die entsprechenden Informationen haben, selbst konstruieren.

Dazu müssen Sie zunächst die Informationen zu den Kalendern abfragen. Hierfür ist die Methode `getCalendarListFeed()` vorgesehen. Sie liefert Ihnen ein Objekt, in dem die Informationen zu den einzelnen Feeds enthalten sind. Ein Eintrag in diesem Feed hat einen Aufbau wie diesen:

```
<entry>
<id>http://www.google.com/calendar/feeds/default/allcalendars/full/
user%40gmail.com</id>
  <published>2007-07-11T22:10:30.257Z</published>
  <updated>2007-07-11T21:46:35.000Z</updated>
  <title type="text">My Primary Calendar</title>
  <summary type="text">A primary calendar ....</summary>
  <author>
    <name>Coach</name>
    <email>user@gmail.com</email>
  </author>
  <gCal:timezone value="America/Los_Angeles"/>
  <gCal:hidden value="false"/>
  <gCal:color value="#2952A3"/>
  <gCal:selected value="true"/>
  <gCal:accesslevel value="owner"/>
  <gd:where valueString="Mountain View"/>
</entry>
```

Interessant dabei ist das Element `id`. Es enthält im letzten Teil eine eindeutige Kennung für den entsprechenden Kalender. Mit dieser Kennung, die im Google-Manual auch als »UserID« bezeichnet wird, kann man die URL des entsprechenden Kalender-Feeds erstellen. Bei dem Hauptkalender entspricht diese ID der E-Mail-Adresse, mit der Sie sich anmelden. Leider ist zurzeit anscheinend noch

keine Methode in `Zend_Gdata` vorhanden, um diese ID entsprechend aufzubereiten, sodass dies manuell geschehen muss.

Nachdem man die ID extrahiert hat, könnte man die URL manuell erstellen. Glücklicherweise ist das aber nicht nötig. Die Klasse `Zend_Gdata_Calender_EventQuery` hilft Ihnen dabei. Eine solche Query-Klasse steht allen GData-Anwendungen zur Verfügung. Die Klasse ist ein Kind der Klasse `Zend_Gdata_Query`, welche die Funktionalitäten definiert, die in allen Klassen benötigt werden. Ein Objekt der Klasse `Zend_Gdata_Calender_EventQuery` erhalten Sie, wenn Sie aus dem Kalender-Objekt heraus die Methode `newEventQuery()` aufrufen.

Damit die Methode die korrekte URL für den Feed konstruieren kann, benötigt sie mindestens die User-ID, die Visibility und den Projection-Level. Die User-ID ist derjenige Teil, der aus der oben erwähnten ID extrahiert wird. Die Visibility, also die Sichtbarkeit, bezeichnet, ob Sie nur die Einträge erhalten wollen, die für alle sichtbar sind, oder ob Sie die Einträge haben wollen, die für Sie als Eigentümer des Kalenders sichtbar sind. Im ersten Fall würden Sie der Methode `setVisibility()` 'public' übergeben und im zweiten Fall, der der übliche sein dürfte, dagegen 'private'. Der Projection-Level legt fest, wie viele Daten Sie von den Einträgen auslesen wollen. Hier dürfte es üblich sein, der Methode `setProjection()` den Wert 'full' zu übergeben, um alle Daten zu erhalten. Möglich wäre allerdings auch der Wert 'basic'. Nachdem Sie das Objekt entsprechend vorbereitet haben, können Sie auch schon den Kalender abfragen, indem Sie das Objekt als Parameter an die Methode `getCalendarEventFeed()` übergeben. Das fertige Listing zum Abfragen aller Kalender könnte beispielsweise so aussehen:

```
require_once 'Zend/Gdata/ClientLogin.php';
require_once 'Zend/Gdata/Calendar.php';
require_once 'Zend/Gdata/Calendar/EventQuery.php';
require_once 'Zend/Date.php';

header ('Content-Type: text/html; charset=utf-8');

$email = 'zf-buch@netviser.de';
$password = 'totalgeheim';
$service = 'cl';
try
{
    $client = Zend_Gdata_ClientLogin::getHttpClient($email, $password, $service);
}
catch (Exception $e)
{
```

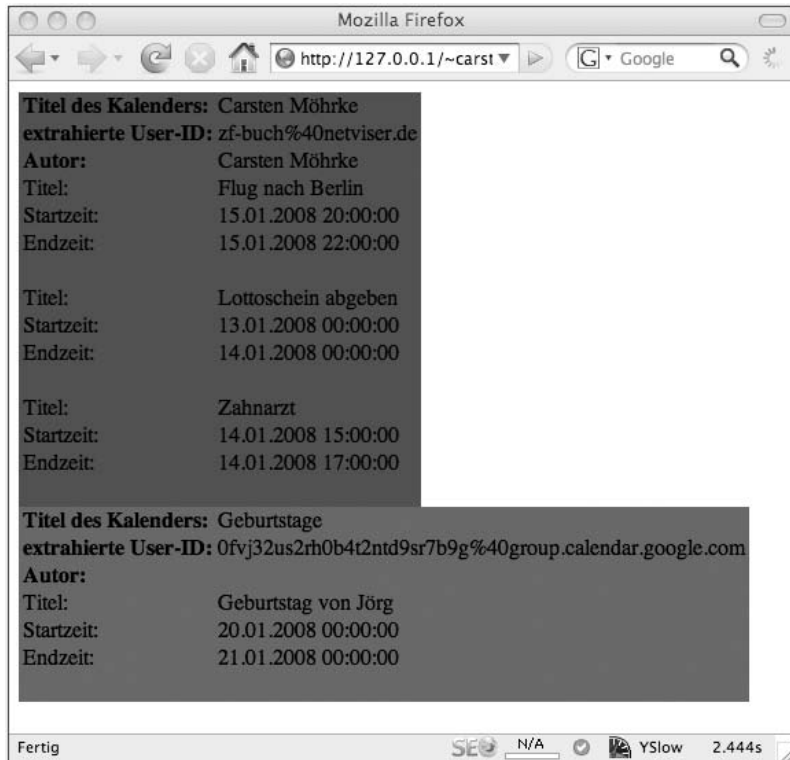


```

    }
    echo '</table>';
}

```

**Listing 5.14** Ausgabe aller Kalender eines Benutzers



**Abbildung 5.15** Ausgabe aller Kalenderdaten

Wie Sie in Abbildung 5.15 sehen, habe ich die Ausgabe der einzelnen Termine ein wenig gekürzt. Die Kalender sind in diesem Fall farblich hinterlegt. Dabei handelt es sich um dieselben Farben, die auch bei der Darstellung bei Google genutzt werden. Die Tatsache, dass bei dem Kalender »Geburtstage« kein Autor angegeben ist, liegt daran, dass ich ihn bei Google nicht eingepflegt habe. Bei dem Termin in der Geburtstagsliste handelt es sich um ein ganztägiges Ereignis, genau wie bei dem Termin »Lottoschein abgeben« im Hauptkalender. Daraus resultieren die Start- und die Endzeit.

### Nutzung des Event-Query-Objekts

Das `EventQuery`-Objekt kann aber noch mehr. Die Methoden, die Sie bisher kennengelernt haben, sind spezifisch für die Kalenderabfragen. Allerdings sind im `Query`-Objekt einige Abfragen enthalten, die in allen `GData`-Bereichen zur Verfügung stehen. Das ist beispielsweise die Möglichkeit, mit `setQuery()` eine Abfrage, sprich eine Volltextsuche, zu definieren. Das heißt, wenn die Zeile

```
$query->setQuery('Zahnarzt');
```

ergänzt wird, wird nur der eine Termin gefunden, in dem das Wort »Zahnarzt« vorkommt. Dabei handelt es sich übrigens nicht um eine Substring-Suche. Das heißt, eine Suche nach »Zahn« würde keinen Termin zurückliefern.

Eine weitere Möglichkeit, die alle `Query`-Objekte unterstützen, ist die maximale Anzahl der Treffer, die Sie mit der Abfrage erhalten, einzuschränken. Das können Sie mit `setMaxResults()` machen. Damit Sie durch die Gesamtzahl der Treffer blättern können, ist es natürlich auch möglich festzulegen, mit welchem Treffer Sie beginnen wollen. Dafür ist die Methode `setStartIndex()` vorgesehen.

Hilfreich kann es auch noch sein, das Veröffentlichungsdatum bzw. das Aktualisierungsdatum einzuschränken. Dafür sind die Methoden `setPublishedMin()`, `setPublishedMax()`, `setUpdatedMin()` und `setUpdatedMax()` vorgesehen. Sie bekommen einen Timestamp im RFC 3339-Format übergeben und schränken die Ergebnismenge entsprechend ein. Das RFC 3339-Format ist weitestgehend kompatibel mit ISO 8601, sodass Sie auf die entsprechenden Funktionalitäten von `Zend_Date` zurückgreifen können.

Nur für die Kalenderabfragen gilt, dass Sie die Einträge sortieren können. Standardmäßig werden die Einträge nach dem Datum der letzten Veränderung sortiert. Mit der Methode `setOrderBy()` können Sie festlegen, ob nach Startzeit des Termins `'starttime'` oder nach dem Datum der letzten Änderung `'lastmodified'` sortiert werden soll. Die Richtung, in der die Daten sortiert werden sollen, also absteigend oder aufsteigend, können Sie mit der Methode `setSortOrder()` festlegen. Dieser übergeben Sie entweder `'ascending'` oder `'a'` für eine aufsteigende Sortierung bzw. `'descending'` oder `'d'` für eine absteigende Sortierung. Darüber hinaus sind noch weitere Funktionalitäten definiert, die Sie bitte der Dokumentation der Methode `Zend_Gdata_Calendar_EventQuery` entnehmen.

### Neue Einträge anlegen

Nachdem Sie nun wissen, wie Sie Einträge auslesen, stellt sich die Frage, wie neue Einträge im Kalender angelegt werden. Auch das ist recht einfach zu erledigen. Zunächst benötigen Sie ein `EventEntry`-Objekt. Dieses muss dann nur mit

den entsprechenden Informationen bestückt werden und kann dann zu Google übertragen werden.

Die wichtigsten Dinge, die bei einem Termin vorhanden sein sollten, sind natürlich der Titel des Termins, der Ort sowie die Uhrzeit. Eine kleine Beschreibung ist sicher auch noch hilfreich. Die entsprechenden Eigenschaften kennen Sie ja schon vom Auslesen der Daten. Das heißt, die Eigenschaft `title`, die ausgelesen wurde, muss hier wieder mit einem Wert, genauer gesagt einem Objekt belegt werden. In diesem Fall handelt es sich um ein Objekt der Klasse `Zend_Gdata_App_Extension_Title`, das durch eine magische Factory namens `newTitle()` angelegt wird. Gleiches gilt für die Beschreibung, die der Eigenschaft `content` zugewiesen wird, wobei das dazugehörige Objekt durch die Methode `newContent()` erstellt wird.

Gleiches gilt grundsätzlich auch für die Eigenschaften `where` und `when`, welche die Zeit und den Ort enthalten. Allerdings ist hier zu beachten, dass die Eigenschaften mit Arrays bestückt werden müssen. Die Objekte, die mit `newWhen()` und `newWhere()` abgeleitet werden, müssen somit in Form eines indizierten Array an die Eigenschaft übergeben werden.

Ein komplettes Beispiel, um einen neuen Termin anzulegen, finden Sie in Listing 5.15:

```
$cal = new Zend_Gdata_Calendar($client);

// Neues Event-Objekt ableiten
$event= $cal->newEventEntry();

// Titel des Termins anlegen
$text = utf8_encode('Essen im Möpken');
$title = $cal->newTitle($text);
$event->title = $title;

// Beschreibung festlegen
$text = utf8_encode('Den guten Anzug anziehen!');
$content = $cal->newContent($text);
$event->content = $content;

// Ort festlegen
$text = utf8_encode('Schloß Neuhaus');
$where = $cal->newWhere($text);
$event->where = array($where);

// Zeiten festlegen
$when = $cal->newWhen();
```

```
// Zeiten als Zend_Date-Objekte anlegen
$start = new Zend_Date('18.01.2008 20:00:00','de_DE');
$ende = new Zend_Date('18.01.2008 22:00:00','de_DE');
// Nach ISO konvertieren und den Eigenschaften zuweisen
$when->startTime = $start->get(Zend_Date::ISO_8601);
$when->endTime = $ende->get(Zend_Date::ISO_8601);
$event->when = array($when);

// Termin speichern
$cal->insertEvent($event);
```

**Listing 5.15** Anlegen eines neuen Termins

Ich denke, das Listing ist recht einfach zu verstehen. Ein paar Kleinigkeiten möchte ich aber dennoch erwähnen. Zu Beginn sollten Sie immer darauf achten, dass die Daten UTF-8-codiert sind. Nutzen Sie Texte, die nicht UTF-8-codiert sind, macht sich das nicht sofort bemerkbar. Die Klasse wirft keine Exception; Google verwirft die Daten lediglich. Das heißt, es könnte unter Umständen ein Termin angelegt werden, der leer wäre.

Der zweite Punkt, auf den ich hinweisen möchte, ist, dass die Namen der Factory-Methoden immer den Namen der Eigenschaften mit einem vorangestellten `new` entsprechen. Wenn Sie dies stets bedenken, haben Sie eine gute Chance, bei der Vielzahl der Methoden den Überblick zu behalten.

Der dritte und letzte Punkt ist, dass Sie die Texte, die den Eigenschaften zugewiesen werden sollen, direkt an den Konstruktor übergeben sollten. Zwar gibt es auch Methoden, mit denen Sie die Texte setzen können, aber das führt schnell dazu, dass Sie die falsche Methode nutzen. Wollen Sie die Texte über Methoden festlegen, setzt das eine recht gute Kenntnis der Google-API voraus.

Wie der Termin im Google-Kalender dargestellt wird, sehen Sie in Abbildung 5.16. Auf diese Art und Weise wird der Termin im Standard-Kalender angelegt. Möchten Sie einen anderen Kalender ansprechen, kommt das `EvenQuery`-Objekt wieder ins Spiel, das Sie ja schon bei den Abfragen kennengelernt haben. Bei den Abfragen wurde das Objekt komplett übergeben. Im Hintergrund wurde aber im Endeffekt nur die URL ausgelesen, die mithilfe des Objekts konstruiert wurde. Auch beim Speichern eines Termins muss die URL des entsprechenden Kalender-Feeds konstruiert werden. Die Vorgehensweise ist dabei dieselbe, nur muss hier die URL manuell mit der Methode `getQueryUrl()` ausgelesen und dann als zweiter Parameter an die Methode `insertEvent()` übergeben werden.



Abbildung 5.16 Der neu angelegte Termin bei Google

Da in diesem Beispiel nur der Geburtstagskalender als weiterer Kalender zur Verfügung steht, sollte der Termin auch ganztägig angelegt werden. Um das zu erreichen, übergeben Sie die Daten von zwei aufeinanderfolgenden Tagen, wobei Sie keine Uhrzeit angeben. Einen neuen Eintrag im Geburtstagskalender anzulegen, könnte also so aussehen:

```
// Neues Event-Objekt ableiten
$event= $cal->newEventEntry();

// Titel des Termins anlegen
$text = utf8_encode('Geburtstag von Anika');
$title = $cal->newTitle($text);
$event->title = $title;

// Zeiten für ganztägigen Termin festlegen
$when = $cal->newWhen();
$when->startTime = '2008-03-25';
$when->endTime = '2008-03-26';
$event->when = array($when);

// Query-Objekt ableiten
$query = $cal->newEventQuery();
// User-ID des Geburtstagskalenders
```

```

$query->setUser(
    '0fvj32us2rh0b4t2ntd9sr7b9g%40group.calendar.google.com');
$query->setVisibility('private');
$query->setProjection('full');
// URL auslesen
$url = $query->getQueryUrl();
// Termin speichern
$cal->insertEvent($event, $url);

```

**Listing 5.16** Anlegen eines Geburtstags

Geburtstage und einige andere Termine haben eine Eigenschaft, die sie von normalen Terminen unterscheidet. Und zwar wiederholen sie sich jährlich. Sie können auch solche Termine anlegen. Die Vorgehensweise ist dabei allerdings ein wenig anders. Und zwar übergeben Sie die Zeiten nicht in Form eines `when`-Objekts, sondern in Form eines `recurrence`-Objekts. Leider gibt es hier noch keine schönen Zugriffsmethoden. Daher müssen Sie die Regeln für die Wiederholung des Termins von Hand erstellen. Dabei geben Sie den Beginn des Termins, das Ende des Termins und das Wiederholungsmuster an. Der Aufbau dieses Regelwerks ist in RFC 2445 (<http://www.ietf.org/rfc/rfc2445.txt>) definiert. Um den Termin, wie er oben angelegt wurde, jährlich zu wiederholen, würde im obigen Code die Festlegung der `when`-Bedingung entfernt. Die Zeilen würden durch die folgenden ersetzt:

```

$recurrence = "DTSTART;VALUE=DATE:20070325\r\n" .
    "DTEND;VALUE=DATE:20070326\r\n" .
    "RRULE:FREQ=YEARLY\r\n";
$event->recurrence = $cal->newRecurrence($recurrence);

```

Hierbei definiert `DTSTART` den Starttermin und `DTEND` den Endtermin des ersten Termins. Da es sich um ein ganztägiges Ereignis handelt, wird keine Zeit angegeben. Mit `RRULE` wird die Recurrence-Rule, also diejenige Regel definiert, nach welcher sich der Termin wiederholt. `FREQ=YEARLY` definiert eine jährliche Wiederholung. Sie könnten hier auch eine monatliche Wiederholung oder eine Wiederholung zu einem bestimmten Wochentag angeben. Weitere Informationen entnehmen Sie bitte dem RFC.

### Verändern von Kalendereinträgen

Natürlich haben Sie auch die Möglichkeit, einen Eintrag zu verändern. Auch das ist recht einfach zu bewerkstelligen. Die Vorgehensweise dabei ist, dass Sie das Objekt des entsprechenden Eintrags auslesen, die neuen Werte darin speichern und das Ganze dann wieder zurück auf den Server speichern.

So weit, so gut. Allerdings wurde in den vorhergehenden Beispielen immer ein kompletter Kalender ausgelesen. Natürlich könnte man jetzt auf die Idee kommen mitzuzählen, welches Objekt geändert werden soll, aber das wäre recht umständlich. Sie können aber auch einzelne Einträge gezielt auslesen. Jeder Eintrag hat, wie Sie das auch schon von den Kalender-Feeds kennen, eine eigene URL, unter der er angesprochen werden kann. Diese URL können Sie über die Eigenschaft `id` auslesen. Glücklicherweise müssen Sie diese URL nicht weiter zerlegen, sondern können sie direkt verwenden. Eine solche URL können Sie an die Methode `getCalendarEventEntry()` übergeben, welche dann den einzelnen Eintrag ausliest und Ihnen ein `Event`-Objekt zurückgibt. Dieses können Sie anschließend verändern und mithilfe der Methode `save()`, welche Sie aus dem `Event`-Objekt heraus aufrufen, wieder abspeichern:

```
// Verbindungsaufbau etc.
$cal = new Zend_Gdata_Calendar($client);
// URL eines Termins der aus der Eigenschaft id ausgelesen wurde
$eventURL = "http://www.google.com/calendar/feeds/zf-
buch%40netviser.de/private/full/uitntmn0q8ea1n930bpc93r1cg";
/// Einzelnes Event auslesen
$event = $cal->getCalendarEventEntry($eventURL);
// Neuen Titel setzen
$text = utf8_encode('Flug nach Hamburg, nicht nach München');
$event->title = $cal->newTitle($text);
// Aktualisiertes Event wieder speichern
$event->save();
```

**Listing 5.17** Ändern eines Eintrags

Sie können sonst also genau so vorgehen, als würden Sie den Termin neu anlegen.

### Löschen von Kalendereinträgen

Das Löschen eines Eintrags im Kalender ist dem Editieren recht ähnlich. Auch in diesem Fall sollten Sie zunächst das entsprechende Eintrags-Objekt auslesen. Danach können Sie die Methode `delete()` aufrufen, die den Eintrag löscht.

```
$cal = new Zend_Gdata_Calendar($client);
$eventURL = "http://www.google.com/calendar/feeds/zf-
buch%40netviser.de/private/full/uitntmn0q8ea1n930bpc93r1cg";
$event = $cal->getCalendarEventEntry($eventURL);
$event->delete();
```

**Listing 5.18** Löschen eines Eintrags

### 5.5.4 Nutzung von Google Spreadsheets

Unter der URL <http://docs.google.com> bietet Google die Möglichkeit, eine Textverarbeitung, ein Präsentationsprogramm sowie eine Tabellenkalkulation zu nutzen. Sollten Sie keine sonderlich anspruchsvollen Büroaufgaben mit einem Office-Paket erledigen müssen, dann ist die Nutzung der Google-Angebote eine echte Alternative. Zwar sind die Möglichkeiten nicht ganz so umfangreich wie bei einem echten Office-Paket, aber der Funktionsumfang ist schon recht beeindruckend.

Das Zend Framework unterstützt zurzeit nur den Zugriff auf Spreadsheets, also die Tabellenkalkulation. Innerhalb der Tabellenkalkulation können Sie verschiedene Arbeitsmappen (die Spreadsheets) verwalten. Ein Spreadsheet beinhaltet jeweils einzelne Tabellenblätter, die Worksheets. Innerhalb eines Worksheets sind schließlich die einzelnen Felder mit den Daten zu finden.

Auch hier möchte ich Ihnen empfehlen, einen Blick in die Google-Dokumentation zur Spreadsheets-API zu werfen, die Sie unter der folgenden Adresse finden: [http://code.google.com/apis/spreadsheets/developers\\_guide\\_protocol.html](http://code.google.com/apis/spreadsheets/developers_guide_protocol.html)

Der Zugriff auf die Daten erfolgt wieder über Streams, wie Sie es schon kennengelernt haben. Das Anlegen neuer Spreadsheets ist zurzeit noch nicht möglich.

#### Auslesen von Spreadsheets

Bevor Sie ein Spreadsheet auslesen können, ist es hilfreich zu wissen, welche Spreadsheets überhaupt zur Verfügung stehen. Die Vorgehensweise ist hier der bei dem Auslesen der Kalender sehr ähnlich. Sie können einen Feed mit allen Spreadsheet-Einträgen auslesen, indem Sie die Methode `getSpreadsheetFeed()` aus einem `Zend_Gdata_Spreadsheets`-Objekt heraus aufrufen. Das `Zend_Gdata_Spreadsheets`-Objekt wird genau so abgeleitet wie das entsprechende Kalender-Objekt. Allerdings müssen Sie beim Ableiten des HTTP-Clients die Kennung `c1` durch `wise` ersetzen.

Die Methode liefert Ihnen ein Objekt der Klasse `Zend_Gdata_Spreadsheets_SpreadsheetFeed` zurück, das Sie direkt an eine `foreach`-Schleife übergeben können. Bei jeder Iteration wird dann ein Objekt der Klasse `Zend_Gdata_Spreadsheets_SpreadsheetEntry` ausgelesen. Die dahinterliegende XML-Struktur finden Sie in der API-Dokumentation von Google. Die wichtigsten Eigenschaften werden auch in Listing 5.19 verwendet:

```
require_once 'Zend/Gdata/ClientLogin.php';
require_once 'Zend/Gdata/Spreadsheets.php';
require_once 'Zend/Date.php';
```

```

header ('Content-Type: text/html; charset=utf-8');

$email = 'zf-buch@netviser.de';
$password = 'totalgeheim';
$service = 'wise';
try
{
    $client = Zend_Gdata_ClientLogin::getHttpClient(
        $email, $password, $service);
}
catch (Exception $e)
{
    die ('Folgender Fehler trat auf: ' . $e->getMessage());
}

// Spreadsheets Client ableiten
$spread = new Zend_Gdata_Spreadsheets($client);
// Feed auslesen
$spreadsheets = $spread->getSpreadsheetFeed();
echo '<table>';
foreach ($spreadsheets as $spreadsheet)
{
    echo '<tr>';
    echo '<td>Titel</td>';
    echo '<td>'.$spreadsheet->title.'</td>';
    echo '</tr>';
    echo '<tr>';
    echo '<td>Autor</td>';
    echo '<td>'.$spreadsheet->author[0]->name.'</td>';
    echo '</tr>';
    echo '<tr>';
    echo '<td>Geändert</td>';
    // Datum mit Zend_Date konvertieren
    $datum = new Zend_Date($spreadsheet->updated,
        Zend_Date::ISO_8601, 'de_DE');
    echo '<td>'.$datum.'</td>';
    echo '</tr>';
    echo '<tr><td colspan="2">&nbsp;</td></tr>';
}
echo '</table>';

```

**Listing 5.19** Auslesen der verfügbaren Spreadsheets

Auch hier hat Google das Prinzip der IDs verfolgt. Das heißt, jedes Spreadsheet hat eine eigene ID. Dabei handelt es sich um den letzten Teil der Eigenschaft `id`, die in jedem Spreadsheet-Objekt enthalten ist. Mit dieser ID ist es dann möglich, auf die einzelnen Arbeitsblätter (Worksheets) zuzugreifen. Auch diese können in Form eines Feeds ausgelesen werden, wobei auch jedes einzelne Tabellenblatt über eine Eigenschaft `id` verfügt, von der der letzte Teil eine eindeutige Kennung innerhalb der Arbeitsmappe darstellt. Mit dieser `id` ist es dann wiederum möglich, die Feeds abzufragen, welche die Tabellenfelder beinhalten. Ein solcher Feed beinhaltet für jede Zeile ein eigenes Objekt der Klasse `Zend_Gdata_Spreadsheets_ListEntry`. Auch wenn das bereits eine Zeile darstellt, müssen Sie auf dieses Objekt zunächst die Methode `getCustom()` anwenden. Sie liefert Ihnen ein Array zurück, das für jedes Feld in der Zeile ein Objekt der Klasse `Zend_Gdata_Spreadsheets_Extension_Custom` enthält. Diese Vorgehensweise mag auf den ersten Blick ein wenig komplex erscheinen, ist aber sehr klar strukturiert.

In dem folgenden Beispiel wird ein Spreadsheet ausgelesen, das nur ein Worksheet beinhaltet. Die ID des Spreadsheets wurde vorher ermittelt. Das Tabellenblatt, das ausgelesen werden soll, sehen Sie in Abbildung 5.17.

	A	B	C	D
1	Produkt	Einkaufspreis	Verkaufspreis	
2	Hose	54,20 €	99,90 €	
3	Schuhe	35,98 €	69,00 €	
4	Jacke	31,00 €	68,99 €	
5				
6				

Abbildung 5.17 Tabellenblatt bei Google Spreadsheets

Der Code, der die Daten ausliest und darstellt, sieht folgendermaßen aus:

```
$spread = new Zend_Gdata_Spreadsheets($client);
// ID eines Spreadsheets
$spreadsheetId='http://spreadsheets.google.com/feeds/spreadsheets/
```

```

o17931546445181751748.995413063132391215';
// Key extrahieren
$spreadsheetKey = substr(strrchr($spreadsheetId, '/'), 1 );
// Neues Query-Objekt erstellen und mit Key initialisieren
$query = new Zend_Gdata_Spreadsheets_DocumentQuery();
$query->setSpreadsheetKey($spreadsheetKey);

// Feed mit Tabellenblättern auslesen
$worksheets = $spread->getWorksheetFeed($query);
// es ist nur ein Blatt => Kann direkt übernommen werden
$worksheet = $worksheets->current();
// ID extrahieren
$worksheetId = $worksheet->id;
$worksheetKey = substr(strrchr($worksheetId, '/'), 1 );

//Neue Abfrage erstellen und mit Key initialisieren
$query = new Zend_Gdata_Spreadsheets_ListQuery();
$query->setSpreadsheetKey($spreadsheetKey);
$query->setWorksheetId($worksheetKey);
// Daten auslesen
$listFeed = $spread->getListFeed($query);

// Titel des Blattes ausgeben
echo "Inhalt des Blattes <b>". $worksheet->title. "</b><br>";
echo "<table border='1'>";
// Daten der ersten Zeile ausgeben
$ersteZeile = $listFeed->current()->getCustom();
// Spaltenüberschriften ausgeben
echo '<tr>';
foreach ($ersteZeile as $entry)
{
    echo '<td>'. $entry->columnName. '</td>';
}
echo '</tr>';
// Werte zeilenweise ausgeben
foreach ($listFeed as $entry)
{
    echo '<tr>';
    $custom = $entry->getCustom();
    // Einzelnes Feld ausgeben
    foreach ($custom as $feld)
    {
        echo '<td>';
        echo $feld->text;
        echo '</td>';
    }
}

```

```

    }
    echo '</tr>';
}
echo '</table>';

```

**Listing 5.20** Abfrage der Daten aus einem Tabellenblatt

Wie Sie sehen, werden hier zwei unterschiedliche Query-Klassen genutzt. Das ist zum einen `Zend_Gdata_Spreadsheets_DocumentQuery`, um eine Abfrage auf Dokumentenebene durchzuführen, sprich das Tabellenblatt zu finden. Zum anderen handelt es sich um die Klasse `Zend_Gdata_Spreadsheets_ListQuery`, mit der auf die Zelleninhalte zugegriffen werden kann. Ein wenig verwirrend ist, dass die eindeutige Kennung für die Dokumente als `Key` bezeichnet wird, wohingegen die Kennung für die Blätter eine `ID` ist, wie Sie an den beiden folgenden Zeilen sehen können, die die zweite Abfrage initialisieren:

```

$query->setSpreadsheetKey($spreadsheetKey);
$query->setWorksheetId($worksheetKey);

```

Die meisten anderen Punkte dürften klar sein. Ich möchte aber noch auf die Ausgabe der Spaltenüberschriften eingehen, wofür diese Zeilen zuständig sind:

```

$ersteZeile = $listFeed->current()->getCustom();
// Spaltenüberschriften ausgeben
echo '<tr>';
foreach ($ersteZeile as $entry)
{
    echo '<td>'.$entry->columnName.'</td>';
}
echo '</tr>';

```

Nach der Abfrage enthält `$listFeed` für jede Zeile ein Objekt. Das erste wird mit der Methode `current()` ausgelesen, und auf das Ergebnis wird sofort `getCustom()` angewandt. Dadurch erhalten Sie an der Stelle ein Array mit den Daten der ersten Zeile. Erste Zeile meint bei Google Spreadsheets aber nicht die Zeile, in der Sie die Worte »Produkt«, »Einkaufspreis« und »Verkaufspreis« sehen. Die erste Zeile ist vielmehr die darunterliegende Zeile. Die Spaltenüberschriften können leider nicht so einfach und direkt ausgelesen werden. Dafür ist die Überschrift jeder Spalte aber in jedem Zellen-Objekt in der Eigenschaft `columnName` enthalten, was ich mir hier zunutze mache. Daher wird die erste Zeile einmal vorweg ausgelesen, und die Eigenschaft `columnName` wird ausgegeben.

Die Ausgabe des Scripts sehen Sie in Abbildung 5.18.

produkt	einkaufspreis	verkaufspreis
Hose	54,20 €	99,90 €
Schuhe	35,98 €	69 €
Jacke	31 €	68,99 €

Abbildung 5.18 Ausgabe der Tabellendaten

Wäre in einer der Zellen eine Formel enthalten gewesen, hätte das System nur das Ergebnis der Berechnung zurückgegeben.

Wie am Anfang des Kapitels bereits erwähnt, können Sie auch hier nach einzelnen Zellinhalten suchen. Wäre bei der zweiten Abfrage `setQuery('Hose')` ergänzt worden, hätte die Abfrage nur diejenigen Zeilen geliefert, in denen das Wort »Hose« vorkommt. Gerade bei einer Tabellenkalkulation wäre es natürlich langweilig, wenn man lediglich eine Textsuche durchführen könnte. Daher kennt die Spreadsheet-API auch die Möglichkeit, eine »Structured Query« auszuführen. Dabei haben Sie deutlich mehr Möglichkeiten. Wollten Sie beispielsweise alle Zeilen auslesen, bei denen der Einkaufspreis kleiner als 50 ist, so würde das Initialisieren des Query-Objekts so aussehen:

```
$query = new Zend_Gdata_Spreadsheets_ListQuery();
$query->setSpreadsheetKey($spreadsheetKey);
$query->setWorksheetId($worksheetKey);
$query->setSpreadsheetQuery('einkaufspreis < 50');
```

Listing 5.21 Nutzung einer Structured Query

Die Methode `setSpreadsheetQuery()` ist also dafür zuständig, eine Structured Query zu generieren. Hier können Sie auch die Operatoren `!=` und `==` sowie die Verknüpfungen `AND` und `OR` nutzen, wobei Sie Teilausdrücke auch klammern dürfen. Eine solche Abfrage bezieht sich immer auf eine ganze Zeile und nicht nur auf ein Feld. Das heißt, mit der Abfrage

```
$query->setSpreadsheetQuery('einkaufspreis < 50 AND
                             verkaufspreis < 69');
```

erhalten Sie lediglich die Zeile mit der Jacke. Bitte beachten Sie dabei, dass die Spaltenüberschriften, die hier als Referenz genutzt werden, zwingend kleingeschrieben werden müssen, weil alles andere in einer Exception resultiert.

### Einfügen von neuen Zeilen

Sie können auch jederzeit eine neue Zeile in eine Tabelle einfügen. Dazu benötigen Sie nur die IDs bzw. die Keys der Arbeitsmappe und des Tabellenblatts. Sobald Sie diese Informationen haben, können Sie direkt aus dem `Zend_Gdata_Spreadsheets`-Objekt heraus die Methode `insertRow()` aufrufen. Sie erhält als ersten Parameter die Daten für die neue Zeile in Form eines Arrays übergeben. Die beiden IDs folgen danach als Parameter.

Das Array muss so aufgebaut sein, dass der Spaltenname jeweils als Schlüssel genutzt wird und dann auf den Wert verweist, der in der Zeile eingefügt werden soll. Von daher sollten Sie jeder Spalte einen eindeutigen Namen geben. Wenn Sie dies nicht tun, vergibt das System automatisch einen Namen für die Spalte. Diesen können Sie wie oben bereits beschrieben auslesen.

Daten in eine Spalte zu schreiben, die noch nicht initialisiert wurde, ist zurzeit noch nicht möglich und resultiert in einer Exception.

Das Einfügen einer Zeile könnte so aussehen:

```
$spread = new Zend_Gdata_Spreadsheets($client);
// ID eines Spreadsheets
$spreadsheetId='http://spreadsheets.google.com/feeds/spreadsheets/
o17931546445181751748.995413063132391215';
// Key extrahieren
$spreadsheetKey = substr(strrchr($spreadsheetId, '/'), 1 );
$worksheetId = 'http://spreadsheets.google.com/feeds/' .
    'worksheets/o17931546445181751748.995413063132391215/' .
    'private/full/od6';
$worksheetKey = substr(strrchr($worksheetId, '/'), 1 );
$zeile = array ('produkt'=>'Socken',
    'einkaufspreis'=>'2', 'verkaufspreis'=>'5');
$spread->insertRow($zeile, $spreadsheetKey, $worksheetKey);
```

**Listing 5.22** Einfügen einer neuen Zeile

### Löschen von Zeilen

Auch das Löschen einer Zeile ist kein Problem. Die Vorgehensweise entspricht der, die Sie schon beim Löschen von Kalendereinträgen kennengelernt haben. Und zwar können Sie aus einem Zeilen-Objekt heraus die Methode `delete()` aufrufen, die die Zeile entfernt. Um möglichst einfach an das entsprechende Zeilen-Objekt zu kommen, nutzen Sie die Methode `getListEntry()`, die direkt aus dem `Spreadsheets`-Objekt heraus aufgerufen wird. Sie bekommt die ID der Zeile (auch in diesem Fall den kompletten Inhalt der Eigenschaft `id`) übergeben und liefert

die entsprechende Zeile als Objekt zurück. Danach können Sie dann `delete()` aufrufen, um die Zeile zu löschen.

```
$zeilenId='http://spreadsheets.google.com/feeds/list/o17931546445181751748.995413063132391215/od6/private/full/ckd7g';
$zeile = $spread->getListEntry($zeilenId);
$zeile->delete();
```

**Listing 5.23** Löschen einer Zeile

### Aktualisieren von Zeilen

Grundsätzlich ist es auch kein Problem, eine Zeile zu aktualisieren. Die Vorgehensweise ist ähnlich wie beim Löschen. Zunächst lesen Sie das entsprechende Entry-Objekt aus. Dieses übergeben Sie dann zusammen mit einem Array mit den neuen Daten an die Methode `updateRow()`. Allerdings ist hierbei ein kleiner Fallstrick zu beachten: Die Methode übernimmt nur Daten, die auch in dem Array enthalten sind. Spalten, die nicht genannt werden, verlieren ihre Werte. Wenn Sie also nur einen Wert aktualisieren wollen, so sollten Sie vorher das Array mit den alten Werten belegen, damit diese nicht verlorengehen.

Das könnte dann beispielsweise so aussehen:

```
$zeilenId='.... schrecklich lange ID ....';
$entry = $spread->getListEntry($zeilenId);
$zeile = $entry->getCustom();
$daten = array();
// Alte Daten auslesen
foreach ($zeile as $zelle)
{
    $daten[$zelle->columnName] = $zelle->text;
}
// Neue Information einfügen
$daten['einkaufspreis'] = '1000';
// Daten speichern
$spread->updateRow($entry, $daten);
```

**Listing 5.24** Aktualisieren einer Zeile

# Index

## A

---

Access Control List 115  
Account-Authentication 250  
ACL 115  
Action Controller 30  
Active Recordset 88  
AJAX 306  
Amazon 220  
APC 137, 150  
ASIN 225, 227  
Assertion 121  
Atom 209, 214  
Authentifikation 124  
    *dateibasierend* 128  
    *datenbankbasiert* 124  
AuthSub 250  
AWS 220

## B

---

Basename 168  
Benutzerauthentifizierung 124  
Bildersuche 242  
Bogenmaß 326  
Bootstrap-File 27  
Breitengrad 362  
Bugs 16

## C

---

Cache  
    *Einträge löschen* 153  
    *Garbage Collection* 140  
Caching 137  
    *Dateien* 147  
    *Debugging* 142  
    *Gültigkeitsdauer* 139  
    *komplette Seiten* 141  
    *MVC* 142  
CC-Nummern 157  
Chain 166  
Changelog 16  
Controller 25  
Cookies 346

*auslesen* 350  
*getExpiryTime* 348  
*isExpired* 348  
*isSecure* 348  
*isSession* 348  
*setzen* 350

## D

---

Datenbank 59  
    *DELETE* 77  
    *DISTINCT* 82  
    *GROUP BY* 82  
    *HAVING* 82  
    *INSERT* 76  
    *JOIN* 83  
    *Konsistenz* 103  
    *LIMIT* 81  
    *Performance* 110  
    *Prepared Statements* 66  
    *Profiling* 110  
    *UPDATE* 77  
Datenbankunabhängigkeit 59  
Datum 395  
    *Ausgabe* 397  
    *prüfen* 157, 407  
    *rechnen mit* 402  
    *Vergleich von* 404  
Datum korrigieren 379  
Datumsangaben lokalisieren 378  
Deklarative referentielle Integrität 103  
DIN-Datum 378  
Dokumentation 15  
    *Blogs* 15  
    *Wiki* 15  
DPI 308  
DRI 103

## E

---

E-Mail-Adresse  
    *validieren* 157  
E-Mails 279  
    *abholen* 288  
    *Anhänge* 283

## Index

*Betreff* 280  
*Body* 280  
*CC* 281  
*Content-Disposition* 284  
*Content-Type* 284  
*Dringlichkeit* 282  
*Expunge* 298, 301  
*Header* 281  
*Header auslesen* 292  
*HTML* 282  
*Kopie* 281  
*kopieren* 304  
*löschen* 298  
*Ordner* 302  
*Papierkorb* 302  
*Return-Path* 281  
*SMTP* 286  
*Subject* 280  
*Unique-IDs* 299  
*verschieben* 304  
*verschlüsseln* 288  
*versenden* 280  
Exception Handling 21  
Expunge 298

## F

---

Feeds 209  
  *finden* 209  
  *generieren* 218  
Filtern von Daten 165  
Flickr 232  
Fließen, Maßeinheiten 386  
Fluent Interface 23  
Fluid Interface 23  
Formulare  
  *validieren* 175  
Front Controller 29

## G

---

getPost 39  
Glyph 316  
GMT 396  
Google 245  
  *Calendar* 255  
  *Spreadsheets* 271

## H

---

Helper 49  
hexadezimale Zahlen 160  
htaccess 18  
HTTP 335  
  *Authentifizierung* 341  
  *Cookies* 346  
  *Datei-Download* 345  
  *Server-Antworten* 342  
  *SSL* 352  
  *Uploads* 340  
HTTP-Client 336

## I

---

IIS 28  
Installation 17  
Internationalisierung 369  
IP-Adressen  
  *validieren* 163  
Issue Tracker 16

## J

---

JOIN 83  
JSON 306

## K

---

Kaskadierendes Löschen 109  
Kochen, Maßeinheiten 386  
Konfiguration 194  
Kreditkartennummern 157

## L

---

Längengrad 362  
lastInsertId 75  
Laufweite 316  
Locale 369  
Logfiles 184  
Lokalisierung 369  
  *von Zahlen* 373

## M

---

m:n-Verknüpfung XE 108  
Maildir 288

Mailing-Listen 15  
 Maßeinheiten 385  
     *konvertieren von* 388  
     *rechnen mit* 389  
 mbox 288  
 Mehrsprachigkeit 381  
 Memcached 150  
 MIME-Type 340  
 mod\_rewrite 28  
 Model 26  
 Model View Controller → MVC  
 Module 33  
 MVC 25  
     *\_\_call* 43  
     *Action Controller* 30  
     *assign* 36  
     *Beispiel* 50  
     *Caching* 142  
     *canSendHeaders* 48  
     *Controller* 25  
     *Controller Benennung* 32  
     *Datenübergabe* 36  
     *dispatch* 30  
     *Error Handling* 42  
     *errorAction* 45  
     *ErrorController* 45  
     *escape* 37  
     *Exception Handling* 30  
     *Exceptions* 30  
     *Fortgeschrittene Techniken* 47  
     *forward* 44  
     *Front Controller* 29  
     *getParam* 39  
     *getPost* 39  
     *getRequest* 38  
     *getResponse* 46  
     *getStaticHelper* 49  
     *GET-Werte* 38  
     *Header* 48  
     *init* 47  
     *initView* 36  
     *mehrere Controller* 32  
     *Model* 41  
     *Module* 33  
     *Moved Permanently* 45  
     *Moved Temporarily* 45  
     *Parameter* 39  
     *Plug-ins* 49  
     *postDispatch* 49  
     *POST-Werte* 38

*preDispatch* 49  
     *redirect* 45  
     *Request-Objekt* 38  
     *setControllerDirectory* 30, 31, 33  
     *setParam* 44  
     *throwExceptions* 30, 45  
     *Übergabe von Werten* 38  
     *useDefaultControllerAlways* 42  
     *Verarbeitungsschritte* 37  
     *Verzeichnisstruktur* 27  
     *View* 34  
     *View unterdrücken* 30  
     *ViewRenderer* 48  
 MX-Eintrag 158

## N

---

Natural Key 90  
 Negotiation 370  
 News-Suche 240

## O

---

One-Time-Use-Token 251

## P

---

Papierkorb 302  
 PDF 308  
     *Bilder* 330  
     *Clipping* 331  
     *CMYK* 320  
     *Datei* 312  
     *drawLine* 323  
     *einlesen* 333  
     *Ellipse zeichnen* 327  
     *Farben* 320  
     *Fließtexte* 315  
     *Graustufen* 321  
     *Kreis zeichnen* 326  
     *laden* 333  
     *Meta-Informationen* 331  
     *Polygon zeichnen* 328  
     *Rechteck* 324  
     *RGB* 320  
     *Schriftarten* 313  
     *Schriftschnitt* 314  
     *Seiten hinzufügen* 309  
     *Text ausgeben* 311

## Index

*TrueType-Font* 313

*Zeichnen* 322

PECL 151

Performance 110

php.ini 17

Plug-ins 49

preDispatch 49

## R

---

Realm 128

RealPath 170

Rechteverwaltung 115

referentielle Integrität 103

Ressourcen 116

REST 361

*Client* 362

*Server* 363

RewriteBase 29

Rewrite-Engine 28

Rewrite-Rule 28

RGB 320

Rollback 73

Rollen 116

RSS 209, 211

## S

---

Schaltjahr 407

Sequenzen 60, 75

Session

*beenden* 132

*Save-Handler* 133

*Schutz von Daten* 132

*Time-out* 132

Sessions 129

SMTP after IMAP 287

SMTP after POP 287

SPL 24

Spreadsheets 271

SQLite 152

Standard Programmers Library 24

## T

---

Tabellenkalkulation 271

Tags entfernen 172

toLower 170

Transaktionen 73

## U

---

Übersetzen 371

Unique-IDs 299

URL-Rewriting 28

useDefaultControllerAlways 42

UTC 396

## V

---

Validierung 154

Vererbung von Rechten 118

Verzeichnisstruktur 27

View 25, 34

*Speicherort* 35

*Übergabe von Werten* 36

View-Helper 54

Viskosität, Maßeinheiten 386

## W

---

Währung

*Namen festlegen* 391

Währungsdarstellung 389

Webinare 16

Websuche 236

Whitespaces 171

## X

---

xapi 247

XML-RPC 355

*Client* 359

*Server* 356

## Y

---

Yahoo! 236

Yahoo!-Maps 361

## Z

---

Zeit 395

*Ausgabe* 397

*rechnen mit* 402

*Vergleich von* 404

Zeitangaben lokalisieren 378

Zeitzone 396

- Zenc\_Acl
  - isAllowed* 118
- Zend Platform 152
- Zend\_Acl 115
  - add* 118
  - addRole* 117
  - allow* 118
  - assert* 121
  - bekannte Rechte* 123
  - Bootstrap-File* 123
  - deny* 118
  - getRoleId* 117
  - Manipulieren von Rechten* 123
  - remove* 123
  - removeAll* 123
  - removeAllow* 123
  - removeDeny* 123
  - removeRole* 123
  - removeRoleAll* 123
  - Vererbung* 118
  - Verfeinerung* 120
- Zend\_Auth 124, 127
  - authenticate* 127
  - dateibasiert* 128
  - Datenbank* 124
  - MVC* 129
  - setCredentialColumn* 125
  - setIdentityColumn* 125
  - setTableName* 125
- Zend\_Cache 137
  - APC* 150
  - ausschalten* 139
  - automatic\_cleaning\_factor* 140
  - automatic\_serialization* 139
  - Backend File* 149
  - Backends* 149
  - cacheByDefault* 144
  - cachedEntity* 146
  - cachedFunctions* 145
  - caching* 139
  - call* 146
  - debug\_header* 142
  - Einträge löschen* 153
  - end* 138
  - factory* 138
  - Frontend Class* 146
  - Frontend Core* 149
  - Frontend File* 147
  - Frontend Function* 144
  - Frontend Output* 140
  - Frontend Page* 141
  - Frontends* 139
  - lifetime* 139
  - load* 148
  - Memcached* 150
  - MVC* 142, 144
  - remove* 154
  - Serialisierung* 139
  - start* 138, 140
  - Tags* 153
  - write\_control* 139
- Zend\_Client\_Http
  - Authentifizierung* 341
- Zend\_Config 194
  - Arrays* 195
  - Default-Werte* 196
  - INI-Dateien* 197
  - Konstante* 199
  - Sektionen* 199
  - SimpleXML* 201
  - XML-Dateien* 200
- Zend\_Console\_Getopt 203
  - Argumente* 207
  - Cluster* 204
  - Flag* 203
  - getRemainingArgs* 207
  - getUserMessage* 206
  - Hilfe* 207
  - setHelp* 207
- Zend\_Controller\_Action 30
  - getResponse* 46
- Zend\_Controller\_Action\_Exception 42
- Zend\_Controller\_Dispatcher\_Exception 42
- Zend\_Controller\_Front 27, 29
  - getInstance* 29
  - setControllerDirectory* 31, 33
  - setParam* 30
- Zend\_Controller\_Request\_Http 38
- Zend\_Currency 389
  - Caching* 394
  - getCurrencyList* 394
  - getName* 393
  - getRegionList* 394
  - getSymbol* 393
  - name* 391
  - setFormat* 391
  - toCurrency* 390

## Index

- Währungen* 390
- Zend\_Date 395
  - add* 402
  - checkLeapYear* 407
  - compare* 405
  - compareDate* 406
  - compareTime* 406
  - get* 398
  - getArpa* 401
  - getDate* 401
  - getGmtOffset* 396
  - getMonth* 402
  - getTime* 401
  - getTimezone* 396
  - isDate* 407
  - isEarlier* 407
  - isLater* 407
  - isLeapYear* 407
  - isToday* 407
  - isTomorrow* 407
  - isYesterday* 407
  - now* 395
  - set* 400
  - setLocale* 395
  - setTimezone* 396
  - sub* 402
  - Zeitzonen* 396
- Zend\_Db 59, 65
  - Aggregat-Funktionen* 79
  - Aliasnamen* 64
  - auslesen von Daten* 69
  - beginTransaction* 73
  - Binding* 67
  - bindParam* 67
  - Case-Folding* 62
  - DELETE* 77
  - DISTINCT* 82
  - execute* 66
  - factory* 60
  - fetchAll* 70
  - fetchAssoc* 70
  - fetchCol* 71
  - fetchColumn* 71
  - Fetch-Mode* 69
  - fetchObject* 71
  - fetchOne* 72
  - fetchPairs* 72
  - GROUP BY* 82
  - HAVING* 82
  - INSERT* 76
  - JOIN* 83
  - lastInsertId* 75
  - limitPage* 81
  - Optionen* 61
  - PDO-Optionen* 62
  - Platzhalter* 65
  - Platzhalter, benannte* 65
  - prepare* 66
  - quote* 63
  - quoteColumnAs* 64
  - quoteIdentifier* 64
  - quoteInto* 63
  - quoteTableAs* 64
  - rollback* 73
  - rowCount* 68
  - SELECT* 78
  - Sequenzen* 75
  - setFetchMode* 69
  - Sortierung* 82
  - Transaktionen* 73
  - unterstützte Datenbanken* 61
  - UPDATE* 77
  - WHERE* 80
- Zend\_Db\_Expr 94
- Zend\_Db\_Profile
  - getLastQueryProfile* 112
  - getQueryProfile* 112
- Zend\_Db\_Profiler 110
  - getProfiler* 110
  - getTotalElapsedSecs* 112
  - getTotalNumQueries* 112
  - setFilterElapsedSecs* 113
- Zend\_Db\_Select 78
- Zend\_Db\_Table 88
  - Abhängigkeiten* 100
  - alternative Syntax* 108
  - Einfügen von Daten* 93
  - fetchAll* 97
  - fetchRow* 97
  - findDependentRowset* 106
  - insert* 93, 94
  - Kaskadierendes Löschen* 109
  - LIMIT* 97
  - Löschen von Daten* 95
  - Natural Key* 90
  - onDelete* 103
  - onUpdate* 103
  - Primärschlüssel* 90

- save* 98
- SELECT* 95
- Sequenz* 91
- setFromArray* 98
- setUpTableName* 89
- Tabellenname* 89
- Zend\_Db\_Table\_Row 95
- Zend\_Db\_Table\_Rowset 95
- Zend\_Feed 209
  - Atom* 214
  - Attribute auslesen* 211
  - Elemente auslesen* 210
  - generieren von Feeds* 218
  - import* 210
  - importArray* 218
  - RSS* 211
- Zend\_Feed P Feeds
- Zend\_Filter 165
  - Absoluter Pfad* 170
  - alphanumerische Zeichen* 167
  - Basename* 168
  - Buchstaben filtern* 168
  - Chain* 166
  - eigene Filter* 174
  - Entitäten* 169
  - Großbuchstaben* 171
  - HTML-Tags* 172
  - Integer-Werte* 170
  - Kleinbuchstaben* 170
  - RealPath* 170
  - setEncoding* 170
  - Whitespaces* 171
  - Ziffern* 169
- Zend\_Filter\_Input 175
  - Escape-Filter* 179
  - Fehlermeldungen* 179
  - getEscaped* 178
  - getMessages* 183
  - getMissing* 183
  - getUnescaped* 178
  - getUnknown* 183
  - hasInvalid* 183
  - hasUnknown* 183
  - setDefaultEscapeFilter* 179
  - Wildcard* 176
- Zend\_Gdata 245
  - Account-Authentication* 250
  - Allgemeines* 246
  - Authentifikation* 246
  - AuthSub* 250
  - AuthSubRevokeToken* 253
  - Calendar* 255
  - CAPTCHA* 248
  - delete* 270
  - Erstellungsdatum* 265
  - Event-Query* 262, 265
  - eventStatus* 260
  - Exception Handling* 254
  - Feed* 256
  - Geburtstage* 269
  - gelöschte Termine* 260
  - getAuthSubSessionToken* 252
  - getAuthSubTokenUri* 251
  - getCalendarEventEntry* 270
  - getCalendarListFeed* 261
  - getCaptchaToken* 248, 250
  - getCaptchaUrl* 248, 250
  - getCustom* 273
  - getHttpClient* 247, 249
  - getQueryUrl* 267
  - getSpreadsheetFeed* 271
  - insertEvent* 267
  - insertRow* 277
  - Kalendereintrag anlegen* 265
  - Kalendereintrag auslesen* 270
  - Kalendereintrag löschen* 270
  - Kalendereinträge ändern* 269
  - magische Methoden* 256
  - newContent* 266
  - newTitle* 266
  - One-Time-Use-Token* 251
  - regelmäßige Termine* 269
  - save* 270
  - setProjection* 262
  - setPublishedMax* 265
  - setPublishedMin* 265
  - setSpreadsheetQuery* 276
  - setUpdatedMax* 265
  - setUpdatedMin* 265
  - setVisibility* 262
  - Sortierreihenfolge* 265
  - Spreadsheets auslesen* 271
  - Structured Query* 276
  - Suche* 265
  - Tabellenblatt auslesen* 273
  - Tabellenfelder auslesen* 273
  - Tabellenzeile aktualisieren* 278
  - Tabellenzeile löschen* 277

## Index

- Tabellenzeilen einfügen* 277
- Termin anlegen* 265
- updateRow* 278
- Veränderungsdatum* 265
- verschiedene Kalender* 261
- Volltextsuche* 265
- Zugriff auf Elemente* 256
- Zend\_Gdata\_Entry 256
- Zend\_Gdata\_Feed 256
- Zend\_Http 335
- Zend\_Http\_Client 336
  - Adapter* 351
  - addCookie* 350
  - Cookie setzen* 350
  - Cookies* 346
  - Datei-Download* 345
  - getCookies* 347
  - getHeader* 345
  - getHeaders* 345
  - getHeadersAsString* 345
  - isError* 343
  - isRedirect* 343
  - isSuccessfull* 343
  - Port* 337
  - Proxy* 351
  - Redirect* 338
  - request* 336
  - Server-Antworten* 342
  - setAuth* 341
  - setConfig* 338
  - setCookieJar* 347
  - setFileUpload* 340
  - setParameterGet* 339
  - setParameterPost* 339
  - setUri* 338
  - Uploads* 340
  - User-Agent* 338
- Zend\_Http\_CookieJar 346
- Zend\_Http\_Response 336
- Zend\_Json 306
  - decode* 307
  - encode* 306
- Zend\_Locale 369, 376
  - checkDate* 380
  - getBrowser* 370
  - getDate* 378
  - getDateFormat* 378
  - getFloat* 376
  - getHttpCharset* 371
  - getInteger* 376
  - getLanguage* 370
  - getLanguageTranslation* 373
  - getQuestion* 373
  - getTranslationList* 371
  - toFloat* 374
  - toInteger* 376
  - toNumber* 374
  - Zahlen* 373
- Zend\_Log 184
  - addFilter* 190
  - addWriter* 186
  - CSV-Format* 192
  - Datenbank* 186
  - eigene Items* 194
  - eigener Log-Level* 189
  - Einträge filtern* 190
  - Einträge formatieren* 191
  - Einträge verwerfen* 188
  - Filtern nach Priorität* 191
  - Formatter* 192
  - log* 189
  - Loglevel* 185
  - Mock-Writer* 188
  - setEventItem* 194
  - Stream-Writer* 186
  - suppress* 190
  - Writer* 186
  - XML-Format* 192
- Zend\_Mail 279
  - addBcc* 281
  - addCc* 281
  - addHeader* 281
  - addTo* 280
  - Anhänge* 283
  - Anzahl der E-Mails* 289
  - Attachments* 283
  - Bcc* 281
  - Betreff* 280
  - Body* 280
  - CC* 281
  - Content-Disposition* 284
  - Content-Type* 284
  - copyMessage* 305
  - countMessages* 289
  - createAttachment* 283
  - Dringlichkeit* 282
  - E-Mails abholen* 288
  - E-Mails löschen* 298

- E-Mails versenden* 280
- Expunge* 298, 301
- Flag entfernen* 302
- Flags, IMAP* 300
- getContent* 293
- getCurrentFolder* 304
- getFolders* 302
- getHeader* 292
- getHeaders* 293
- getLocalName* 302
- getMessage* 289
- getNumberByUniqueId* 299
- getPart* 293
- getUniqueId* 299
- hasFlag* 300
- Header* 281
- Header auslesen* 292
- HTML-E-Mails* 282
- IMAP* 288
- IMAP, erweiterte Möglichkeiten* 300
- isMultipart* 293
- Kopie* 281
- kopieren von E-Mails* 304
- Ordner* 302
- Papierkorb* 302
- POP3* 288
- Recent-Flag* 301
- removeMessage* 298
- setBodyHtml* 282
- setBodyText* 280
- setFlags* 300
- setReturnPath* 281
- setSubject* 280
- setType* 284
- SMTP-Server* 286
- Subject* 280
- undelete* 299
- Unique-IDs* 299
- verschieben von E-Mails* 304
- Verschlüsseln* 288
- Zeichensatz* 280
- Zend\_Measure* 385
  - add* 389
  - compare* 389
  - convertTo* 388
  - equals* 389
  - getType* 387
  - setValue* 387
  - sub* 389
- Zend\_Pdf* 308
  - Bilder* 330
  - Clipping* 331
  - CMYK* 320
  - Datei* 312
  - DPI* 308
  - drawCircle* 326
  - drawEllipse* 327
  - drawImage* 330
  - drawLine* 323
  - drawPolygon* 328
  - drawRectangle* 324
  - drawText* 311
  - einlesen* 333
  - Ellipse* 327
  - Farben* 320
  - Fließtexte* 315
  - fontWithName* 311, 313
  - fontWithPath* 313
  - Füllfarbe* 321
  - Graustufen* 321
  - imageWithPath* 330
  - Kreis* 326
  - laden* 333
  - Linienfarbe* 321
  - load* 333
  - mehrzeilige Texte* 315
  - Meta-Informationen* 331
  - OpenType-Font* 313
  - Polygon* 328
  - Rechteck* 324
  - render* 312
  - RGB* 320
  - save* 312
  - Schriftart* 311
  - Schriftarten* 313
  - Schriftschnitt* 314
  - Seiten hinzufügen* 309
  - setFillColor* 321
  - setFont* 311
  - setLineColor* 321
  - setLineDashingPattern* 322
  - setLineWidth* 322
  - setStyle* 311
  - Stil* 310
  - Stile* 309
  - Text ausgeben* 311
  - TrueType-Font* 313
  - Zeichnen* 322

## Index

- Zend\_Rest 361
  - get* 362
  - MVC* 365
  - post* 362
- Zend\_Rest\_Client
  - setUri* 362
- Zend\_Rest\_Server 363
  - addFunction* 364
  - Fehlerbehandlung* 367
  - setClass* 364
- Zend\_Service\_Amazon 220
  - Antworten* 228
  - ASIN* 225
  - DetailPageURL* 225
  - itemLookup* 228
  - itemSearch* 221
  - Kategorien* 222
  - Manufacturer* 227
  - Offers* 231
  - Response-Group* 228
  - Rückgabewert* 225
  - SearchIndex* 221
- Zend\_Service\_Flickr 232
  - firstResultPosition* 233
  - Image-Objekt* 234
  - Optionen* 233
  - Seitenweises blättern* 232
  - tagSearch* 232
  - totalResultsAvailable* 233
  - userSearch* 232
- Zend\_Service\_Yahoo 236
  - Bildersuche* 242
  - imageSearch* 242
  - News-Suche* 240
  - searchNews* 240
  - webSearch* 236
  - Websuche* 236
- Zend\_Session 129
  - expireSessionCookie* 132
  - forgetMe* 132
  - Namespaces* 129
  - rememberMe* 132
  - Save-Handler* 133
  - Session starten* 130
  - setSaveHandler* 133
  - start* 130
  - Time-out* 132
- Zend\_Translate 381
  - Adapter* 381
  - addTranslation* 383
  - CSV-Dateien* 384
  - Datenquellen* 381
  - Delimiter* 385
  - getMessageIds* 384
  - Gettext* 381
  - isAvailable* 384
  - isTranslated* 384
  - Mastersprache* 383
  - separator* 385
  - setLocale* 384
- Zend\_Uri 352
  - URI analysieren* 354
- Zend\_Validate 154
  - alphanumerische Daten* 156
  - Arrays* 163
  - Datum* 157
  - E-Mail-Adresse* 157
  - Fehlermeldungen* 155
  - Fließkomma* 160
  - getMessages* 154
  - Größer als* 160
  - hexadezimale Zahlen* 160
  - Hostnames* 161
  - Integer-Werte* 163
  - IP-Adressen* 163
  - isValid* 154
  - Kleiner als* 164
  - Kreditkartennummern* 157
  - regulärer Ausdruck* 164
  - setMessage* 155
  - String-Länge* 165
  - Texte* 156
  - Variable* 164
  - Zahlen* 156
  - Ziffernfolge* 157
- Zend\_View 34, 36
  - escape* 37
- Zend\_View\_Exception 34
- Zend\_XmlRpc 355
  - Fehlerbehandlung* 358
  - setClass* 357
  - system-Methoden* 357
- Zend\_XmlRpc\_Client 359
  - getLastRequest* 361
- ZFForum 16
- ZFTutorials 16
- Ziffern prüfen 157