WoltLab Community Framework

Technische Dokumentation

WoltLab GmbH
http://www.woltlab.com

Inhaltsverzeichnis

I. Das WoltLab Community Framework 8

1. Einleitung 9
1.1. Über das WoltLab Community Framework . . . 9
1.2. Begriffe . . . 9
1.3. Lizenz . . . 10

2. Installation 11
2.1. Systemvoraussetzungen . . . 11
2.2. Download . . . 11
2.3. Durchführung der Installation . . . 11

3. Das Paketsystem 13
3.1. Grundlegendes . . . 13
3.2. Mitglieferte Basispakete . . . 13

4. Schnelleinstieg 18
4.1. Die Klasse WCF . . . 18
4.1.1. Datenbankzugriff . . . 18
4.1.2. Templatesystem . . . 19
4.1.3. Angemeldeter Benutzer . . . 19
4.1.4. Session . . . 19
4.1.5. Sprachsystem . . . 20
4.1.6. Cache . . . 20
4.1.7. Request . . . 20
4.2. Die Klasse DatabaseObject . . . 20

5. Datenbankschnittstelle 22
5.1. Methoden der Klasse Database . . . 22

6. Das Templatesystem 27
6.1. Grundlegende Syntax für Template-Designer . . . 27
6.1.1. Variablen . . . 27
6.1.2. Kommentare . . . 28
6.1.3. Funktionen . . . 28

6.1.4. Modifikatoren . . . 29
6.2. Das Templatesystem für Programmierer . . . 29
6.2.1. assign . . . 30
6.2.2. append . . . 30
6.2.3. assignByRef . . . 30
6.2.4. clearAssign . . . 30
6.2.5. clearAllAssign . . . 31
6.2.6. display . . . 31
6.2.7. fetch . . . 31
6.2.8. registerPrefilter . . . 31
6.3. Fest eingebaute Funktionen . . . 32
6.3.1. if,else,elseif – Fallunterscheidungen . . . 32
6.3.2. include . . . 32
6.3.3. foreach . . . 34
6.3.4. section . . . 34
6.3.5. capture . . . 35
6.4. Mitgelieferte Funktionen . . . 36
6.4.1. append . . . 36
6.4.2. assign . . . 36
6.4.3. counter . . . 37
6.4.4. cycle . . . 37
6.4.5. fetch . . . 38
6.4.6. htmloptions . . . 38
6.4.7. htmlcheckboxes . . . 38
6.4.8. implode . . . 41
6.4.9. lang . . . 41
6.4.10. pages . . . 42
6.5. Mitgelieferte Modifikatoren . . . 42
6.5.1. concat . . . 42
6.5.2. date . . . 43
6.5.3. encodejs . . . 43
6.5.4. filesize . . . 43
6.5.5. fulldate . . . 43
6.5.6. shorttime . . . 43
6.5.7. time . . . 44
6.5.8. truncate . . . 44
6.6. Das Templatesystem erweitern . . . 44
6.6.1. Eigene Modifikatoren . . . 44
6.6.2. Eigene Funktionen . . . 45
6.6.3. Eigene Block-Funktionen . . . 45
6.6.4. Eigene Prefilter . . . 46
6.6.5. Eigene Compiler-Funktionen . . . 46

7. Sprachverwaltung **50**

7.1. Grundlegendes . . . 50
7.2. Verwendung von Sprachvariablen . . . 50
7.3. Aufbau von Sprachdateien . . . 51
7.4. Sprachdateien einbinden . . . 52

8. Events **54**
8.1. Ereignisse auslösen . . . 54
8.2. Ereignisse nutzen . . . 54

9. Sessions **55**
9.1. SessionFactory . . . 55
9.2. Session . . . 55

10. Caching **57**

11. RequestHandler & die Page-, Form- und Action-Klassen **59**
11.1. RequestHandler . . . 59
11.2. Page und AbstractPage . . . 60
11.3. Form und AbstractForm . . . 60
11.4. Action und AbstractAction . . . 61

II. Pakete erstellen **62**

12. WCF-Pakete **63**
12.1. Das Format . . . 63
12.2. Die package.xml Datei . . . 63
12.2.1. Der Paketbezeichner . . . 64
12.2.2. Mehrsprachige Paketnamen und Paketbeschreibungen . . . 64
12.2.3. <requiredpackage>-Tag . . . 64
12.2.4. <optionalpackage>-Tag . . . 66
12.2.5. Installations- und Updateinstruktionen . . . 66
12.3. Verschiedene Pakettypen . . . 67

13. Package Installation Plugin **68**
13.1. Datei-basierte PIPs . . . 68
13.1.1. Das Files-PIP . . . 68
13.1.2. Das Templates-PIP . . . 69
13.1.3. ACPTemplates-PIP . . . 70
13.1.4. Das Style-PIP . . . 70
13.1.5. Das PIPs-PIP . . . 71
13.2. Import PIPs (XML) . . . 71
13.2.1. Das EventListener-PIP . . . 72
13.2.2. Das Cronjobs-PIP . . . 73
13.2.3. Das Options-PIP . . . 75

13.2.4. Das UserOptions-PIP . . . 78
13.2.5. Das GroupOptions-PIP . . . 79
13.2.6. Das FeedReaderSource-PIP . . . 79
13.2.7. Das Help-PIP . . . 80
13.2.8. Das BBCodes-PIP . . . 81
13.2.9. Das Smilies-PIP . . . 82
13.2.10. Das SearchableMessageType-PIP . . . 83
13.2.11. Das PageLocation-PIP . . . 84
13.2.12. Das HeaderMenu-PIP . . . 85
13.2.13. Das UserCPMenu-PIP . . . 87
13.2.14. Das ACPMenu-PIP . . . 87
13.2.15. Das StyleAttributes-PIP . . . 88
13.2.16. Das Languages-PIP . . . 89
13.3. Sonstige PIPs . . . 91
13.3.1. Das SQL-PIP . . . 91
13.3.2. Das Script-PIP . . . 92
13.3.3. Das TemplatePatch-PIP . . . 92
13.3.4. Das ACPTemplatePatch-PIP . . . 94
13.4. Eigene PIPs . . . 94
13.4.1. Das Interface . . . 94
13.4.2. Abstrakte Klassen . . . 96
13.4.3. Installation des PIPs . . . 97

14. Endanwendungen **98**
14.1. Paket erstellen . . . 98
14.2. Ableitung der Klassen WCF und WCFACP . . . 98
14.3. Erstellung einer IndexPage-Klasse . . . 99
14.4. Erstellung einer index.php-Datei . . . 99

III. Anhänge **101**

15. Events **102**
15.1. Events der freien WCF-Pakete . . . 102

16. Stilvariablen **104**
16.1. Global . . . 104
16.1.1. Allgemein . . . 104
16.1.2. Seite . . . 105
16.1.3. Kästen . . . 106
16.1.4. Rahmen . . . 107
16.1.5. Formulare . . . 108
16.2. Text . . . 109
16.2.1. Textarten . . . 109

16.2.2. Verweise . . . 109
16.3. Buttons . . . 110
16.3.1. Kleine Buttons . . . 110
16.3.2. Große Buttons . . . 111
16.4. Menüs . . . 112
16.4.1. Hauptmenü . . . 112
16.4.2. Tabs . . . 113
16.4.3. Tab-Buttons . . . 113
16.4.4. Spaltenköpfe . . . 114
16.4.5. Extras . . . 115
16.5. Erweitert . . . 116
16.5.1. Nachrichtendarstellung . . . 116
16.5.2. Zusätzliche CSS-Deklarationen . . . 116
16.5.3. Kommentare . . . 117
16.6. Wertebereiche . . . 117

Teil I.

Das WoltLab Community Framework

1. Einleitung

1.1. Über das WoltLab Community Framework

Die Grundlage für das WoltLab Burning Board 3 (kurz WBB) bildet das WoltLab Community Framework (kurz WCF). Es wird die Basis für alle zukünftigen WoltLab–Produkte im Bereich der Community–Entwicklung sein. Es ist komplett objektorientiert in PHP 5 programmiert und liefert die Unterstützung für modulartige Pakete. Als erste Anwendung nutzt Burning Board 3 diese Funktionen. Die neue Forensoftware besteht aus mehreren kleineren Paketen, die sie sich mit anderen Anwendungen teilen kann.

Das Community Framework liefert auch Funktionen für das automatische Update von installierten Paketen. Dabei wird die Paketdatenbank des Herstellers (z. B. WoltLab) nach aktualisierten Versionen dieser Pakete abgefragt. Bei Bedarf werden die Updates automatisch heruntergeladen und ausgeführt. Außerdem ist es möglich, die Paketdatenbank des Herstellers manuell nach neuen Paketen zu durchsuchen, um Zusätze oder komplett neue Anwendungen zu installieren. Ein ausgewähltes Paket wird zusammen mit allen zusätzlich benötigen Paketen automatisch heruntergeladen und installiert. Ein installiertes Paket kann, solange es nicht von einem anderen Paket benötigt wird, mit einem einzigen Mausklick wieder deinstalliert werden.

1.2. Begriffe

Innerhalb dieser Dokumentation werden die folgenden Begriffe benutzt:

Paket ein Modul für das WCF

Stil optisches Erscheinungsbild einer Endanwendung (vergleichbar mit „Skin“, „Theme“ oder „Style“). Ein Stil besteht aus Wertdefinitionen für Farben, Schriften und Größen (CSS) sowie den dazugehörigen Templates.

Template XHTML–Vorlage mit Platzhaltern (sog. Templatevariablen)

1.3. Lizenz

Das WoltLab Community Framework als solches steht unter der „GNU Lesser General Public License“ Lizenz (kurz LGPL). Sie finden die vollständigen Lizenzbestimmungen unter folgender Adresse:

http://www.gnu.org/licenses/lgpl.html

In einfachen Worten gesagt, können Sie alle Pakete des WCF, die unter der LGPL stehen, für eigene sowohl kostenlose als auch kostenpflichtige Anwendungen verwenden.

Einige Pakete stehen unter der Burning Board Lizenz. Diese können nur genutzt werden, wenn für jede Installation der eigenen Anwendung eine entsprechende Lizenz bei WoltLab erworben wurde. Alternativ sind auch individuelle Lizenzbestimmungen verhandelbar.

2. Installation

2.1. Systemvoraussetzungen

Für die Installation von WoltLab Community Framework müssen folgende Voraussetzungen erfüllt werden:

- Ein Webserver mit PHP 5-Unterstützung
- Apache 2 mit PHP 5.2.x-Modul empfohlen
- Eine MySQL-Datenbank in Version 4.1.2 oder höher
- Ca. 6 MB Festplattenspeicher
- Ein FTP-Programm, um die Programmdateien auf den Webserver/Webspace zu laden

2.2. Download

Bitte verwenden Sie erstmal das WCF, welches in der Download-Datei der Forensoftware WoltLab Burning Board mitgliefert wird. Wir arbeiten noch an einem separaten Download des WCF.

2.3. Durchführung der Installation

Zur Installation des WCF benötigen Sie zwei Dateien, die innerhalb des heruntergeladenen Zip-Archivs zu finden sind:

install.php Das Installationsscript, welches komfortabel über den Browser ausgeführt werden kann.

WCFSetup.tar.gz Enthält alle Dateien und Pakete des Community Frameworks.

Diese beiden Dateien laden Sie bitte auf Ihren Webserver und führen die Installationsdatei `install.php` aus. Folgende Schritte sind dabei mindestens zu durchlaufen:

1. Auswahl der Sprache des Installationsassistenten
2. Akzeptieren der Lizenzbestimmungen
3. Systemvoraussetzungen prüfen
4. Installationsverzeichnis für das WCF auswählen
5. Zeichenkodierung und Sprache auswählen
6. Datenbankzugangsdaten eingeben
7. Administratorkonto anlegen
8. Eventuelle weitere Endanwendungen installieren

3. Das Paketsystem

3.1. Grundlegendes

Der elementare Bestandteil des WCF ist das Paketsystem. Pakete sind einzelne Komponenten, die eine bestimmte Funktionalität kapseln und von anderen Paketen genutzt werden können. Über das WCF lassen sich eine Vielzahl von Paketen installieren, automatisch aktualisieren und verwalten.

Ein Paket ist ein Archiv mit Dateien. Die wichtigste Datei ist die `package.xml`. Hier stehen alle wichtigen Angaben über das Paket wie z. B. dessen Name, von welchen anderen Paketen es abhängig ist und welche Aufgaben bei der Installation des Pakets durchgeführt werden müssen.

1. **einfaches Paket** Innerhalb eines einfachen Pakets werden lediglich Funktionen bereitgestellt, ohne dass diese aktiv genutzt werden. Ein Paket liefert z. B. PHP-Klassen, eigene Datenbanktabellen, Sprachdateien, Grafiken und Templates. Pakete können von anderen Paketen abhängig sein.
2. **Endanwendung** Das Paket Endanwendung nutzt die Funktionen der anderen (einfachen) Pakete und stellt eigene Schnittstellen bereit. Nur Endanwendungen verfügen über eine eigene grafische Oberfläche.
3. **Plugin** Ein Plugin erweitert ein existierendes Paket um neue Funktionen. Dazu müssen vorher von dem Paket definierte Schnittstellen genutzt werden. Das Plugin ist daher optional, es wird nicht direkt vom Paket vorausgesetzt.

3.2. Mitglieferte Basispakete

Das Community Framework stellt die nötigsten Grundfunktionen von Webapplikationen bereit. Weiterführende Funktionen werden als freie Basispakete mitgeliefert. Zu den fest eingebauten Systemen des WCF gehören:

Paketverwaltung Das Paketsystem verwaltet die installierten Pakete, installiert, aktualisiert und deinstalliert Pakete.

Datenbankanbindung Das WCF stellt einen sog. „Database Abstraction Layer“ bereit, der die Benutzung einer MySQL Datenbank erleichtert.

Benutzerverwaltung und Authentifizierung Die Benutzerverwaltung erstellt, bearbeitet und entfernt Benutzerkonten. Benutzer können sich authentifizieren (z. B. auch gegen LDAP[1]).

Gruppenverwaltung Die Gruppenverwaltung übernimmt die Einteilung der Benutzer in Gruppen.

Sessions Das WCF verwaltet Sessions für Sie.

Sprachverwaltung Erlaubt mehrsprachige Benutzerschnittstellen.

Caching Das Caching System erlaubt die Zwischenspeicherung von Daten, die schnell zugänglich sein müssen im Dateisystem.

Events Das Eventsystem ermöglicht das Ausführen von Funktionen zu vorher definierten Events.

Templatesystem Das WCF stellt ein eigenes Templatesystem zur Ausgabe der Seiten zur Verfügung.

com.woltlab.wcf

Die oben beschriebenen Systeme werden durch die Installation vom WCF bereitgestellt. Zusätzlich werden alle wichtigen Package Installation Plugins (siehe Kapitel 13 auf Seite 68) mitgeliefert.

com.woltlab.wcf.data.cronjobs

Cronjobs sind aus der Unix-Welt bekannte zeitgesteuerte Aufgaben. Echte Cronjobs stehen meistens nicht zur Verfügung, da dafür ein eigener Server benötigt wird. Daher wurde die Funktionalität von Cronjobs mit AJAX und PHP nachempfunden. Durch diese Nachbildung können regelmäßige Aufgaben dennoch durchgeführt werden. Zu solchen Aufgaben könnte die Aktualisierung von Anzeigen zählen. Häufig werden bestimmte Anzeigen zwischengespeichert (Caching), da sie oft von den Benutzern angefordert werden und die Datenbank entlastet werden soll. Durch einen Cronjob kann so ein Zwischenspeicher beispielsweise einmal täglich aktualisiert werden.

com.woltlab.wcf.data.feed.reader

Informationen auf Webseiten werden heute häufig über ein RSS-Feed zur Verfügung gestellt. Dieses Plugin erlaubt es die Informationen solcher Feeds regelmäßig einzulesen, um sie innerhalb der eigenen Anwendung darzustellen.

[1] http://de.wikipedia.org/wiki/Ldap

com.woltlab.wcf.data.help

Dieses Paket liefert eine einfache Möglichkeit, um eine Endnutzer-Hilfe für die eigene Anwendung zu implementieren. Es ist so konzipiert, dass über eine XML-Datei eigene Hilfe-Themen definiert werden können. Dem Endanwender steht eine Suchfunktion zur Verfügung, des Weiteren kann für jedes Thema ein Referrer[2] definiert werden. Befindet sich also ein Benutzer auf einer Seite und ruft die Hilfe auf, so kann er gleich zur entsprechenden Hilfe-Seite geleitet werden.

com.woltlab.wcf.data.image

Innerhalb dieses Moduls werden Hilfsfunktionen geliefert, die zum Verarbeiten von Bildern genutzt werden können. So wird z. B. eine `Thumbnail`-Klasse bereitgestellt, um kleine Vorschaubilder zu erzeugen.

com.woltlab.wcf.data.image.captcha

Captcha-Bilder sind für die Wahrung der Sicherheit von Webanwendungen ein wichtiges Instrument. Überall wo Formulareingaben von unregistrierten Benutzern gemacht werden können, sind Angriffspunkte für automatisierte Spam-Roboter gegeben. Dem ist mit diesen Sicherheitsbildern vorzubeugen. Der Nachteil dieser Lösung besteht darin, dass solche Bilder nicht barrierefrei sind und blinde oder sehbehinderte Menschen die Buchstaben und Zahlen nicht entziffern können.

com.woltlab.wcf.data.message

Das Message-Paket ist immer dann zu verwenden, wenn eine typische Nachrichtenform vorhanden ist. Die besteht in der Regel aus einer Überschrift und einem Text.

com.woltlab.wcf.data.message.attachment

Beim Verfassen von Nachrichten werden häufig auch andere Medien als Dateianhang eingebunden. Dies ist mit Hilfe dieses Pakets möglich.

[2]Der Referrer ist die Internetadresse, auf der die aktuelle Seite aufgerufen wurde.

com.woltlab.wcf.data.message.bbcode

Beim Verfassen von Nachrichten auf Webseiten können diese mit Hilfe von BBCodes formatiert werden. Sie sind eine Art HTML-Ersatz, da die direkte Verwendung von HTML aus Sicherheitsgründen häufig nicht zu empfehlen ist.

com.woltlab.wcf.data.message.censorship

Um bestimmte Wörter einer Nachricht zensieren zu können, ist dieses Modul notwendig. Dafür muss eine Liste von Wörtern angelegt werden, die nicht verwendet werden dürfen. Beim Abschicken der Nachricht wird diese nach den Wörtern auf der Liste überprüft, ggf. kommt es zu einer Fehlermeldung.

com.woltlab.wcf.data.message.poll

Um eine Nachricht zu schreiben, die eine Umfrage enthält, kann dieses Paket genutzt werden.

com.woltlab.wcf.data.message.search

Die Suche ist ein zentraler Bestandteil eines jeden Systems, welches Nachrichten speichert. Die konkrete Implementierung dieser Suche verwendet den MySQL-Fulltext-Index.

com.woltlab.wcf.data.page

Das Paket enthält die Templates für den äußeren Rahmen einer Seite. Zusätzlich wird das PageLocation-PIP (siehe Kapitel 13.2.11 auf Seite 84) mitgebracht.

com.woltlab.wcf.data.page.headerMenu

In diesem Modul wird die Funktionalität bereitgestellt, die notwendig ist, um ein Hauptmenü auf der Seite anzeigen zu lassen. Über eine XML-Datei können beliebig viele Einträge definiert werden. Während der Installation werden diese in die Datenbank übernommen.

com.woltlab.wcf.form.message

In diesem Paket sind all die Funktionalitäten enthalten, die beim Verfassen oder Bearbeiten einer Nachricht benötigt werden.

com.woltlab.wcf.form.message.wysiwyg

Der WYSIWYG-Editor wird zum Verfassen von Nachrichten angeboten. Dies erhöht den Komfort für den Nutzer, da er beim Erstellen seiner Nachricht direkt sieht, wie sich Formatierungen des Textes auswirken und er die entsprechenden Codes nicht kennen muss.

com.woltlab.wcf.form.user

Sämtliche Benutzeraktionen, die mit dem eigenen Benutzerkonto zu tun haben, sei es registrieren, anmelden, E-Mail-Adresse ändern oder ein vergessenes Passwort anfordern, werden von diesem Paket abgedeckt.

com.woltlab.wcf.page.user.profile

Mit Hilfe dieser Komponente können die öffentlichen Profile der Benutzer angezeigt werden.

com.woltlab.wcf.system.style

Das Style-Paket liefert für viele Benutzerelemente in einer Applikation bereits die entsprechende Formatierung. Es werden CSS-Klassen bereit gestellt, die z. B. für die Darstellung von Tab-Menüs, tabellarischen Auflistungen oder Buttons zuständig sind.

com.woltlab.wcf.system.template.pack

Sollen innerhalb der Anwendung verschiedene Templategruppen unterstützt werden, so ist dieses Paket zu verwenden. So können den Benutzern der Community verschiedene Stile zur Auswahl angeboten werden, die auf unterschiedlichen Templates aufbauen.

com.woltlab.wcf.system.template.plugin.includePHP

Dieses Plugin ermöglicht die Verwendung von PHP-Code innerhalb der Templates.

4. Schnelleinstieg

Um schnell zu Ergebnissen zu gelangen, bietet dieses Kapitel einen kurzen Überblick über die Klasse `WCF` und den objekt-orientierten Ansatz von WoltLab Community Framework.

4.1. Die Klasse `WCF`

Die Klasse `WCF` (in der Datei `wcf/lib/system/WCF.class.php`) ist die zentrale Klasse von WoltLab Community Framework. Sie ermöglicht z. B. den Zugriff auf die Datenbank, das Template- oder das Sprachsystem.

Das WCF ist objektorientiert programmiert und für die Datenbank, das Templatesystem, etc. gibt es eigene Objekte (Instanzen einer Klasse). Das WCF verwendet diese Objekte nach dem sog. *Singleton Pattern*[1], wodurch sichergestellt ist, dass anwendungsweit genau je eine Instanz dieser Klassen existiert. Der Zugriff erfolgt über die Klasse `WCF`, in der die Instanzen als statische Membervariablen gespeichert sind.

4.1.1. Datenbankzugriff

Mit `WCF::getDB()` erhält man die Instanz des Datenbankobjektes um Anfragen an den Datenbankserver zu senden und zu verarbeiten, z. B.

```
// Sende die SQL Abfrage $sql an den Server
$result = WCF::getDB()->sendQuery($sql);

// hole das Ergebnis ab und durchlaufe jede Zeile
while ($row = WCF::getDB()->fetchArray($result)) {
  // ..
}
```

Genaue Informationen über die Verwendung des Datenbanksystems finden Sie in Kapitel 5 auf Seite 22.

[1] http://de.wikipedia.org/wiki/Singleton_(Entwurfsmuster)

4.1.2. Templatesystem

Mit `WCF::getTPL()` erhält man die Instanz des Templatesystems.

```
// Variablenzuweisung
WCF::getTPL()->assign('action', $action);

// Zeige das Template $templateName an
WCF::getTPL()->display($templateName);
```

Detaillierte Informationen erhalten Sie im Kapitel 6 auf Seite 27.

4.1.3. Angemeldeter Benutzer

`WCF::getUser()` liefert das Objekt, das den aktuellen Besucher der Seite repräsentiert. Normalerweise ist das eine Instanz der Klasse `UserSession` oder einer davon abgeleiteten Klasse (in der Endanwendung WoltLab Burning Board z. B. die abgeleiteten Klassen `WBBUserSession` oder `WBBGuestSession`).

Die Klasse `UserSession` leitet sich wiederum von der Klasse `User` ab.

```
WCF::getUser()->userID
WCF::getUser()->username
```

4.1.4. Session

`WCF::getSession()` liefert die Instanz der Klasse `Session` (oder einer davon abgeleiteten Klasse), um z. B. Session-Variablen zu registrieren oder auszulesen.

```
// hole alle Session Variablen
$sessionVars = WCF::getSession()->getVars();

// hole nur die Session Variable 'test'
$test = WCF::getSession()->getVar('test');

// registriere die Variable 'blub'
WCF::getSession()->register('blub', $blub);
```

Im Kapitel 9 auf Seite 55 erfahren Sie mehr zu diesem Thema.

4.1.5. Sprachsystem

`WCF::getLanguage()` liefert die Instanz der Klasse `Language`, z. B. zum Laden von Sprachvariablen:

```
WCF::getLanguage()->get("name.der.variablen");
```

4.1.6. Cache

Der Cache wird verwendet, um häufig verwendete Daten im Dateisystem zwischenzuspeichern. `WCF::getCache()` liefert die Instanz der Klasse `CacheHandler`, die den Cache steuert. Im Kapitel 10 auf Seite 57 erfahren Sie mehr.

```
// Daten auslesen
$data = WCF::getCache()->get('spiders');
```

4.1.7. Request

`WCF::getRequest()` liefert die Instanz der Klasse `RequestHandler`. Der RequestHandler kann verwendet werden, um festzustellen, welche Seite aufgerufen wurde. Siehe Kapitel 11 auf Seite 59.

4.2. Die Klasse `DatabaseObject`

Alle Klassen, die einen Datensatz repräsentieren, leiten sich von der abstrakten Klasse `DatabaseObject` ab (nachfolgend auch `Data`-Klassen genannt). Die zu verwaltenden Daten werden dabei in der Klassenvariablen `$data` gespeichert. In der Praxis ist `$data` oft ein assoziatives Array.

Die vielleicht wichtigste und nützlichste Eigenschaft der `DatabaseObject`-Klasse ist die Verwendung der magischen Methode `__get()` von PHP5[2].

Dadurch ist es möglich mit der Schreibweise `$dataObject->myVar` auf die Variable `$dataObject->data[$myVar]` zuzugreifen – vorausgesetzt, `$dataObject->myVar` existiert nicht als Klassenvariable. Erst diese magische Methode erlaubt Zugriffe wie z. B. `WCF::getUser()->username`.

In der Regel werden Sie zu jeder `Data`-Klasse eine entsprechende `DataEditor`-Klasse finden. Die `DataEditor`-Klasse enthält häufig statische Methoden wie `create()`, `delete()`

[2] siehe http://www.php.net/manual/de/language.oop5.magic.php
und http://www.php.net/manual/de/language.oop5.overloading.php

und `insert()`. Während die `Data`-Klassen dafür sorgen, dass Daten ausgelesen werden, so verwendet man die `DataEditor`-Klassen dazu neue Datensätze zu erstellen oder vorhandene zu bearbeiten bzw. zu löschen.

5. Datenbankschnittstelle

Obwohl das WoltLab Community Framework in Ansätzen eine Datenbankabstraktionsschicht mitliefert, ist in der aktuellen Version[1] derzeit nur eine Unterstützung für das Datenbanksystem MySQL implementiert. Implementierungen für andere Datenbanksysteme müssen sich von der abstrakten Klasse `Database` ableiten.

5.1. Methoden der Klasse `Database`

sendQuery

resource **sendQuery** (*string* $query)

Die Methode `sendQuery()` sendet eine SQL-Abfrage an den Datenbankserver und gibt die Ergebnis-Kennung zurück. Falls die SQL Abfrage fehlschlägt, wird eine `DatabaseException` geworfen.

Bemerkung: Die SQL-Abfrage sollte nicht mit einem abschließenden Semikolon enden.

sendUnbufferedQuery

resource **sendUnbufferedQuery** (*string* $query)

`sendUnbufferedQuery()` sendet eine SQL-Anfrage an den Datenbankserver, ohne gleich die Datensätze des Ergebnisses zu holen[2].

Bemerkung: Auf Ergebniskennungen von `sendUnbufferedQuery()` können `countRows()` und `getAffectedRows()` nicht angewendet werden.

[1] Stand: WCF Version 1.0.1

[2] siehe auch http://www.php.net/manual/de/function.mysql-unbuffered-query.php

fetchArray

array **fetchArray** ([*resource* $queryID = null], [*int* $type = null])

`fetchArray()` liefert einen Datensatz einer SQL-Abfrage als Array. Falls der Parameter `$queryID` nicht angegeben wurde, wird die zuletzt gesendete SQL-Abfrage angenommen.

Der zweite optionale Parameter `$type` bestimmt den Typ des Arrays. Der Standardwert `Database::SQL_ASSOC` liefert ein assoziatives Array, `Database::SQL_NUM` ein numerisches Array und `Database::SQL_BOTH` beides.

getFirstRow

array **getFirstRow** (*string* $query, [*integer* $type = null])

`getFirstRow()` wird verwendet, wenn man von einer SQL-Abfrage nur den ersten Datensatz benötigt. Dabei ist `getFirstRow()` eine abkürzende Schreibweise für die Ausführung von `sendQuery()` und `fetchArray()`.

Wie bei `fetchArray()` bestimmt der optionale Parameter `$type` den Typ des Arrays.

getResultList

array **getResultList** (*string* $sql)

`getResultList()` sendet eine SQL-Abfrage, holt alle Datensätze ab und gibt diese in einem mehrdimensionalem Array zurück. `getResultList()` bietet sich an, wenn Sie alle Datensätze einer SQL-Abfrage auslesen, aber danach im Skript nicht mehr weiterverarbeiten möchten.

countRows

integer **countRows** ([*resource* $queryID = null])

`countRows()` liefert die Anzahl der Datensätze im Ergebnis einer SQL-Abfrage. `countRows()` ist nur auf SELECT-Abfragen anwendbar. Wenn Sie die Zahl der manipulierten Datensätze einer UPDATE-, INSERT- oder DELETE-Abfrage ermitteln wollen, benutzen Sie bitte `getAffectedRows()`.

Wird keine `$queryID` angegeben, wird die zuletzt gesendete SQL-Abfrage angenommen.

getAffectedRows

integer **getAffectedRows** ()

`getAffectedRows()` gibt die Zahl der durch die zuletzt ausgeführten INSERT-, DELETE- oder UPDATE-Abfrage manipulierten Datensätze zurück.

getInsertID

int **getInsertID** ()

`getInsertID()` liefert die ID, die bei der letzten INSERT-Abfrage für ein Feld vom Typ AUTO_INCREMENT vergeben wurde. `getInsertID()` liefert 0, wenn die vorhergehende INSERT-Abfrage keinen AUTO_INCREMENT Wert erzeugt hat.

Falls Sie den Wert zur späteren Verwendung speichern möchten, stellen Sie sicher, dass Sie `getInsertID()` direkt nach der INSERT-Abfrage aufrufen, die einen Wert erzeugt hat.

seekResult

void **seekResult** ([*integer* $queryID = null], *integer* $offset)

`seekResult()` bewegt den Lesezeiger einer Menge von Ergebnissen zur Position `$offset`. Dabei beginnt der Lesezeiger bei der Position 0, d. h. Position 0 zeigt auf den ersten Datensatz.

Falls der Parameter `$queryID` nicht angegeben wurde, wird die zuletzt gesendete SQL Abfrage angenommen.

registerShutdownUpdate

void **registerShutdownUpdate** (*string* $query, [*integer* $key = null])

`registerShutdownUpdate()` fügt eine UPDATE-Anweisung in eine Warteliste von UPDATE-Anweisungen ein, die am Ende des Skriptes ausgeführt werden.

escapeString

string **escapeString** (*string* $string)

`escapeString()` ist ein Wrapper für die PHP-Funktion `mysql_real_escape_string()` und muss zur Maskierung von Sonderzeichen auf Strings angewendetet werden, die in einer SQL-Abfrage verwendet werden sollen.

Bemerkung: Anstelle des Aufrufes `WCF::getDB()->escapeString($string)` kann man auch `escapeString($string)` schreiben.

getErrorDesc

string **getErrorDesc** ()

Liefert die Fehlermeldung einer zuvor ausgeführten Datenbankabfrage.

getErrorNumber

int **getErrorNumber** ()

Liefert die Nummer einer Fehlermeldung einer zuvor ausgeführten Datenbankabfrage.

getVersion

string **getVersion** ()

Liefert die Versionsnummer des Datenbanksystems.

getDBType

string **getDBType** ()

Gibt den verwendeten Datenbanktyp zurück.

getDatabaseName

string **getDatabaseName** ()

Gibt den Namen der verwendeten Datenbank zurück.

getCharset

string **getCharset** ()

Gibt den verwendeten Zeichensatz zurück.

getTableNames

array **getTableNames** ([*mixed* $database = ''])

Gibt einen Array mit allen in der Datenbank vorhandenen Tabellen zurück.

getTableStatus

array **getTableStatus** ()

Gibt einen Array mit detaillierten Informationen zu jeder Tabelle in der Datenbank zurück.

6. Das Templatesystem

Das Templatesystem des WoltLab Community Framework orientiert sich in Syntax und Funktionsweise an Smarty[1], dem „kompilierenden Templatesystem“, ohne jedoch dessen gewaltigen Funktionsumfang zu erreichen.

Das Templatesystem liest die Templatedateien (Dateiendung `.tpl`) und wandelt sie direkt in PHP-Skripte um. Templates müssen deshalb nur noch nach einer Änderung geparst werden. Für die folgenden Aufrufe wird einfach das generierte PHP-Skript ausgeführt.

Ausserdem lässt sich das Templatesystem um eigene Funktionen erweitern, mehr dazu später in Kapitel 6.6 auf Seite 44.

6.1. Grundlegende Syntax für Template-Designer

Alle Templatebefehle werden von geschweiften Klammern umschlossen, z. B. `{include file="boardList"}`.

6.1.1. Variablen

Bevor man eine Templatevariable in den Templates verwenden kann, muss man sie in dem zugehörigen PHP-Skript deklarieren und der Variable einen Wert zuweisen.

Dies funktioniert mit der Methode `assign()` der Klasse `Template`, die in 6.2.1 auf Seite 30 ausführlicher beschrieben wird.

Templatevariablen werden innerhalb der Templates wie PHP-Variablen mit dem Dollarzeichen $ angesprochen. Zudem kann auch auf PHP-Konstanten zugriffen werden. Programm 6.1 auf der nächsten Seite zeigt einige Beispiele.

[1] http://smarty.php.net

`@` und `#`

Bei der Ausgabe von Variablen maskiert das Templatesystem HTML-Sonderzeichen automatisch. Falls dies in bestimmten Fällen nicht gewünscht ist, kann dies mit dem @-Zeichen unterdrückt werden: `{@$variable}`.

Warnung: Benutzen Sie @ nur, wenn Sie wünschen, dass HTML-Sonderzeichen als solche dargestellt werden.

Bei der Ausgabe von numerischen Werten ist die Raute # nützlich. Die Raute bewirkt, dass Zahlen automatisch formatiert werden, d. h. Kommazahlen werden auf 2 Stellen gerundet, und mit dem sprachenabhängigen Dezimaltrennzeichen versehen. Wenn nötig wird ein Tausendertrennzeichen eingefügt: `{#$number}`

Programm 6.1 Variablenzugriff innerhalb von Templates

```
{$variable}
{@$variable}
{#$variable}
{CONST}
{$array.0} (für $array[0])
{$array.key} (für $array['key'])
{$array.$key} (für $array[$key])
{$object->var}
{$object->method($foo)}
{WCF::getUser()->userID}
```

6.1.2. Kommentare

Man kann einzelne Abschnitte in Templates durch Kommentare vom Parsen ausschließen. Ein Kommentar wird durch `{*` geöffnet und durch `*}` geschlossen.

```
{* Dies ist ein Kommentar *}
```

Kommentare können sich auch über mehrere Zeilen erstrecken.

Kommentare werden in der XHTML-Ausgabe nicht an den Client geschickt.

6.1.3. Funktionen

Templatefunktionen werden mit der Syntax `{funktionsname}` aufgerufen. Es wird zwischen normalen Funktionen und Blockfunktionen unterschieden. Blockfunktionen sind Funktionen der Form `{block} ... {/block}`, die ähnlich wie XML-Tags einen Text umschließen. Solche Funktionen verarbeiten den umschlossenen Text als Eingabe.

Parameter

Mit der Syntax `{funktionsname param1=$val1 param2="value"}` kann man Paramter an die Funktion `funktionsname` übergeben. Bei Blockfunktionen werden die Parameter im öffnenden Funktionsaufruf eingefügt. `{block param1=$val1} ... {/block}`

6.1.4. Modifikatoren

Modifikatoren werden verwendet, um eine Funktion auf eine Variable anzuwenden. Dabei können sowohl eigene Modifikatoren als auch die PHP-Funktionen aufgerufen werden. Die Syntax ist kurz und einfach: Man hängt `|modifier` an die Variable an, um den Modifikator `modifier` aufzurufen, z. B. `{$var|empty}`.

Parameter

Ein Modifikator wird auf eine Templatevariable angewandt. Jedoch ist es auch möglich dem Modifikator weitere Parameter zu übergeben. Hierzu hängt man die Werte der Parameter durch Doppelpunkte (:) getrennt an den Modifikatornamen an:
`{$var|modifier:"val1":"val2":$foo}`

Program 6.2 zeigt einige typische Anwendungsfälle für Modifikatoren.

Programm 6.2 Anwendung von Modifikatoren auf Templatevariablen

```
{@TIME_NOW|fulldate}
{$title|truncate:40:"..."}

{function param1=$val1|modifier}
{function param1=$val1|modifier:$val2:"val3"}
```

6.2. Das Templatesystem für Programmierer

In diesem Abschnitt werden die wichtigsten Methoden der Klasse `Template` vorgestellt.

6.2.1. assign

void **assign** (*mixed* $variable, [*mixed* $value = ”])

Die `assign()`-Methode deklariert und initialisiert eine oder mehrere Templatevariablen, so dass sie anschließend in den Templates verwendet werden können. Die `assign()`-Methode akzeptiert ein Name-, Wertepaar oder alternativ ein assoziatives Array mit Name-, Wertepaaren (siehe Programm 6.3).

Programm 6.3 Variablenzuweisung mit `assign`

```
// ..
$variable = 'Wert';
$object = new myObject();
$array = array(1,2,3,4);

WCF::getTPL()->assign('variable', $variable);
WCF::getTPL()->assign(array(
  'object' => $object,
  'array' => $array
));
```

6.2.2. append

void **append** (*mixed* $variable, [*mixed* $value = ”])

Die `append()`-Methode hängt Inhalte an eine bestehende Templatevariable an (Konkatenation, Verkettung). Wie auch die `assign()`-Methode akzeptiert auch die `append()`-Methode ein Name-, Wertepaar oder ein assoziatves Array mit Name-, Wertepaaren.

6.2.3. assignByRef

void **assignByRef** (*string* $variable, *mixed* &$value)

`assignByRef()` erstellt eine Templatevariable mit einer Referenz auf eine PHP-Variable. Als Argument wird der Name der Template- sowie der PHP-Variablen erwartet.

6.2.4. clearAssign

void **clearAssign** (*mixed* $variable)

Die `clearAssign()`-Methode löscht eine oder mehrere Templatevariablen aus dem Speicher. Die Methode erwartet als Argument den Namen der zu löschenden Templatevariablen oder ein Array mit den Namen der zu löschenden Templatevariablen.

6.2.5. clearAllAssign

void **clearAllAssign** ()

`clearAllAssign()` löscht alle Templatevariablen. Die Methode besitzt keine Argumente.

6.2.6. display

void **display** (*string* $templateName, [*boolean* $sendHeaders = true])

Um ein Template darstellen zu lassen und die Ausgabe an den Client zu schicken, wird die `display()`-Methode verwendet.

Die Methode erwartet den Namen des Templates. Das zweite boolsche Argument `$sendHeaders` ist optional und bestimmt, ob die HTTP HEADER der Seite gesendet werden sollen.[2].

Die `display()`-Methode kümmert sich automatisch um die Kompilierung des Templates.

Bei aktivierter Sendung der HTTP HEADER werden vor und nach der Ausgabe des Templates noch die Events `shouldDisplay` und `didDisplay` ausgelöst.

6.2.7. fetch

string **fetch** (*string* $templateName)

Wenn man ein Template darstellen möchte, aber die Ausgabe nicht gleich an den Client schicken sondern z. B. noch weiterverarbeiten möchte, verwendet man die `fetch()`-Methode, die die Ausgabe eines Templates zurückgibt.

Die Methode erwartet den Namen des Templates als Argument.

6.2.8. registerPrefilter

void **registerPrefilter** (*mixed* $name)

Ein Prefilter ist ein Programm, das vor der Kompilierung von Templates auf den Templateinhalt angewandt werden kann. Um einen Prefilter zu verwenden, muss dieser vorher mit `registerPrefilter()` registriert werden. `registerPrefilter()` erwartet dabei als Argument den Namen des Prefilters oder einen Array mit den Namen von mehreren Prefiltern.

[2]siehe Klasse `HeaderUtil`

Wie man eigene Prefilter erstellt erfahren Sie in 6.6.4 auf Seite 46.

6.3. Fest eingebaute Funktionen

6.3.1. `if`,`else`,`elseif` – Fallunterscheidungen

Die Funktion `{if} ... {/if}` wird für Fallunterscheidungen in Templates verwendet. Dabei hat man (fast) die gleichen Möglichkeiten wie in PHP-Skripten. Wie in PHP gibt es auch die Ausdrücke `{else}` und `{elseif}` (und als Alias dazu `{else if}`).

Programm 6.4 Syntax der {if}-Funktion

```
{if BEDINGUNG1}
  Bedingung1 ist wahr
{elseif BEDINGUNG2}
  Bedingung2 ist wahr
{elseif BEDINGUNG3}
{else}
  keine der Bedingungen ist wahr
{/if}
```

Die Syntax der `{if}`-Funktion ist in Programm 6.4 zu sehen. Dabei sind `BEDINGUNG1`, `BEDINGUNG2` und `BEDINGUNG3` Ausdrücke, in denen neben Variablen auch Modifikatoren und die aus PHP bekannten Operatoren `||`,`&&`,`>`, `>=`,`<`,`<=`,`==`,`===`,`!=`,`!===`[3] erlaubt sind.

Warnung: Im WoltLab Community Framework 1.0.0 sind keine geklammerten Ausdrücke in Bedingungen der `if`-Funktion möglich[4].

6.3.2. include

Mit der `{include}`-Funktion kann man ein anderes Template in das aktuelle Template laden. So kann man z. B. häufig benutze Abschnitte in ein eigenes Templates auslagern, um sie dann in anderen Templates einzubinden.

Die `{include}`-Funktion erwartet den Parameter `file`, der das einzubindende Template definiert.

Dem eingebundenen Template können per Paramter auch zusätzliche Templatevariablen übergeben werden: `{include file='header' var1='foo' var2=$foo}`. Eingebundene

[3] siehe http://www.php.net/manual/de/language.operators.logical.php
und http://www.php.net/manual/de/language.operators.comparison.php
[4] http://www.woltlab.com/forum/index.php?page=Thread&threadID=122382

Programm 6.5 Beispiele für die {if}-Funktion

```
{if $userMessages|isset}{@$userMessages}{/if}

{if $boards|count > 0}
  ...
  ...

  {if $hasChildren}<ul id="category{@$boardID}">{else}</li>{/if}
  {if $openParents > 0}{@"</ul></li>"|str_repeat:$openParents}{/if}
{/if}

{if $this->user->userID}
  {if $this->user->activationCode && REGISTER_ACTIVATION_METHOD == 1}
    ....
  {/if}
  ...
{elseif !$this->session->spiderID}
  ...
{/if}
```

Programm 6.6 Syntax der {include}-Funktion

```
{include file='headerMenu'}
{include file='header' sandbox=false}
{include file='template' assign='var' append=true}
```

Templates werden in einem „Sandkasten“ ausgeführt, d. h. Änderungen an Templatevariablen innerhalb des eingebundenen Templates haben keinen Einfluss auf die Variablen in dem aktuellen Template. Dieses Verhalten kann über den optionalen `sandbox`-Parameter gesteuert werden.

Über den optionalen Parameter `assign` kann der Inhalt des eingebundenen Templates einer Templatevariablen zugewiesen werden. Wenn man den optionalen Parameter `append=true` übergibt, wird der Inhalt an die in `assign` definierte Templatevariable angehängt.

Parameter	erforderlich	Vorgabe	Bedeutung
`file`	ja	–	Name der eingebundenen Template
`assign`	nein	–	Variable, der der Inhalt der Template zugewiesen werden soll
`append`	nein	false	Inhalt an die Templatevariable anhängen?
`sandbox`	nein	true	„Sandkasten“ aktivieren/deaktivieren
`beliebig`	nein	–	Variablenübergabe an die eingebundene Template

Tabelle 6.1.: Paramter der {include}-Funktion

6.3.3. foreach

Die Funktion `{foreach from=$array item='val'} ... {/foreach}` ist das Äquivalent zur `foreach`-Schleife[5] `foreach($array as $val) { ... }` in PHP.

Der benötigte Parameter `from` gibt das Array an, das in der Schleife durchlaufen werden soll. Im benötigten Parameter `item` gibt man den Namen der Variablen an, der das jeweils aktuelle Element des Arrays zugewiesen wird.

Bei Verwendung des optionalen Parameters `key` wird zusätzlich der jeweils aktuelle Schlüssel in der durch `key` gewählten Variablen gespeichert.

Programm 6.7 Beispiele für die {foreach}-Schleife

```
<?php
$arr = array("eins", "zwei", "drei");
$arr2 = array(
  array("a", "b"),
  array("y", "z")
);

WCF::getTPL()->assign(array(
  'arr' => $arr,
  'arr2' => $arr2
));
?>

{* foreach-Schleife *}
{foreach from=$arr item=value key=key}
  Schlüssel: {$key}<br />
  Wert: {$value}<br />
{/foreach}

{* auch mehrdimensionale Arrays kann man mit
   verschachtelten foreach-Schleifen durchlaufen *}
{foreach from=$arr2 item=v1}
  {foreach from=$v1 item=v2}
    {$v2}<br />
  {/foreach}
{/foreach}
```

6.3.4. section

Die `{section}`-Funktion wird für Schleifendurchläufe verwendet, bei denen man die Funktionalität einer `for`-Schleife aus PHP benötigt. Die Syntax ist im Vergleich zu `{foreach}` etwas aufwändiger, dafür bietet `{section}` einige zusätzliche Möglichkeiten.

Der benötigte Parameter `name` enthält den Namen der Zählvariablen, die für die Iteration benutzt wird, z. B. `name=i` oder `name=j` etc.

[5] http://www.php.net/manual/de/control-structures.foreach.php

Der benötigte Parameter `loop` bestimmt die Anzahl der Iterationen. `loop` kann ein Array sein – in diesem Fall ist die Anzahl der Elemente die Anzahl der Iterationen. Man kann aber auch eine ganze Zahl übergeben.

Parameter	erforderlich	Vorgabe	Bedeutung
`loop`	ja	–	Bestimmt die Anzahl der Iterationen, kann eine ganze Zahl sein oder ein Array
`name`	ja	–	Name der Zählvariablen.
`start`	nein	0	Startposition der Zählvariablen (ganze Zahl)
`step`	nein	1	Schrittweite der Iteration (ganze Zahl)
`max`	nein	–	maximale Zahl an Iterationen (natürliche Zahl)
`show`	nein	true	Ausgabe der `{section}` anzeigen oder nicht

Tabelle 6.2.: Paramter der {section}-Funktion

Mit den Paramtern `start`, `step` und `max` kann man die Iteration manipulieren. Normalerweise startet die mit `name` definierte Zählvariable bei 0. Mit `start` kann auch ein anderer Startwert vorgegeben werden.

`step` gibt die Schrittweite der Iteration vor. Standardwert ist 1. So iteriert man beispielsweise mit `start=1` und `step=2` über die Menge {1, 3, 5, 7, ...}.

`max` beschränkt die Anzahl der Iterationen auf einen bestimmten Wert.

Mit `show=false` kann man den Schleifendurchlauf deaktivieren.

Bemerkung: Beim Verschachteln von `{section}`-Schleifen muss man darauf achten für jede Schleife einen eigenen, eindeutigen `name`-Parameter zu wählen.

Tabelle 6.2 bietet eine Übersicht der Parameter und Programm 6.8 ein Beispiel.

Programm 6.8 Beispiele für die Verwendung der {section}-Schleife

```
{* gibt 0,2,4,6,8 aus *}
{section name=i loop=10 step=2}
  {$i}<br />
{section}
```

6.3.5. capture

Die `{capture} ... {/capture}` Funktion aktiviert die Ausgabepufferung[6] und speichert die Ausgabe des Templatecodes, der von der Funktion umschlossen wird, in einer

[6] durch die PHP-Funktionen `ob_start` und `ob_get_contents`

Templatevariablen. Dabei werden im Templatecode wie gewohnt auch Variablen und andere Templatefunktionen ausgewertet.

Der Name der Templatevariablen wird durch den Parameter `assign="var"` definiert. Der Inhalt zwischen `{capture assign="var"}` und `{/capture}` wird dann in der Templatevariablen `$var` gespeichert.

Möchte man den Templatecode an den bestehenden Inhalt einer Templatevariable anhängen (anstatt den Inhalt der Templatevariablen zu überschreiben), kann man den Parameter `append` verwenden: `{capture append="var"} ... {/capture}`

6.4. Mitgelieferte Funktionen

6.4.1. append

Die `{append}`-Funktion erlaubt es, im Template die `append()`-Methode des aktuellen Templateobjekts aufzurufen. Siehe dazu auch 6.2.2 auf Seite 30.

```
{append var=name value="foo"}
```

Es handelt sich hierbei technisch um ein `TemplateCompilerPlugin` - diese Methode wird also bereits während des Kompilierens des Templates aufgerufen.

6.4.2. assign

Wie auch bei der `{append}`-Funktion ruft `{assign}` die gleichnamige Methode des aktuellen Templateobjekts auf. Siehe dazu auch 6.2.1 auf Seite 30.

```
{assign var=name value="foo"}
```

Es handelt sich auch hierbei technisch um ein `TemplateCompilerPlugin`.

6.4.3. counter

Die `{counter}`-Funktion wird benutzt um eine Zahlenfolge zu erstellen. Beim ersten Aufruf wird eine Zählvariable initialisiert, die bei jedem folgenden Aufruf erhöht wird. Dabei sind sowohl der Startwert (`start`) als auch die Schrittweite des Zählers (`skip`) frei wählbar (Standardwerte sind 1).

Sie können auch bestimmen, ob der Wert des Zählers in dem Template ausgegeben werden soll (`print`, Standard ist `true`) und ob der Wert einer Templatevariablen zugewiesen werden soll (`assign`).

Über den Parameter `direction="down"` können Sie den Zähler in umgekehrter Reihenfolge zählen lassen. Das hat den gleichen Effekt wie ein negativer Wert für den `skip` Paramter.

Bemerkung: Falls Sie mehrere voneinander unabhängige Zähler benötigen, müssen Sie diesen mit dem `name`-Parameter einen eindeutigen Namen zuweisen.

Parameter	erforderlich	Vorgabe	Bedeutung
`name`	nein	default	Interner Name des Zählers
`start`	nein	1	Startwert des Zählers (ganze Zahl)
`skip`	nein	1	Schrittweite des Zählers (ganze Zahl)
`direction`	nein	up	Richtung des Zählers („up“ oder „down“)
`print`	nein	true	Zähler ausgeben
`assign`	nein	–	Zähler einer Templatevariablen zuweisen

Tabelle 6.3.: Parameter der {counter}-Funktion

Programm 6.9 Beispiel für die Verwendung der {counter}-Funktion

```
{foreach from=$boards item=child}
  ...
  {counter assign=boardNo print=false}

  ...
  {$boardNo}
  ...

{/foreach}
```

6.4.4. cycle

Mit `{cycle}` können Sie abwechselnd zwischen zwei Werten alternieren. Dies wird häufig benötigt, um z. B. in der Ausgabe einer Tabelle die Zeilen abwechselnd zu formatieren.

Parameter	erforderlich	Vorgabe	Bedeutung
`values`	ja	-	Zwei komma-separierte Werte, zwischen denen zirkuliert wird
`name`	nein	default	Interner Name des Zyklus
`print`	nein	1	Werte ausgeben
`advance`	nein	1	Nächsten Wert automatisch holen
`reset`	nein	0	Zurücksetzen

Tabelle 6.4.: Parameter der {cycle}-Funktion

Programm 6.10 Beispiel für die Verwendung der {cycle}-Funktion

```
{foreach from=$boards item=child}
  ...
  <tr style="background-color: {cycle values="#eee,#fff"}">
  ...
  </tr>
  ...
{/foreach}
```

6.4.5. fetch

Die `{fetch}`-Funktion ermöglicht es, den Inhalt von Dateien auszugeben. Diese können auf dem lokalen Dateisystem liegen, aber auch durch HTTP oder FTP abgerufen werden. Intern wird die php-Funktion `file_get_contents()` verwendet.

Der Parameter `assign` ist hierbei optional. Sie können den Inhalt der Datei damit in eine Templatevariable speichern und erst zu einem späteren Zeitpunkt ausgeben.

```
{fetch file='x.html'}
{fetch file='x.html' assign=var}
```

Es handelt sich auch hierbei technisch um ein `TemplateCompilerPlugin`.

6.4.6. htmloptions

Die Funktion `{htmloptions}` erzeugt eine Ausgabe eine HTML-Select-Box mit den dazugehörigen Options-Elementen.

6.4.7. htmlcheckboxes

Die Funktion `{htmlcheckboxes}` erzeugt eine Auflistung mehrerer Checkbox-Elemente als HTML-Code.

Parameter	erforderlich	Vorgabe	Bedeutung
`name`	nein	-	Name der Select-Box
`options`	ja[a]	-	Angabe eines assoziativen Arrays mit Options-Werten und Options-Namen.
`output`	ja[b]	-	Angabe eines Arrays mit Options-Namen.
`values`	ja[c]	-	Angabe eines Arrays mit Werten, der jeweiligen Optionen.
`selected`	nein	-	Die angegebene Option ist bereits markiert (Mehrfachmarkierung über ein Array möglich).
`disableEncoding`	nein	0	Ruft intern die `htmlspecialchars()`-Methode auf, um vor fehlerhaften Eingaben zu schützen.

[a] Alternativ kann `output` und `values` bzw. nur `output` verwendet werden.
[b] Alternativ kann `options` verwendet werden.
[c] Kann nur in Verbindung mit `output` verwendet werden. Bei leeren Werten wird eine Optionsgruppe angelegt. Nachfolgende Werte sollten in einem verschachtelten Array gespeichert werden.

Tabelle 6.5.: Parameter der {htmloptions}-Funktion

Programm 6.11 Beispiel für die Verwendung der {htmloptions}-Funktion

```
{* Mögliches Array mit Werten *}
$array = array(
  "key1" => "value1",
  "key2" => "value2"
  "option group" => array(
    "keyA" => "valueA",
    "keyB" => "valueB"
  );
);

{* Mögliche Verwendung von htmloptions *}
{htmloptions options=$array}
{htmloptions options=$array selected=$selected}
{htmloptions options=$array selected=$selectedArray}
{htmloptions options=$array name="x"}
{htmloptions output=$outputArray}
{htmloptions output=$outputArray values=$valueArray}

{* Mögliche Ausgabe *}
<select>
  <option label="output" value="value">output</option>
</select>
```

Parameter	erforderlich	Vorgabe	Bedeutung
`name`	ja	-	Name der Checkbox-Liste
`options`	ja[a]	-	Angabe eines assoziativen Arrays mit Checkbox-Werten und -Beschriftungen.
`output`	ja[b]	-	Angabe eines Arrays mit Checkbox-Beschriftungen.
`values`	ja[c]	-	Angabe eines Arrays mit Werten, der jeweiligen Checkboxen.
`selected`	nein	-	Das angegebene Element ist bereits markiert (Mehrfachmarkierung über ein Array möglich).
`disableEncoding`	nein	0	Ruft intern die `htmlspecialchars()`-Methode auf, um vor fehlerhaften Eingaben zu schützen.
`separator`	nein	-	Angabe von Text/HTML, dass vor jedem Element eingefügt wird.

[a] Alternativ kann `output` und `values` bzw. nur `output` verwendet werden.
[b] Alternativ kann `options` verwendet werden.
[c] Kann nur in Verbindung mit `output` verwendet werden.

Tabelle 6.6.: Parameter der {htmlcheckboxes}-Funktion

Programm 6.12 Beispiel für die Verwendung der {htmlcheckboxes}-Funktion

```
{* Mögliches Verwendung von htmlcheckboxes *}
{htmlcheckboxes name="x" options=$array}
{htmlcheckboxes name="x" options=$array selected=$selected}
{htmlcheckboxes name="x" options=$array selected=$selectedArray}
{htmlcheckboxes name="x" output=$outputArray}
{htmlcheckboxes name="x" output=$outputArray values=$valueArray}

{* Mögliche Ausgabe *}
<label><input type="checkbox" name="x[]" value="value" />output</label>
```

6.4.8. implode

Mit `{implode}` können Sie den Inhalt von Arrays ausgeben lassen, so ähnlich wie Sie das bereits von `{foreach}` kennen. Der Unterschied ist hier lediglich der, dass die Elemente automatisch durch einen String verbunden werden. Dazu gibt es den Parameter glue, für den Sie einen beliebiger String angegeben können. Standardmäßig wird ", " angenommen.

Wenn Sie beispielsweise eine Auflistung von Namen realisieren wollen und diese durch ein Komma trennen wollen, so ist `{implode}` die beste Wahl. Bei `{foreach}` müssten Sie hierfür einen extra Zähler anlegen, um das Problem zu umgehen, dass vor jedem Element das Trennzeichen steht (denn vor dem ersten Element soll kein Trennzeichen stehen).

Das Beispiel 6.13 zeigt die Verwendung der `{implode}`-Funktion.

Programm 6.13 Beispiel für die Verwendung der {implode}-Funktion

```
{* Mögliches Verwendung von implode *}
{implode from=$names key=key item=name}{$key}: {$name}{/implode}<br />
{implode from=$names item=name glue=";"}{$name}{/implode}<br />

{* Mögliche Ausgabe *}
0: Name a, 1: Name b, 2: Name c
Name a;Name b;Name c
```

6.4.9. lang

Mit der `{lang}`-Funktion rufen Sie Sprachvariablen auf. Mehr zum Thema Sprachvariablen finden Sie in Abschnitt 7.2 auf Seite 50.

Innerhalb von `{lang}` geben Sie den Namen der gewünschten Sprachvariable an. Wenn dieser Name dynamisch ist, also aus einer anderen Variable kommt oder Sie der Sprachvariablen weitere Parameter übergeben, so handelt es sich um eine dynamische Sprachvariable. Siehe Beispiel 6.14.

Beachten Sie, dass es vorr. erst mit WCF 1.1 möglich sein wird, Templatescripting in dynamischen Sprachvariablen zu verwenden.

Programm 6.14 Beispiel für die Verwendung der {lang}-Funktion

```
{* statische Sprachvariable *}
{lang}wcf.example{/lang}

{* dynamische Sprachvariablen *}
{lang}wcf.example.$foo{/lang}
{lang foo=$foo bar=$bar}wcf.example{/lang}
```

Es handelt sich hierbei technisch um ein `TemplateCompilerPlugin` - diese Methode wird also bereits während des Kompilierens des Templates aufgerufen.

6.4.10. pages

Mit Hilfe der {pages}-Funktion können Sie einfach eine Seitenzahlennavigation erzeugen. Wenn sich die Inhalte eines Bereichs auf mehrere Seiten verteilen, lassen sich die einzelnen Seitenzahlen hiermit ausgeben.

Parameter	erforderlich	Vorgabe	Bedeutung
`pages`	ja	-	Seitenanzahl
`page`	nein	-	aktuelle Seite
`link`	ja	-	Link zur jeweiligen Seite (siehe Code-Beispiel)
`assign`	nein	-	Ausgabe kann an Variable übergeben werden.
`print`	nein	1	Ausgabe der Seitenzahlen.

Tabelle 6.7.: Parameter der {pages}-Funktion

Programm 6.15 Beispiel für die Verwendung der {pages}-Funktion

```
{* Erzeugt 10 Seitenzahlen*}
{pages pages=10 link='page-%d.html'}

{* Erzeugt 10 Seitenzahlen, wobei Seite 8 als aktuelle Seite markiert ist*}
{pages page=8 pages=10 link='page-%d.html'}

{* Weist die Seitenzahlen der Variable output zu, keine Ausgabe *}
{pages page=8 pages=10 link='page-%d.html' assign='output'}

{* Weist die Seitenzahlen der Variable output zu, inkl. Ausgabe *}
{pages page=8 pages=10 link='page-%d.html' assign='output' print=true}
```

6.5. Mitgelieferte Modifikatoren

6.5.1. concat

Der `concat`-Modifikator macht es möglich, mehrere Strings mit einander zu verbinden. Das folgende Beispiel liefert den String zurück, der sich nach PHP-Syntax aus `"left".$right` ergeben würde.

```
{"left"|concat:$right}
```

6.5.2. date

Der `date`-Modifikator bringt einen Unix-Timestamp, in ein für Menschen lesbares Format. Das Standard-Datumsformat enthält Jahr, Monat und Tag in folgender Ausgabe: „5. November 2007“.

```
{$timestamp|date}
{"132845333"|date:"%Y-%m-%d"}
```

6.5.3. encodejs

Mit Hilfe des `encodejs`-Modifikators ist es möglich Strings für die Verwendung als Single-Quote Javascript-String zu formatieren. Dabei werden Single Quotes und Zeilenumbrüche escaped.

```
{$string|encodejs}
{"bl''ah"|encodejs}
```

6.5.4. filesize

Der `filesize`-Modifikator formatiert Dateigrößen, welche in Byte angegeben werden.

```
{$string|filesize}
{123456789|filesize}
```

6.5.5. fulldate

Der `fulldate`-Modifikator formatiert einen Unix-Timestamp als Datum in folgender Ausgabe: „Montag, 5. November 2007, 11:32“.

```
{$timestamp|fulldate}
{"132845333"|fulldate}
```

6.5.6. shorttime

Mit `shorttime` formatieren Sie einen Unix-Timestamp mit Datum in folgender Ausgabe: „5. November 2007, 11:32“.

```
{$timestamp|shorttime}
{"132845333"|shorttime:"Y-m-d h:ia"}
```

6.5.7. time

Der `time`-Modifikator formatiert einen Unix-Timestamp mit Datum in folgender Ausgabe: „Montag, 5. November 2007, 11:32“. Der Unterschied zu `fulldate` liegt darin, dass das Datum ggf. durch „Heute“ oder „Gestern“ ersetzt wird.

```
{$timestamp|time}
{"132845333"|time:"%Y-%m-%d %I:%M%p"}
```

6.5.8. truncate

Mit dem `truncate`-Modifikator können Sie einen String nach einer bestimmten Anzahl von Zeichen abschneiden. Sie können auch definieren, was an diesen String angehangen werden soll.

```
{$foo|truncate:35:'...'}
```

6.6. Das Templatesystem erweitern

Das Templatesystem ist sehr flexibel und kann durch eigene Modifikatoren, Funktionen oder Prefilter erweitert werden. Dafür muss man lediglich eine PHP-Klasse programmieren, die ein bestimmtes Interface implementiert und diese PHP-Klasse im Ordner `wcf/lib/system/template/plugin` abspeichern (bzw. durch ein Paket in diesen Ordner installieren).

6.6.1. Eigene Modifikatoren

Eigene Modifikatoren müssen das Interface `TemplatePluginModifier`[7] implementieren. Dieses Interface enthält die Methode `execute()`.

string **execute** (*array* $tagArgs, *Template* $tplObj)

`$tagArgs` enthält alle Parameter des Modifiers. Der Inhalt der Templatevariablen, auf die der Modifier angewandt wird, steht in der Variablen `$tagArgs[0]` zur Verfügung. Ein zweiter Paramter des Modifiers würde in der Variablen `$tagArgs[1]` gespeichert werden usw.

Zusätzlich steht in der Variablen `$tplObj` eine Instanz der Template-Klasse zur Verfügung.

[7] in der Datei `wcf/lib/system/template/TemplatePluginModifier.class.php`

Als Rückgabewert wird der veränderte Variableninhalt erwartet.

Ein Beispiel sehen Sie in Programm 6.17 auf Seite 48.

6.6.2. Eigene Funktionen

Für eigene Funktionen gibt es das Interface `TemplatePluginFunction`[8] mit der Methode `execute()`.

string **execute** (*array* $tagArgs, *Template* $tplObj)

`$tagArgs` ist ein **assoziatives** Array mit den Parametern und `$tplObj` die Instanz der Template-Klasse. Ein Aufruf im Template `{funktion parameter="test"}` führt dazu, dass `$tagArgs['parameter']` den Wert „test“ enhält.

6.6.3. Eigene Block-Funktionen

Eigene Block-Funktionen müssen das Interface `TemplatePluginBlock`[9] implementieren.

Dieses Interface enthält die Methoden `execute()`, `init()` und `next()`.

string **execute** (*array* $tagArgs, *string* $blockContent, *Template* $tplObj)

void **init** (*array* $tagArgs, *Template* $tplObj)

boolean **next** (*Template* $tplObj)

Die Methode `init()` wird als erstes aufgerufen. Solange `next()` *true* zurückgibt, wird die Methode `execute()` in einer while-Schleife aufgerufen. Auf diese Art sind Schleifen-Funktionen realisierbar. Wenn Ihr Block-Funktion nur einmal aufgerufen werden soll, darf `next()` nur beim ersten Aufruf *true* zurückgeben. Sie können dazu z. B. eine eigene Klassenvariable benutzen.

Das Array `$tagArgs` enthält jeweils ein assoziatives Array mit den Parametern der Block-Funktion.

Warnung: `next()` muss auf jeden Fall irgendwann *false* zurückgeben, sonst erzeugt Ihre Block-Funktion eine Endlosschleife.

[8] in der Datei `wcf/lib/system/template/TemplatePluginFunction.class.php`
[9] in der Datei `wcf/lib/system/template/TemplatePluginBlock.class.php`

6.6.4. Eigene Prefilter

Eigene Prefilter müssen das Interface `TemplatePluginPrefilter`[10] implementieren. Dieses Interface enthält lediglich die Methode `execute`.

string **execute** (*string* $sourceContent, *TemplateCompiler* $compiler)

`execute()` kann dabei auf dem Inhalt eines Templates (`$sourceContent`) beliebige Operationen durchführen. Ausserdem steht mit `$compiler` die aktuelle Instanz des TemplateCompiler zur Verfügung.

Als Rückgabewert wird der (veränderte) Templateinhalt erwartet.

Eine Beispielimplementierung sehen Sie im Programm 6.17 auf Seite 48.

6.6.5. Eigene Compiler-Funktionen

Compiler-Funktionen müssen das Interface `TemplatePluginCompiler`[11] mit den Methoden `executeStart()` und `executeEnd()` implementieren.

string **executeStart** (*array* $tagArgs, *TemplateCompiler* $compiler)

string **executeEnd** (*TemplateCompiler* $compiler)

`executeStart()` wird für den öffnenden Templatetag (`{function param='value'}`) und `executeEnd()` für den schließenden Templatetag (`{/function}`) aufgerufen.

Der Rückgabewert ist jeweils der PHP-Code der in die kompilierte Template-Datei geschrieben wird.

Die Parameter des öffnenden Tags stehen nur in der Methode `executeStart()` als assoziatives Array zur Verfügung.

Eine Beispielimplementierung sehen Sie in 6.18 auf Seite 49.

[10] in der Datei `wcf/lib/system/template/TemplatePluginPrefilter.class.php`
[11] in der Datei `wcf/lib/system/template/TemplatePluginCompiler.class.php`

Programm 6.16 concat Modifier

```
<?php
// imports
if (!defined('NO_IMPORTS')) {
  require_once(WCF_DIR.'lib/system/exception/SystemException.class.php');
  require_once(WCF_DIR.'lib/system/template/TemplatePluginModifier.class.php');
  require_once(WCF_DIR.'lib/system/template/Template.class.php');
}

/**
 * The 'concat' modifier returns the string that
 * results from concatenating the arguments.
 * May have two or more arguments.
 *
 * Usage:
 * "left"|concat:$right
 *
 * @package com.woltlab.wcf.system.template.plugin
 * @author  Marcel Werk
 * @copyright 2001-2007 WoltLab GmbH
 * @license GNU Lesser General Public License
 *    <http://opensource.org/licenses/lgpl-license.php>
 */
class TemplatePluginModifierConcat implements TemplatePluginModifier {
  /**
   * @see TemplatePluginModifier::execute()
   */
  public function execute($tagArgs, Template $tplObj) {
    if (count($tagArgs) < 2) {
      throw new SystemException("concat modifier needs two
              or more arguments", 12001);
    }

    $result = '';
    foreach ($tagArgs as $arg) {
      $result .= $arg;
    }

    return $result;
  }
}
?>
```

Programm 6.17 lang Prefilter

```
<?php
// imports
if (!defined('NO_IMPORTS')) {
  require_once(WCF_DIR . 'lib/system/template/TemplatePluginPrefilter.class.php');
  require_once(WCF_DIR . 'lib/system/template/TemplateCompiler.class.php');
}

/**
 * The 'lang' prefilter compiles static language variables.
 * Dynamic language variables will catched by the 'lang' compiler function.
 * It is recommended to use static language variables.
 *
 * Usage:
 * {lang}foo{/lang}
 * {lang}lang.foo.bar{/lang}
 *
 * @package com.woltlab.wcf.system.template.plugin
 * @author  Marcel Werk
 * @copyright 2001-2007 WoltLab GmbH
 * @license GNU Lesser General Public License
 *          <http://opensource.org/licenses/lgpl-license.php>
 */
class TemplatePluginPrefilterLang implements TemplatePluginPrefilter {
  /**
   * @see TemplatePluginPrefilter::execute()
   */
  public function execute($sourceContent, TemplateCompiler $compiler) {
    $ldq = preg_quote($compiler->getLeftDelimiter(), '~');
    $rdq = preg_quote($compiler->getRightDelimiter(), '~');
    $sourceContent = preg_replace("~{$ldq}lang{$rdq}([\w\.]+){$ldq}/lang{$rdq}~e",
                                  'WCF::getLanguage()->get(\'$1\')', $sourceContent);

    return $sourceContent;
  }
}
?>
```

Programm 6.18 lang Compiler Funktion

```
<?php
// imports
if (!defined('NO_IMPORTS')) {
  require_once(WCF_DIR.'lib/system/template/TemplatePluginCompiler.class.php');
  require_once(WCF_DIR.'lib/system/template/TemplateCompiler.class.php');
}

/**
 * The 'lang' compiler function compiles dynamic language variables.
 * Warning: a dynamic language variable does not support template scripting.
 *
 * Usage:
 * {lang}$blah{/lang}
 * {lang var=$x}foo{/lang}
 *
 * @package com.woltlab.wcf.system.template.plugin
 * @author  Marcel Werk
 * @copyright 2001-2007 WoltLab GmbH
 * @license GNU Lesser General Public License
 *    <http://opensource.org/licenses/lgpl-license.php>
 */
class TemplatePluginCompilerLang implements TemplatePluginCompiler {
  /**
   * @see TemplatePluginCompiler::executeStart()
   */
  public function executeStart($tagArgs, TemplateCompiler $compiler) {
    $compiler->pushTag('lang');

    $newTagArgs = array();
    foreach ($tagArgs as $key => $arg) {
      $newTagArgs['$'.$key] = 'StringUtil::encodeHTML('.$arg.')';
    }

    $tagArgs = $compiler->makeArgString($newTagArgs);
    return "<?php \$this->tagStack[] = array('lang', array($tagArgs));
    ob_start(); ?>";
  }

  /**
   * @see TemplatePluginCompiler::executeEnd()
   */
  public function executeEnd(TemplateCompiler $compiler) {
    $compiler->popTag('lang');
    $hash = StringUtil::getRandomID();
    return "<?php \$_lang".$hash." = ob_get_contents(); ob_end_clean();
    echo WCF::getLanguage()->get(\$_lang".
    $hash.", \$this->tagStack[count(\$this->tagStack) - 1][1]);
    array_pop(\$this->tagStack); ?>";
  }
}
?>
```

7. Sprachverwaltung

Das WCF bietet eine umfangreiche Sprachverwaltung an, mit der es möglich ist, die Benutzeroberfläche von Anwendungen unabhängig von einer bestimmten Sprache zu entwickeln. In den Templates werden dazu Platzhalter (sog. Sprachvariablen) verwendet, die dann vom System mit dem entsprechenden Satz oder Wort ersetzt werden.

7.1. Grundlegendes

Sprachvariablen werden zur besseren Übersicht in Kategorien eingeteilt. Sprachvariablen und -kategorien werden in Sprachdateien (`xml`) angelegt. Diese werden einem Paket zugeordnet und bei der Paketinstallation von der Sprachverwaltung des WCF in die Datenbank geschrieben.

Um bei der Verwendung der Sprachvariablen eine optimale Performance zu erreichen, werden die Variablen vom System automatisch in ein schnell auslesbares Format gebracht und als Sprachdatei (`php`) im `language`-Ordner innerhalb des WCF-Verzeichnis angelegt. Für jede Sprache, jedes Paket und jede Sprachkategorie wird eine separate Sprachdatei erstellt. Der Dateiname für die Sprachdatei setzt sich wie folgt zusammen: `packageID_languageID_languageCategory.php`. Es wird nur für Sprachkategorien eine Datei erstellt, die in dem jeweiligen Paket und der jeweiligen Sprache auch Sprachvariablen enthalten. Sprachdateien werden nur für Pakete generiert, die eine Endanwendung sind. Alle anderen Paketen haben keine eigene Oberfläche und können nicht direkt auf Sprachdateien zugreifen.

7.2. Verwendung von Sprachvariablen

Zur Benennung von Sprachvariablen und Kategorien dürfen folgende Zeichen verwendet werden: „a-zA-Z0-9 _ -“. Wie bei Paketnamen ist es empfehlenswert bestimmte Namensräume zu verwenden z. B. „wcf.acp.global“. Kategorien dürfen dabei maximal aus drei Blöcken bestehen, die mit einem Punkt getrennt werden. Anhand des Namensraums „acp“ kann man nun sehr schnell alle Variablen erkennen, die nur im Adminbereich vorkommen.

Bemerkung: Bitte achten Sie darauf, dass der Name einer Sprachvariablen mit dem Namen der zugehörigen Kategorie beginnen muss, z. B. „wcf.acp.global.variable“.

Innerhalb eines Templates werden Sprachvariablen dann folgendermaßen eingebunden: `{lang}wcf.global.acp.variable{/lang}`. Um direkt innerhalb einer PHP-Datei eine Sprachvariable zu nutzen, verwenden Sie bitte die Methode `get("name.der.variable")` der Klasse `Language`. Als optionalen Parameter können Sie der Methode ein assoziatives Array mit Werten übergeben, die innerhalb der Sprachvariablen verwendet werden können - wie Sie im Beispiel 7.1 sehen können. Achten Sie darauf, dass die Schlüssel des Arrays das $-Zeichen enthalten.

Programm 7.1 Beispiel für die Verwendung der Methode Language::get()

```
// Programmcode
$value = 3;
WCF::getLanguage()->get('name.der.sprachvariable', array('$value' => $value);
...
// language.xml
<item name="name.der.sprachvariablen">
  <![CDATA[Ein Dreieck hat {$value} Ecken]]>
</item>
```

Wenn das System keinen zur Variable passenden Inhalt findet, wird lediglich der Name der Variable ausgegeben. Um die Variable mit dem korrekten Inhalt zu ersetzen, muss ein Paket die Sprachvariable mitliefern. Sprachvariablen werden in Sprachdateien gespeichert.

7.3. Aufbau von Sprachdateien

Für jede Sprache wird eine eigene Datei angelegt, die exemplarisch in Programm 7.2 zu sehen ist.

Programm 7.2 Exemplarischer Aufbau einer deutschen Sprachdatei

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE language SYSTEM "http://www.woltlab.com/DTDs/language.dtd">
<language languagecode="de">
  <category name="example.category.name">
    <item name="example.category.name.variable1"><![CDATA[Beispielinhalt]]></item>
    <item name="example.category.name.variable2"><![CDATA[Beispielinhalt]]></item>
    ...
  </category>
  ...
</language>
```

Anhand der ersten Zeile des Beispiels ist festzustellen, dass Sprachdateien im XML-Format und mit UTF-8-Kodierung angelegt werden. Das WCF unterstützt auch andere Zeichenkodierungen und wandelt die UTF-8-Zeichen dann bei Bedarf in eine andere Kodierung um. Das Wurzel-Element dieser XML-Datei ist `<language>`. Das Attribut

`languagecode` definiert einen Sprachcode zur Erkennung der jeweiligen Sprache. Als Namenskonvention wird der ISO 639-1-Standard[1] vorausgesetzt.

Innerhalb von `<language>` werden dann die Kategorien angelegt. Das Tag `<category>` enthält das Attribut `name`, das den Namen der Sprachkategorie angibt. Kindelemente von `<category>` sind die Sprachvariablen mit dem Tag `<item>`. Auch hier enthält `name` den Namen der Sprachvariablen.

Innerhalb von `item` wird der Inhalt der Sprachvariable in einem CDATA[2]-Block angegeben. Hier kann auch Templatescripting und HTML eingesetzt werden, um grammatikalische Probleme zu lösen, die Ausgabe von bestimmten Werten zu ermöglichen oder bestimmte Formatierungen vorzunehmen.

Warnung: Verwenden Sie kein Templatescripting in Variablen, die Sie direkt in einer PHP-Datei aufrufen: `$language->get("name.der.variable")`

Im Programm 7.3 ist eine Ausgabe zu sehen, die bei einer Umfrage benötigt wird. Wenn noch niemand abgestimmt hat, wird „Noch keine Stimme“ ausgegeben. Sofern es eine oder mehrere Stimmen gibt, wird der Text „Insgesamt x Stimmen“ ausgegeben. Anstelle von „x“ wird der aktuelle Zählerstand ausgegeben. Weiterhin ist auch zu sehen, wie eine zusätzliche `if`-Bedingung genutzt wird, um zwischen Einzahl und Mehrzahl des Wortes „Stimme“ zu unterscheiden.

Programm 7.3 Beispiel für Templatescripting in Sprachvariablen

```
<item name="wcf.poll.votes">
  <![CDATA[
    {if $poll->votes == 0}
      Noch keine Stimme
    {else}
      Insgesamt {#$poll->votes} Stimme{if $poll->votes > 1}n{/if}
    {/if}
  ]]>
</item>
```

Ausführliche Informationen über Templatescripting entnehmen Sie bitte Kapitel 6 auf Seite 27.

7.4. Sprachdateien einbinden

Nachdem Sie eine Sprachdatei erstellt haben, kann diese nun einem Paket zugeordnet werden. In der `package.xml` referenzieren Sie die Sprachdatei innerhalb des `<instructions>`-Blocks wie in Programm 7.4 auf der nächsten Seite zu sehen. Hier wird eine deutsche Sprachdatei mit dem Namen `de.xml` während der Installation eingebunden.

[1] http://de.wikipedia.org/wiki/ISO_639
[2] http://de.wikipedia.org/wiki/CDATA

Programm 7.4 Einbindung einer deutschen Sprachdatei in der `package.xml`

```
...
<instructions  type="install">
  ...
  <languages languagecode="de">de.xml</languages>
  ...
</instructions>
...
```

Weitere Informationen zum Thema Languages-PIP finden Sie in Kapitel 13.2.16 auf Seite 89.

8. Events

Das WCF bringt ein eigenes Event-System mit, um an elementaren Stellen den Eingriff in den Programmablauf zu ermöglichen. So können individuelle Veränderungen herbeigeführt werden, ohne die originalen Quellcode-Dateien modifizieren zu müssen. Events sind hierbei Ereignisse, die während des Programmablaufs passieren – z. B. wenn ein Formular abgeschickt wird. Mit einem eigenen `EventListener` können zu diesem Zeitpunkt eigene Methoden aufgerufen werden.

8.1. Ereignisse auslösen

Events werden über die statische Methode `fireAction()` der `EventHandler`-Klasse ausgelöst. Zwei Parameter werden der Methode übergeben:

`$eventObj` Das Objekt, wo das Ereignis ausgelöst wurde.

`$eventName` Der Name des Ereignisses.

Um nun das Ereignis nutzen zu können, muss man zunächst erst einmal wissen, wann das Ereignis ausgelöst wird. Eine Liste über alle Ereignisse im WCF finden Sie im Anhang Kapitel 15 auf Seite 102.

8.2. Ereignisse nutzen

Nun kann ein eigener `EventListener` geschrieben werden. Hierbei muss das gleichnamige Interface und dessen Methode `execute()` implementiert werden. Drei Parameter werden dabei vom System übergeben:

`$eventObj` Das Objekt, wo das Ereignis ausgelöst wurde.

`$className` Der Name der Klasse des Event-Objekts.

`$eventName` Der Name des Ereignisses.

Innerhalb der Methode können Sie nun definieren, was im Falle des Ereignisses passieren soll. Nachdem Sie nun einen `EventListener` geschrieben haben, müssen Sie diesen im System registrieren. Dazu nutzen Sie das EventListener-PIP, das in Kapitel 13.2.1 auf Seite 72 erläutert wird.

9. Sessions

Das Session[1]-System des WCF bietet eine einfache Möglichkeit, um Sitzungsdaten bereitzustellen und zu speichern.

Die Klasse `WCF` ruft bei jedem Seitenaufruf die Methode `initSession()` auf. Dabei wird von der `CookieSessionFactory` eine `Session`-Instanz geholt. Die Begriffe `SessionFactory` und `Session` sollen im Folgenden näher erläutert werden.

9.1. SessionFactory

Eine `SessionFactory` ist dazu da, eine `Session` anzulegen bzw. aus der Datenbank zu holen. Durch diese Klasse wird lediglich der Sessionlink unterstützt. Dies bedeutet, dass anhand der URL die `Session` geholt wird. Im gesamten Administrationsbereich findet nur aufgrund des Sessionlinks ein Abgleich der `Session` statt.

Die `CookieSessionFactory` erweitert die `SessionFactory` um die Möglichkeit, die aktive `Session` über ein Cookie zu identifizieren. Somit können Benutzer automatisch vom System erkannt werden, wenn sie nach Tagen zu der Seite zurückkehren.

Um eine eigene `SessionFactory` zu verwenden, muss die `initSession()`-Methode der `WCF`-Klasse überschrieben werden.

9.2. Session

Das `Session`-Objekt enthält alle wichtigen (Benutzer-)Daten einer Sitzung. Innerhalb des Objekts wird standardmäßig anhand der Browserkennung geprüft, ob es sich um den selben Benutzer handelt. Man kann auch eine Überprüfung der IP-Adresse vornehmen, allerdings ist diese Variante deshalb nicht standardmäßig aktiviert, da viele Provider wechselnde IP-Adressen vergeben.

Die `CookieSession` erweitert die `Session`-Klasse, um die Funktionalität, dass die Sessionkennung im Cookie abgespeichert wird. So können Besucher, die auf die Website zurückkehren automatisch erkannt werden. In der Endanwendung Burning Board wird die

[1] http://de.wikipedia.org/wiki/Sitzung_%28Informatik%29

`CookieSession` zusätzlich um eine eigene `WBBSession` erweitert, die spezielle Informationen bereitstellt, die nur vom Forensystem verwendet werden. So gibt es beispielsweise eine eigene `WBBUser-` sowie `WBBGuestSession`, da für Gäste im Forum einige Daten gar nicht vorhanden sind.

Um temporär Daten in einer Session verwenden zu können, sind die Sessionvariablen zu verwenden:

***void* register (*string* $key, *string* $data)** Verwenden Sie diese Methode um einen Wert (`$data`) einem Schlüssel (`$key`) zuzuweisen.

***void* unregister (*string* $key)** Um einen Wert aus der Session zu entfernen, rufen Sie die `unregister()`-Methode mit dem entsprechenden Schlüssel auf.

***array* getVars ()** Mit dieser Methode erhalten Sie alle Variablen, in einem assoziativen Array gespeichert, zurück.

***mixed* getVar (*string* $name)** Um eine einzelne Variable aus der Session zu erhalten, nutzen Sie bitte diese Methode und übergeben dabei den Schlüssel, mit dem die Variable gespeichert wurde.

10. Caching

Eine Besonderheit des WCF ist das mitgelieferte Caching-System. Hiermit kann die Datenbank entlastet werden. Bei anderen Webtechnologien wie JEE[1], gibt es den sogenannten Application Scope, in den Daten gelegt werden können, die über die gesamte Anwendung hinweg verfügbar sein müssen. Das sind häufig die Bestandteile einer Applikation, die selten geändert, aber häufig verwendet werden. Bei PHP-Anwendungen gibt es keinen Application Scope. Das WCF bietet daher ein eigenes Caching-System an, mit dem Daten zwischengespeichert werden können. Um das System zu nutzen, muss das Interface `CacheBuilder` und dessen Methode `getData()` implementiert werden. Diese Methode gibt einen beliebigen Wert zurück, der serialisiert in einer Datei auf dem Dateisystem abgespeichert wird. Hier wird die fehlende Typsicherheit von PHP ausgenutzt. Es kann ein beliebiger Typ zurückgegeben werden – ein Objekt, ein Array oder ein primitiver Datentyp. Innerhalb dieser Methode wird in der Regel eine komplexere Datenbankabfrage ausgeführt. In Programm 10.1 ist eine exemplarischer `CacheBuilder` abgebildet.

Programm 10.1 Beispiel für einen eigenen CacheBuilder

```
class CacheBuilderTest implements CacheBuilder {
  /**
   * @see CacheBuilder::getData()
   */
  public function getData($cacheResource) {
    $data = array();

    // get data from DB
    $sql = "SELECT  *
            FROM    test";
    $result = WCF::getDB()->sendQuery($sql);
    while ($row = WCF::getDB()->fetchArray($result)) {
      $data[]= $row;
    }
    return $data;
  }
}
```

Um einen erstellten `CacheBuilder` verwenden zu können, muss dieser im System registriert werden. Anschließend können die Daten aus dem Cache abgefragt werden, wie in Programm 10.2 auf der nächsten Seite zu sehen ist.

[1] http://de.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition

Programm 10.2 Beispiel für die Benutzung des CacheBuilders

```
WCF::getCache()->addResource(
  'test-'.PACKAGE_ID,
  WCF_DIR.'cache/cache.test-'.PACKAGE_ID.'.php',
  WCF_DIR.'lib/system/cache/CacheBuilderTest.class.php');

$this->testData = WCF::getCache()->get('test-'.PACKAGE_ID);
```

Über die statische Methode `getCache()` kann auf das globale Cache-Objekt des Systems zugegriffen werden.Zunächst wird eine Cache-Resource angelegt. Die Methode `addResource()` erwartet drei Parameter:

1. Name der Cache-Resource – hier sollte mit der Paket-ID der Endanwendung gearbeitet werden.
2. Pfad zur Cache-Datei, die auf dem Dateisystem abgelegt wird – in diesem Beispiel wird im Cache-Ordner des WCF-Verzeichnis eine Datei abgelegt.
3. Pfad zur `CacheBuilder`-Klasse.

In der zweiten Anweisung des Code-Beispiels werden über die Methode `get("Name der Cache-Resource")` die gespeicherten Daten aus der Cache-Datei geholt.

Um einen Cache neu anzulegen bzw. zurückzusetzen muss die Cache-Datei gelöscht werden. Beim nächsten Aufruf wird der Cache automatisch neu erstellt. Mit folgendem Code wird eine Cache-Resource gelöscht:

```
WCF::getCache()->clearResource('Name des Caches');
WCF::getCache()->clear(WCF_DIR . 'cache/', 'cache.Name.php');
```

11. RequestHandler & die Page-, Form- und Action-Klassen

Internet-Anwendungen werden über Anfragen an den Server gesteuert. Hierbei bietet der `RequestHandler` eine einheitliche Schnittstelle zur Verarbeitung und Steuerung von HTTP-Requests.

11.1. RequestHandler

Der `RequestHandler` kann über seine statische Methode `handle()` genutzt werden. Diese erwartet als einzigen Parameter ein Array mit Pfadangaben zu den Verzeichnissen der jeweiligen Bibliotheken. Gemeint ist hier das `lib`-Verzeichnis des WCF und der jeweiligen Endanwendung. Beim Ausführen der Methode überprüft das System den HTTP-Request nach POST- bzw. GET-Variablen und öffnet bzw. verarbeitet die entsprechende Seite.

Es gibt drei verschiedene Typen von Seiten:

Page Einfache Seiten im WCF, die zur Auflistung von Informationen benötigt werden

Form Spezielle Formularseiten, in denen Daten eingegeben und verarbeitet werden

Action Zum Ausführen von Aktionen, ohne Anzeige einer resultierenden Informations- oder Formularseite

Eine mögliche URL wäre z. B. „`.../index.php?page=Test`“. Der `RequestHandler` wird nun versuchen ein Objekt der Klasse `TestPage` zu erstellen. Beim Link „`.../index.php?form=Test`“ würde nach der Klasse `TestForm` gesucht werden. Innerhalb dieser Klasse können dann weitere Parameter verarbeitet werden.

Das WCF bietet für diese Seitentypen jeweils ein Interface und eine Standard-Implementierung innerhalb einer abstrakten Klasse an.

Bemerkung: Es ist wichtig, dass Sie sich bei der Benennung Ihrer Klassen an das im WCF verwendete Namensschema halten: Klassennamen müssen mit einem Großbuchstaben beginnen, gefolgt vom Typ der Klasse (Page, Form oder Action).

11.2. Page und AbstractPage

Das Page-Interface definiert folgende Methoden:

readParameters() Die POST- und GET-Parameter werden ausgelesen.

readData() Daten, die zur Darstellung der Seite benötigt werden, sind auszulesen bzw. zusammenzustellen.

assignVariables() Variablen, die im Template verwendet werden sollen, sind an die Template-Engine zu übergeben.

show() Die Seite wird dargestellt.

`AbstractPage` ist die Standard-Implementierung des `Page`-Interfaces. Bei allen vier Methoden wird das entsprechende Event ausgelöst. Beim Ableiten dieser Klasse ist zu beachten, dass beim Überschreiben von Methoden entweder das Event selbst ausgelöst wird oder die Eltern-Methode aufgerufen wird, wie im Beispiel-Programm 11.1 zu sehen ist. Des weiteren werden dort zwei Variablen der Template-Engine übergeben. Diese können dann im Template `test.tpl` verwendet werden.

Programm 11.1 Ableitung von AbstractPage

```
class TestPage extends AbstractPage {

  public $templateName = 'test';

  /**
   * @see Page::assignVariables()
   */
  public function assignVariables() {
    parent::assignVariables();

    WCF::getTPL()->assign(array(
      'test' => "Erste Testvariable",
      'test2' => "Zweite Testvariable"
    ));
  }
}
```

Die Klasse `AbstractPage` ist ein guter Start zur Realisierung einfacher Seiten. Es empfiehlt sich vor der Benutzung der Klasse, diese noch mal genauer anzusehen. Weitere abstrakte Klassen wie `MultipleLinkPage` und `SortablePage` können für komplexere Seiten verwendet werden.

11.3. Form und AbstractForm

Das `Form`-Interface erweitert das `Page`-Interface um folgende Methoden:

submit() Beim Absenden des Formulars wird diese Methode aufgerufen.

validate() Fomulareingaben können validiert werden.

save() Diese Methode ist zum Abspeichern der Formulardaten zu nutzen.

readFormParameters() Zum Auslesen der Formulareingaben.

Auch hier gibt es eine abstrakte Klasse `AbstractForm`, die für Formularseiten benutzt werden sollte.

11.4. Action und AbstractAction

Um Benutzereingaben zu Verarbeiten ohne eine eigene Resultats- oder Formularseite anzuzeigen, sollte das Interface `Action` verwendet werden. Folgende Methoden werden definiert:

readParameters() Beim Absenden des Formulars wird diese Methode aufgerufen.

execute() Die Aktion wird durchgeführt.

Die abstrakte Klasse `AbstractAction` definiert darüber hinaus die Methode `executed()`, für Anweisungen die nach einer Aktion durchgeführt werden sollen. Diese muss aber manuell aufgerufen werden, wohingegen die Methoden `readParameters()` und `execute()` beim Erstellen einer Instanz automatisch aufgerufen werden.

Teil II.

Pakete erstellen

12. WCF-Pakete

In diesem Kapitel finden Sie Informationen, wie Sie ein eigenes Paket im WCF erstellen können. Dabei wird auf das Format eingegangen, auf die Besonderheiten der `package.xml`-Datei und die verschiedenen Pakettypen.

12.1. Das Format

Pakete sind `.tar` (oder `.tar.gz`) Archive, die stets eine XML Datei namens `package.xml` enthalten und je nach Paket auch noch weitere Dateien.

```
+ paket.tar
  | ...
  | package.xml
  | ...
  | ...
```

Empfohlene Software:

- tar & gzip (oder z. B. 7Zip)
- ein XML–Editor

12.2. Die `package.xml` Datei

Die `package.xml`-Datei muss sich im `.tar`-Archiv im Hauptordner befinden – die Datei in einen Unterordner zu verschieben ist nicht möglich.

Die `package.xml` enthält Meta-Informationen über das Paket sowie Anweisungen für die Installation und das Update von früheren Versionen. Die Tabelle 12.1 auf Seite 65 zeigt alle erforderlichen und möglichen XML-Tags in der `package.xml`-Datei.

12.2.1. Der Paketbezeichner

Der Paketbezeichner ist ein eindeutiger Name, der zur Identifizierung des Paketes benutzt wird. Das WCF orientiert sich dabei an den Java Paketnamen. Zur Überprüfung des Paketbezeichners wird die Methode `Package::isValidPackageName()` verwendet.

Wichtig:

Ein gültiger Paketbezeichner setzt sich aus mindestens drei Teilen zusammen, die jeweils durch einen Punkt (.) getrennt werden müssen. Jeder Teil muss aus mindestens einem alphanumerischen Zeichen[1] oder dem Unterstrich (_) bestehen. Man kann den Namen auch in mehr als drei Teile aufteilen: z. B.: 'com.woltlab.wcf'

12.2.2. Mehrsprachige Paketnamen und Paketbeschreibungen

Um ein Paket in mehreren Sprachen in einer einzigen Datei auszuliefern, kann man den Paketnamen sowie die Beschreibung des Paketes in der `package.xml`-Datei für mehrere Sprachen definieren. Für die Tags `<packagename>` und `<packagedescription>` gibt es deshalb den Parameter `language`, der den Sprachcode nach dem ISO 639-1-Standard[2] beeinhaltet.

Programm 12.1 Beispiel für mehrsprachige Paketnamen

```
<package name="com.woltlab.wcf.data.page.legalNotice">
  <packageinformation>
    <packagename>Legal Notice Page</packagename>
    <packagedescription></packagedescription>
    <packagename language="de">Impressum</packagename>
    <packagedescription language="de">Kommerzielle deutsche Internetseiten müssen nach
    Paragraph 5 Telemediengesetz über ein Impressum verfügen.</packagedescription>
    ...
  </packageinformation>
  ...
</package>
```

12.2.3. `<requiredpackage>` Tag

Um die Abhängigkeiten eines Paketes zu anderen Paketen zu kennzeichen, gibt es den Tag `<requiredpackage>`. Dort wird auf den Paketbezeichner des benötigten Paketes verwiesen. Dieses Paket muss installiert sein, bevor das aktuelle Paket installiert werden kann.

Über den optionalen Parameter `minversion` kann man eine minimale Version des Paketes definieren, die mindestens installiert sein muss.

[1] a bis z, A bis Z oder 0 bis 9

[2] http://de.wikipedia.org/wiki/ISO_639

Tag	Parameter	Bedeutung
`<package>`	ja, s. 12.2.1	
· `<packageinformation>`	–	Enthält Meta-Informationen über das Paket
· · `<packagename>`	ja, s. 12.2.2	Name des Paketes
· · `<packagedescription>`*[a]	ja, s. 12.2.2	kurze Beschreibung des Paketes
· · `<version>`	–	Versionsnummer des Paketes[b]
· · `<date>`*	–	Datum der Veröffentlichung dieser Version[c]
· · `<isunique>`*	–	Kann dieses Paket nur einmal installiert werden (0 oder 1)?
· · `<standalone>`*	–	Ist dieses Paket eine Endanwendung[d] (0 oder 1)?
· · `<plugin>`*	–	Name des übergeordneten Paketes
· · `<packageurl>`*	–	Webseite des Paketes mit weiteren Informationen
· `<authorinformation>`*	–	Enthält Meta-Informationen über den Paketersteller
· · `<author>`*	–	Namen des Herstellers
· · `<authorurl>`*	–	Webseite des Herstellers
· `<requiredpackages>`*	–	Enthält eine Liste von Paketen, die von diesem Paket benötigt werden.
· · `<requiredpackage>`*	Ja, s. 12.2.3	Name des benötigten Paketes[e]
· `<optionalpackages>`*	–	Enthält eine Liste von Paketen, die mit diesem Paket optional installiert werden können.
· · `<optionalpackage>`*	Ja, s. 12.2.4	Name des optionalen Paketes.
· `<instructions>`*	Ja, s. 12.2.5	Enthält eine Liste Anweisungen für die Installation oder das Update des Paketes. Zwischen Installation und Update wird über Parameter (s. 12.2.5) unterschieden. Innerhalb des `<instructions>` Tag werden die sog. PackageInstallationPlugins aufgerufen.

[a] mit einem Stern (*) markierte Tags sind optional

[b] Es wird empfohlen, dreigeteilte Versionsnummern zu benutzen: X.Y.Z. Angehängte Zusätze wie „Alpha N“, „Beta N“, „RC N“ sind dabei möglich. Siehe auch PHP Funktion `version_compare`, http://www.php.net/manual/de/function.version-compare.php

[c] im englischen Format, mit der PHP Funktion `strtotime` (http://www.php.net/manual/de/function.strtotime.php) verträglich, z. B. YYYY-MM-DD

[d] siehe 12.3 über Pakettypen

[e] die optionale Angabe einer minimalen Version und den Verweis auf die mitgelieferte Paketdatei sind über Parameter möglich, siehe dazu 12.2.3.

Tabelle 12.1.: Tags in der `package.xml` Datei

Damit die Installation nicht abbricht, wenn man ein benötigtes Paket noch nicht installiert hat, kann man den Ort der Paketdatei auch gleich mit dem `file` Parameter definieren. Dabei kann es sich um eine URL (http oder ftp) handeln oder um einen relativen Pfad, der auf eine Paketdatei in der aktuellen Paketdatei verweist. Das benötigte Paket wird dann automatisch heruntergeladen bzw. aus der aktuellen Paketdatei entpackt und installiert.

In Tabelle 12.2 sehen Sie konkrete Beispiele.

Programm 12.2 Auflistung der benötigten Pakete

```
<requiredpackages>
  <requiredpackage minversion="1.0.0" file="requirements/com.woltlab.wcf.tar">
    com.woltlab.wcf
  </requiredpackage>
  <requiredpackage minversion="1.0.0" file="requirements/com.woltlab.wcf.data.page.tar">
    com.woltlab.wcf.data.page
  </requiredpackage>
</requiredpackages>
...
<requiredpackages>
  <requiredpackage minversion="1.0.0" file="http://server.com/paket.tar.gz">
    paket
  </requiredpackage>
</requiredpackages>
```

12.2.4. `<optionalpackage>` Tag

Analog zum <requiredpackage>-Tag, siehe 12.2.3

12.2.5. Installations- und Updateinstruktionen

Der Tag `<instructions>` enthält eine Liste von Instruktionen für die Installation oder das Update eines Paketes. Über den Parameter `type` unterscheidet man zwischen Installation und Update: `type='install'` bzw. `type='update'`.

Im Falle eines Updates muss man den erforderlichen Parameter `fromversion` benutzen, um die Versionen festzulegen, die aktualisiert werden können. In der `fromversion` Angabe sind dabei auch Platzhalter (*) erlaubt, um eine ganze Reihe von Versionen zu erlauben: z. B. `fromversion='3.0.0 RC 1'`, `fromversion='1.1.*'`, `fromversion='*'`

Innerhalb des `<instructions>`-Tags wird dann auf die einzelnen PIPs[3] verwiesen. Es gibt PIPs zum Installieren von Dateien (`<files>files.tar</files>`), zum Installieren von Templates (`<templates>templates.tar</templates>`) oder zum Ausführen von SQL-Anweisungen (`<sql>install.sql</sql>`) und noch viele mehr. Mehr zu PIPs erfahren Sie im folgenden Kapitel.

[3] PackageInstallationsPlugins

12.3. Verschiedene Pakettypen

Das Paketsystem unterscheidet zwischen drei Arten von Paketen.

Paket Der Standardtyp. Dateien dieser Pakete werden ins WCF-Verzeichnis installiert. Ein Paket stellt in der Regel Programmbibliotheken zur Verfügung, die dann von Endanwendungen verwendet werden können.

Endanwendung Eine Endanwendung ist eine eigenständige Applikation auf Basis des WCF. Bei der Installation muss der Benutzer einen eigenen Installationsordner definieren.

Plugin Ein Plugin ist ein Paket, das an ein existierendes Paket gebunden ist und Zusatzfunktionen oder Erweiterungen zu diesem Paket mitliefert. Bei der Erstellung eines Plugins reicht es, in der package.xml per `<plugin>` Tag den Namen des zugehörigen Pakets anzugeben. Vergleiche Tabelle 12.1 auf Seite 65.

13. Package Installation Plugin

Ein Package Installation Plugin, kurz PIP, ist ein Plugin, das den Installationsvorgang von Paketen durch neue Funktionen erweitern kann. Ein PIP kann dabei z. B. auf das Paketarchiv zugreifen, Dateien extrahieren, XML Dateien parsen und Daten in die Datenbank schreiben. Die Möglichkeiten sind vielfältig. Das PIP ist dabei sowohl für die Installation als auch für die Deinstallation seiner Daten zuständig (dafür muss das PIP die vorgenommenen Änderungen ggf. „loggen“).

Ein PIP wird auf eine entsprechende PHP-Klasse abgebildet, die das Interface `PackageInstallationPlugin` implementiert. Innerhalb des `<instruction>`-Blocks der `package.xml` wird ein PIP über einen XML-Code der Form `<pip>datei</pip>` aufgerufen. Es findet also immer ein Verweis auf eine Datei statt, mit der das entsprechende PIP dann etwas machen soll. Wenn die Datei nicht im Hauptverzeichnis des Pakets liegt, sollte der Pfad relativ angegeben werden.

Nachfolgend werden alle PIPs vorgestellt, die über das WCF oder die freien Pakete mitgeliefert werden. Zur besseren Übersicht wurden sie in drei Kategorien eingeteilt: Datei-basierte PIPs, Import-PIPs (XML) sowie Sonstige PIPs. Im Anschluss daran wird auch erklärt, wie man eigene PIPs realisieren kann.

13.1. Datei-basierte PIPs

Bei den Datei-basierten PIPs findet immer eine Installation von Dateien statt, die in einem Tar-Archiv gespeichert sind.

13.1.1. Das Files-PIP

Verwendungszweck

Das Files-PIP wird zum Installieren von Dateien verwendet.

XML-Code in der `package.xml`

```
<files>files.tar</files>
```

Funktionsweise

Die Ordnerstruktur innerhalb des Archivs wird komplett auf das Installationsverzeichnis übertragen. Dateien von Endanwendungen oder deren Plugins werden in das Verzeichnis der Endanwendung kopiert. Bei anderen Paketen ist das WCF-Verzeichnis das Ziel. Bei jeder Datei wird die Installation in der Datenbank festgehalten. In der Tabelle `wcf_package_installation_file_log` wird der Name der Datei und die ID des Pakets gespeichert.

Bemerkung: Sollte im Zielverzeichnis bereits eine Datei mit gleichem Namen existieren, so gibt es zwei Möglichkeiten:

1. Die Datei wurde von einem anderen Paket installiert:
   In diesem Fall bricht die Installation ab. Ein Paket kann keine Dateien von anderen Paketen überschreiben.

2. Die Datei ist dem System unbekannt:
   Der Benutzer erhält eine Sicherheitsabfrage, ob die Datei überschrieben werden soll. Vermutlich wurde die Datei vorher manuell hinein kopiert.

13.1.2. Das Templates-PIP

Verwendungszweck

Das Templates-PIP wird zum Installieren von Templates genutzt.

XML-Code in der `package.xml`

```
<templates>templates.tar</templates>
```

Funktionsweise

Während der Paketinstallation wird die Archiv-Datei entpackt und die Inhalte in den vorgesehenen Templateordner kopiert. Eine Endanwendung hat ihren eigenen Templateordner, andere Pakettypen nutzen den Templateordner des WCF. Wenn Templatenamen doppelt vorkommen, werden die Templatenamen um die Paket-ID erweitert. Zusätzlich werden alle Templatenamen in der Datenbank zusammen mit der Paket-ID gespeichert.

Bemerkung: Es gilt wie beim Files-PIP, dass Templatenamen nur einmal vorkommen dürfen.

13.1.3. ACPTemplates-PIP

Verwendungszweck

Das ACPTemplates-PIP wird zum Installieren von Templates der Administrationsoberfläche genutzt.

XML-Code in der `package.xml`

```
<acptemplates>acptemplates.tar</acptemplates>
```

Funktionsweise

Siehe Templates-PIP 13.1.2 auf der vorherigen Seite.

13.1.4. Das Style-PIP

Verwendungszweck

Mit Hilfe des Style-PIPs ist es möglich einen Stil mit einem Paket zu installieren.

XML-Code in der `package.xml`

`<style>style.tar</style>` Über den optionalen Parameter `default="true"` kann dieser Stil auch gleich während der Installation als Standard gesetzt werden.

Funktionsweise

Die Archivdatei muss zwingend eine Datei namens `style.xml` enthalten. Vergleichbar mit der `package.xml` bei Paketen enthält sie alle wichtigen Informationen über einen Stil. Ein exemplarischer Aufbau dieser Datei ist in Code-Beispiel 13.1 auf der nächsten Seite zu sehen.

Während im `<general>`- und `<author>`-Block lediglich Meta-Informationen angegeben werden, sind im `<files>`-Block wichtige Verweise auf Dateien zu finden, die während der Installation des Stils verarbeitet werden. Das Archiv `images.tar` enthält alle Bilddateien, die von dem Stil benötigt werden. In der `variables.xml` werden alle Werte der Stilvariablen festgelegt.

Bemerkung: Alle Stilvariablen und deren mögliche Werte finden Sie im Anhang in Kapitel 16 auf Seite 104.

Programm 13.1 Exemplarischer Aufbau einer `style.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE style SYSTEM "http://www.woltlab.com/DTDs/SXF/style.dtd">
<style>
  <general>
    <stylename><![CDATA[WoltLab Blue Sunrise]]></stylename>
    <description><![CDATA[The WoltLab Burning Board 3 default style.]]></description>
    <version><![CDATA[1.0.0]]></version>
    <date><![CDATA[2007-07-17]]></date>
    <image><![CDATA[WoltLab Blue Sunrise.png]]></image>
    <copyright><![CDATA[WoltLab GmbH]]></copyright>
    <license><![CDATA[Commercial]]></license>
  </general>
  <author>
    <authorname><![CDATA[Arian Glander, Harald Szekely]]></authorname>
    <authorurl><![CDATA[http://www.woltlab.com/]]></authorurl>
  </author>
  <files>
    <variables>variables.xml</variables>
    <images>images.tar</images>
  </files>
</style>
```

13.1.5. Das PIPs-PIP

Verwendungszweck

Das PackageInstallationPlugins-PIP wird zum Installieren von PIPs benötigt. So ist das System erweiterbar um neue PIPs.

XML-Code in der `package.xml`

```
<packageinstallationplugins>pip.tar</packageinstallationplugins>
```

Funktionsweise

Während der Installation wird die Archivdatei entpackt und die einzelnen PIPs in den Ordner `wcf/lib/acp/package/plugin` kopiert. Zusätzlich werden die PIPs in der Datenbank registriert, zusammen mit der Paket-ID.

13.2. Import PIPs (XML)

Die hier aufgeführten PIPs haben alle eine Gemeinsamkeit: Alle bauen auf XML-Dateien, deren Informationen in die Datenbank eingetragen – also importiert – werden sollen. Der Aufbau einer solchen XML-Datei ist häufig sehr ähnlich und ist exemplarisch in Programm 13.2 auf der nächsten Seite abgebildet.

Programm 13.2 Exemplarischer Aufbau einer Import PIP XML-Datei

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/custom.dtd">
<data>
  <import>
    <custom>
    ...
    </custom>
    ...
  </import>
</data>
```

Nach dem startenden XML-Tag folgt die Angabe des Doctypes mit dem Verweis auf die entsprechende DTD. Das Wurzeltag `<data>` umfasst alle anderen Tags. Darin folgt ein `<import>`-Tag, welches die einzelnen zu importierenden Elemente enthält. Diese sind von PIP zu PIP unterschiedlich. Bei einigen PIPs kann nach einem `<import>`-Tag auch noch ein `<delete>`-Tag angegeben werden, um bestimmte Daten wieder zu löschen. Darauf wird dann aber beim entsprechenden PIP noch hingewiesen.

13.2.1. Das EventListener-PIP

Verwendungszweck

Das EventListener-PIP wird verwendet, um `EventListener` im System zu registrieren.

XML-Code in der `package.xml`

```
<eventlistener>eventlistener.xml</eventlistener>
```

Tags und deren Bedeutung

<eventlistener> Definiert den zu installierenden `EventListener`

- **<eventclassname>** Name der Klasse, die das Event auslöst
- **<eventname>** Name des Events
- **<listenerclassfile>** Relative Pfadangabe zur `EventListener`-Klasse. Nach Konvention werden sie im Ordner `lib/system/event/listener` abgelegt. Zum Installieren dieser Datei verwenden Sie das Files-PIP.
- **<environment>** Wenn das Event innerhalb des Administrationsbereichs auftritt wird hier „admin“ angegeben. Ansonsten geben Sie „user“ an. Dies ist der Standardwert, daher kann das Tag in diesem Fall auch weggelassen werden.

· **<inherit>** Wenn der `EventListener` auch bei Klassen aufgerufen werden soll, die sich von der Klasse ableiten, die im Tag <eventclassname> angegeben ist, bitte hier den Wert 1 angeben. Ansonsten 0 oder das Tag einfach weglassen. *Bemerkung:* Die Vererbung verbraucht zusätzliche Rechenzeit. Bitte setzen Sie diese Funktion nur dort ein, wo es nötig ist.

Code-Beispiel

Programm 13.3 Exemplarischer Aufbau einer `eventlistener.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/eventListeners.dtd">
<data>
  <import>
    <eventlistener>
      <eventClassName>URLParser</eventClassName>
      <eventName>didParse</eventName>
      <listenerClassFile>
        lib/system/event/listener/URLParserThreadURLListener.class.php
      </listenerClassFile>
    </eventlistener>

    <!-- admin -->
    <eventlistener>
      <eventClassName>UserListPage</eventClassName>
      <eventName>assignVariables</eventName>
      <environment>admin</environment>
      <listenerClassFile>
        lib/system/event/listener/UserListPagePermissionsButtonListener.class.php
      </listenerClassFile>
    </eventlistener>

    ...
  </import>
</data>
```

13.2.2. Das Cronjobs-PIP

Verwendungszweck

Das Cronjobs-PIP wird dazu verwendet, um Cronjobs im System zu registrieren. Cronjobs sind aus der Unix-Welt bekannte zeitgesteuerte Aufgaben. Echte Cronjobs stehen meistens nicht zur Verfügung, da dafür ein eigener Server benötigt wird. Daher wurde die Funktionalität von Cronjobs mit AJAX und PHP nachempfunden. Durch diese Nachbildung können regelmäßige Aufgaben dennoch durchgeführt werden. Zu solchen Aufgaben könnte die Aktualisierung von Anzeigen zählen. Häufig werden bestimmte Anzeigen zwischengespeichert (Caching), da sie oft von den Benutzern angefordert werden und die Datenbank entlastet werden soll. Durch einen Cronjob kann so ein Zwischenspeicher beispielsweise einmal täglich aktualisiert werden.

XML-Code in der `package.xml`

```
<cronjobs>cronjobs.xml</cronjobs>
```

Funktionsweise

Ein eigener Cronjob kann geschrieben werden, in dem dass `Cronjob`-Interface und dessen `execute()`-Methode implementiert wird. Innerhalb der Methode können beliebige Aktionen angestoßen werden. Um die erstellte Cronjob-Klasse zu verwenden, muss sie über den Administrationsbereich des WCF eingetragen werden oder über das hier beschriebene PIP installiert werden. Das System verwaltet alle eingetragenen Cronjobs und führt den entsprechenden Cronjob selbstständig aus, sofern er an der Reihe ist. Hierbei können auch komplexe Operationen durchgeführt werden. Der Benutzer, der mit seinem Seitenaufruf den Cronjob anstößt, bekommt davon nichts mit, da hier durch Einsatz von AJAX die gesamte Operation asynchron im Hintergrund abläuft.

Tags und deren Bedeutung

<cronjob> Enthält die nachfolgend beschriebenen Tags.

- **<classpath>** Pfadangabe zu der PHP-Datei, die die entsprechende auszuführende Klasse enthält. Die Angabe muss relativ zum Installationsverzeichnis der ausgewählten Endanwendung sein.
- **<description>** Eine kurze Beschreibung zur Aufgabe dieses Cronjobs.
- **<startminute>** Zu diesen Minuten (0 - 59) soll die Aufgabe ausgeführt werden, sonst bitte das *-Symbol angeben.
- **<starthour>** Zu diesen Stunden (0 - 23) soll die Aufgabe ausgeführt werden, sonst bitte das *-Symbol angeben.
- **<startdom>** An diesen Tagen des Monats (1 - 31) soll die Aufgabe ausgeführt werden, sonst bitte das *-Symbol angeben.
- **<startmonth>** In diesen Monaten (1 - 12 oder jan - dec) soll die Aufgabe ausgeführt werden, sonst bitte das *-Symbol angeben.
- **<startdow>** An diesen Tagen der Woche (0 - 6 mit Sonntag = 0 oder mon - sun) soll die Aufgabe ausgeführt werden, sonst bitte das *-Symbol angeben.
- **<execmultiple>** Mehrfachausführung (0 - nein, 1 - ja): Durch Aktivieren dieser Option wird die Aufgabe mehrfach ausgeführt, wenn zwischen dem letzten Ausführungszeitpunkt und dem aktuellen Zeitpunkt weitere Ausführungszeitpunkte liegen.

- **<canbeedited>** Kann dieser Cronjob später über das ACP editiert werden (0 - nein, 1 - ja)
- **<canbedisabled>** Kann dieser Cronjob später über das ACP deaktiviert werden (0 - nein, 1 - ja)

Code-Beispiel

Programm 13.4 Exemplarischer Aufbau einer `cronjobs.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/cronjobs.dtd">
<data>
  <import>
    <cronjob>
      <classpath>lib/system/cronjob/CleanupCronjob.class.php</classpath>
      <description>Hourly Cleanup</description>
      <startminute>0</startminute>
      <starthour>*</starthour>
      <startdom>*</startdom>
      <startmonth>*</startmonth>
      <startdow>*</startdow>
      <execmultiple>0</execmultiple>
      <canbeedited>0</canbeedited>
      <canbedisabled>0</canbedisabled>
    </cronjob>
    ...
  </import>
</data>
```

13.2.3. Das Options-PIP

Verwendungszweck

Das Options-PIP wird dazu genutzt um innerhalb der Administrationsoberfläche Einstellungsmöglichkeiten zu definieren.

XML-Code in der `package.xml`

```
<options>options.xml</options>
```

Tags und deren Bedeutung

<categories> Enthält eine Liste von Kategorien

- **<category>** Die Kategorie wird verwendet, um Einstellungen zu sortieren. Das Tag enthält das Attribut `name`. Darin wird der Name der Kategorie angegeben.

· · **<parent>** Angabe des Namens der Elternkategorie.

· · **<showorder>** Angabe einer Zahl, die die Position der Kategorie bestimmt im Vergleich zu anderen Kategorien der gleichen Ebene.

<options> Enthält eine Liste von Einstellungen

· **<option>** Das Tag enthält das Attribut `name`. Darin wird der Name der Einstellung angegeben.

· · **<categoryname>** Gibt den Namen der Kategorie an, zu der diese Einstellung zugeordnet werden soll.

· · **<optiontype>** Bezeichnet den Optionstypen. Eine Liste mit den verfügbaren Optionstypen finden Sie im Anhang. Es ist auch möglich eigene Optionstypen zu schreiben.

· · **<showorder>** Position der Einstellung innerhalb der genannten Kategorie.

· · **<defaultvalue>** Vorgabewert der Einstellung

· · **<hidden>** Mit `<hidden>1</hidden>` können Einstellungen versteckt werden. Diese werden z. B. von der Installations angelegt (z. B. Datum der Installation), sollen aber zum einen nicht mit angezeigt werden und zum anderen auch nicht verändert werden können.

· · **<selectoptions>** Wenn als Optionstyp `radiobuttons` oder ein `select`-Typ gewählt wurde, können über dieses Tag mögliche Optionen angegeben werden. In jeder Zeile steht eine Option, die folgendermaßen aufgebaut ist:
`Wert:Name der Sprachvariable`
Verwenden Sie keine Leerzeichen oder Tabulatoren, um die einzelnen Optionen zu formatieren, ein einfacher Zeilenumbruch ist notwendig.

· · **<enableoptions>** Wenn als Optionstyp `boolean` verwendet wird und man eine Verbindung zu anderen Optionen erstellen möchte, wird `<enableoptions>` verwendet. Wenn also diese Option aktiviert ist, dann werden auch die unter `<enableoptions>` aufgelisteten Optionen sichtbar. Es sind die Namen der jeweiligen Optionen anzugeben und per Komma zu trennen.

· · **<validationpattern>** Angabe eines regulären Ausrucks, der zur Validierung dieser Option genutzt werden soll.

Beim Options-PIP kann neben dem `<import>`-Block auf der `<delete>`-Block verwendet werden, um bestimmte Optionen wieder aus dem System zu entfernen.

Programm 13.5 Exemplarischer Aufbau einer `options.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/options.dtd">
<data>
  <import>
    <categories>
      <category name="offline"></category>
      <category name="offline.general">
        <parent>offline</parent>
      </category>
      ...
    </categories>
    ...
    <options>

      <option name="page_title">
        <categoryname>general.page</categoryname>
        <optiontype>text</optiontype>
        <showorder>1</showorder>
        <defaultvalue>WoltLab Burning Board</defaultvalue>
      </option>

      <option name="board_default_days_prune">
        <categoryname>board.threads</categoryname>
        <optiontype>select</optiontype>
        <defaultvalue>1000</defaultvalue>
        <selectoptions><![CDATA[
          1:wbb.board.filterByDate.1
          3:wbb.board.filterByDate.3
          7:wbb.board.filterByDate.7
        ]]></selectoptions>
      </option>
      ...
    </options>
  </import>
  <delete>
    <option name="page_title"/>
  </delete>
</data>
```

Code-Beispiel

13.2.4. Das UserOptions-PIP

Verwendungszweck

Das UserOptions-PIP ermöglicht das Anlegen von Einstellungen für registrierte Benutzer. Während beim Options-PIP die Einstellungen später für den Administrator in der entsprechenden Oberfläche sichtbar sind, werden beim UserOptions-PIP die Einstellungen definiert, die ein Mitglied in seinem Profil vornehmen kann.

XML-Code in der `package.xml`

```
<useroptions>useroptions.xml</useroptions>
```

Tags und deren Bedeutung

Innerhalb dieser XML-Datei können alle Tags benutzt werden, die auch beim Options-PIP erklärt wurden. Zusätzlich kommen folgende Tags hinzu:

<categories> siehe Options-PIP

· **<category>** siehe Options-PIP

· · **<menuicon>** Pfadangabe zu einer Icon-Datei (möglichst S-Größe), die beim Menüeintrag verwendet werden soll.

· · **<icon>** Pfadangabe zu einem Icon, welches zur optischen Erkennung der Kategorie genutzt werden soll (möglichst M-Größe).

<options> siehe Options-PIP

· **<option>** siehe Options-PIP

· · **<outputclass>** Angabe des Namens einer Klasse, die zur späteren Ausgabe dieser Option verwendet werden soll.

· · **<searchable>** Mit `<searchable>1</searchable>` kann bestimmt werden, ob diese Angabe auch in der Mitgliedersuche zu durchsuchen ist.

· · **<visible>** Angabe einer Gruppen-ID einer Benutzergruppe, die diese Option standardmäßig sehen kann. Für andere Gruppen ist sie versteckt.

· · **<editable>** Angabe einer Gruppen-ID einer Benutzergruppe, die diese Option standardmäßig bearbeiten kann.

13.2.5. Das GroupOptions-PIP

Verwendungszweck

Das GroupOptions-PIP wird eingesetzt, um die Benutzerrechte im System anzulegen. Diese können dann später für die einzelnen Benutzergruppen eingestellt werden. Auch dieses PIP ist mit dem Options-PIP stark vergleichbar.

XML-Code in der `package.xml`

```
<groupoptions>groupotions.xml</groupoptions>
```

Tags und deren Bedeutung

Es können die gleichen Angaben wie innerhalb der XML-Datei für das Options-PIP gemacht werden.

13.2.6. Das FeedReaderSource-PIP

Verwendungszweck

Das FeedReaderSource-PIP wird dafür genutzt, Quellen von RSS-Feeds im System einzutragen.

XML-Code in der `package.xml`

```
<feedsource>feedsource.xml</feedsource>
```

Tags und deren Bedeutung

<feedsource> enthält das Attribut `name`, das den Namen des Feeds definiert.

- **<url>** gibt die URL zum Feed an. Im Beispiel ist zu sehen, dass hier ein String-Platzhalter (`%s`) verwendet wird. Dieser wird vom System automatisch durch den Sprachcode der vom Benutzer verwendeten Sprache ersetzt. Sie müssen diesen Platzhalter nicht verwenden. Sofern Sie ihn nutzen, sollten Sie aber auch dafür sorgen, dass die entsprechende XML-Datei existiert.
- **<cycle>** gibt an, wie oft die Informationen des Feeds überprüft und eingelesen werden sollen. Anzugeben ist die Anzahl der Sekunden, nach der die Quelle erneut gelesen werden soll (im Beispiel also nach einem Tag)

Code-Beispiel

Programm 13.6 Exemplarischer Aufbau einer `feedsource.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/feedsource.dtd">
<data>
  <import>
    <feedsource name="woltlab-news">
      <url>http://www.woltlab.com/rss_%s.xml</url>
      <cycle>86400</cycle>
    </feedsource>
    ...
  </import>
</data>
```

13.2.7. Das Help-PIP

Verwendungszweck

Das Help-PIP wird genutzt, um eigene Texte in der Hilfe anzulegen. Ein Hilfeelement ist vergleichbar mit einem Punkt im Inhaltsverzeichnis eines Buches.

XML-Code in der `package.xml`

```
<help>help.xml</help>
```

Tags und deren Bedeutung

<helpitem> hat das Attribut `name`, um ein Hilfeelement zu bezeichnen.

- **<showorder>** erwartet eine Nummer, um die Elemente untereinander zu sortieren.
- **<parent>** Angabe des Namens des übergeordneten Elements. Dadurch kann eine hierarchische Struktur der Hilfeelemente erreicht werden.
- **<refererpattern>** Innerhalb eines CDATA-Blocks kann mit Hilfe eines regulären Ausdrucks angegeben werden, bei welcher Referer-Adresse das entsprechende Hilfeelement aufgerufen werden soll. Dies hat den Sinn, dass der Benutzer gleich auf die zur aktuellen Seite passende Hilfe-Seite kommt, wenn er auf den Hilfe-Button klickt. So können eine aber auch mehrere Seiten auf ein Hilfeelement verweisen.

Nachdem Sie die Struktur der Hilfe über diese Datei angelegt haben, müssen Sie noch für die entsprechenden Inhalte sorgen. Hierfür sind die Sprachdateien zu verwenden. Für jedes Hilfeelement gibt es zwei Sprachvariablen. Beim Hilfeelement `board` sind es folgende Variablen:

```
<item name="wcf.help.item.board">
<item name="wcf.help.item.board.description">
```

In der ersten Variablen wird eine Überschrift angegeben, die zweite enthält den entsprechenden Hilfetext. Hier kann von HTML Gebrauch gemacht werden. Die Kategorie der Sprachvariablen ist `<category name="wcf.help.item">`.

Code-Beispiel

Programm 13.7 Exemplarischer Aufbau einer `help.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/help.dtd">
<data>
  <import>
    <helpitem name="board">
      <showorder>3</showorder>
    </helpitem>
    <helpitem name="board.index">
      <parent>board</parent>
      <showorder>1</showorder>
    </helpitem>
    ...
  </import>
</data>
```

13.2.8. Das BBCodes-PIP

Verwendungszweck

Das BBCodes-PIP wird verwendet, um BB-Codes im System zu installlieren.

XML-Code in der `package.xml`

```
<bbcodes>bbcodes.xml</bbcodes>
```

Tags und deren Bedeutung

<bbcode> Mit dessen `name`-Attribut kann der Name des BB-Codes angegeben werden.

- **<classname>** Angabe des Namens einer Klasse, die sich um die Verarbeitung der Eingabe kümmert.
- **<htmlopen>** Der öffnende HTML-Tag.

· **<htmlclose>** Der schließende HTML-Tag.

· **<textopen>** Der öffnende Text.

· **<textclose>** Der schließende Text.

· **<allowedchilden>** Angabe der Namen der BBCodes (durch Kommata getrennt), die in diesem BBCode enthalten sein dürfen.

· **<wysiwyg>** Dieser BBCode kann im Editor nur als HTML dargestellt werden, wenn das Feld „PHP-Klassenname“ leer ist.

· **<wysiwygicon>** Angabe des Dateinamens des Icons, das im WYSIWYG-Editor verwendet werden soll. Das Icon muss im Ordner `icon/wsyiwyg` liegen. Sie sollten für den Titel des Icons eine Sprachvariable hinzufügen. Sprachkategorie: „wcf.bbcode“, Name der Sprachvariable: „wcf.bbcode.mycode.title“, wobei „mycode“ genau dem Eintrag „BBCode-Tag“ entspricht.

· **<attributes>** Enthält eine Liste von Attributen.

· · **<attribute>** Mit dem Parameter `name`, um das `<attribute>` zu bezeichnen

· · · **<html>** Der HTML-Code des Attributs.

· · · **<validationpattern>** Angabe des regulären Ausdrucks, der zur Validierung des Attributs benötigt wird.

· · · **<required>** Dieses Attribut muss zwingend ausgefüllt werden.

· · · **<text>** Der Text bzw. Inhalt des Attributs.

· · · **<usetext>** Falls dieses Attribut nicht ausgefüllt ist, kann optional der Textinhalt des BBCodes übernommen werden.

13.2.9. Das Smilies-PIP

Verwendungszweck

Das Smilies-PIP wird verwendet, um Smileys im System zu installieren.

XML-Code in der `package.xml`

```
<smilies>smilies.xml</smilies>
```

Tags und deren Bedeutung

<smiley> Dieses speichert im Attribut `name` den Smiley-Code, den der Benutzer später auch beim Schreiben einer Nachricht verwenden muss.

- **<title>** enthält den Namen des Smileys, der auch als HTML-Title- und Alt-Tag verwendet wird.
- **<path>** enthält den Pfad zur Bilddatei, die den Smiley-Code in der Nachricht ersetzt.

Bemerkung: Da in der XML-Datei nur der Pfad zum Smiley-Bild übertragen werden kann und nicht das Smiley selbst, müssen Sie dafür Sorge tragen, dass das Bild auch in den entsprechenden Ordner kopiert wird. Dazu nutzen Sie bitte das Files-PIP.

Code-Beispiel

Programm 13.8 Exemplarischer Aufbau einer `smilies.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/smilies.dtd">
<data>
  <import>
    <smiley name=":)">
      <title>smile</title>
      <path>images/smilies/smile.png</path>
    </smiley>
    <smiley name=":(">
      <title>sad</title>
      <path>images/smilies/sad.png</path>
    </smiley>
    ...
  </import>
</data>
```

13.2.10. Das SearchableMessageType-PIP

Verwendungszweck

Das WCF bietet bereits eine eigene Suchfunktion inklusive Suchformular und Anzeige der Resultate an. Um ein Nachrichtensystem der Suchmaschine bekannt zu machen, muss das SearchableMessageType-PIP genutzt werden.

XML-Code in der `package.xml`

```
<searchablemessagetypes>smt.xml</searchablemessagetypes>
```

Tags und deren Bedeutung

<smt> Dieses speichert im Attribut `name` den Namen des Nachrichtentyps.

- **<classpath>** Ein Pfad zu einer Klasse ist anzugeben, die das Interface `Searchable-MessageType` implementiert.

Es ist zusätzlich auch eine Sprachvariable anzulegen. Der Name ergibt sich aus dem Namen des Nachrichtentyps:

```
<item name="wcf.search.type.pm"><![CDATA[Private Nachrichten]]></item>
```

Code-Beispiel

Programm 13.9 Exemplarischer Aufbau einer `smt.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/smt.dtd">
<data>
  <import>
    <smt name="pm">
      <classpath>lib/data/message/pm/PMSearch.class.php</classpath>
    </smt>
  </import>
</data>
```

13.2.11. Das PageLocation-PIP

Verwendungszweck

Es gibt eine Seite, wo angezeigt wird, welcher Benutzer gerade wo auf der Seite aktiv ist. Dafür wird eine Liste mit Orten im System benötigt, die mit der aktuellen URL abgeglichen werden kann. Das PageLocation-PIP wird verwendet, um mögliche Aufenthaltsorte (Page Locations) des Benutzers im System zu installieren.

XML-Code in der `package.xml`

```
<pagelocation>pagelocation.xml</pagelocation>
```

Tags und deren Bedeutung

<pagelocation> Dieses speichert im Attribut `name` den Namen des Ortes.

- **<classpath>** Optional den Pfad zu einer Klasse angeben, die diese URL verarbeitet, um z. B. zu einer `threadID` den passenden Themennamen herauszusuchen.
- **<pattern>** enthält innerhalb eines CDATA-Blocks den regulären Ausdruck zu einer URL. Diese wird später mit der aktuellen URL des Benutzers verglichen.

Der Name einer Page Location kann so auch als Sprachvariable genutzt werden, dazu müssen die entsprechenden Übersetzungen selbst in einer Sprachdatei beigefügt werden. Wenn Ihre Page Location z. B. folgenden Namen hat, so verwenden Sie diesen bitte auch als Namen der Sprachvariable:

```
<pagelocation name="wbb.usersOnline.location.board"></pagelocation>
<item name="wbb.usersOnline.location.board">
     <![CDATA[Forum: {$board}]]>
</item>
```

Code-Beispiel

Programm 13.10 Exemplarischer Aufbau einer `pagelocation.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/pageLocations.dtd">
<data>
  <import>
    <pagelocation name="wbb.usersOnline.location.subscriptions">
      <pattern><![CDATA[index\.php\?page=Subscriptions]]></pattern>
    </pagelocation>
    <pagelocation name="wbb.usersOnline.location.thread">
      <pattern><![CDATA[index\.php\?page=Thread&.*threadID=(\d+)]]></pattern>
      <classpath>lib/data/page/location/ThreadLocation.class.php</classpath>
    </pagelocation>
    ...
  </import>
</data>
```

13.2.12. Das HeaderMenu-PIP

Verwendungszweck

Das Header-PIP kann genutzt werden, um neue Menüpunkte im Hauptmenü anzulegen.

XML-Code in der `package.xml`

```
<headermenu>headermenu.xml</headermenu>
```

Tags und deren Bedeutung

<headermenuitem> Dieses speichert im Attribut `name` den Namen der Sprachvariable. Diese müssen Sie dann in der Sprachdatei mitliefern.

- **<icon>** enthält den Pfad zu einer Bild-Datei, die als Icon verwendet werden soll. Die Icons im WCF werden im Hauptmenü in M-Größe (24px:24px) verwendet. Es ist empfehlenswert für eigene Menüpunkte die Icons auch in dieser Größe anzulegen, um ein konsistentes Erscheinungsbild zu gewähren.
- **<link>** gibt an, welche Seite mit dem Menüpunkt verlinkt werden soll.
- **<showorder>** Ermöglicht eine Sortierung des Menüpunkts. Je größer die Zahl, desto weiter hinten steht der Menüpunkt.

Bemerkung: Da in der XML-Datei nur der Pfad zum Icon-Bild sowie der Name der Sprachvariablen angegeben wird, müssen Sie dafür sorgen, dass das Bild und die Sprachvariable im System installiert werden. Verwenden Sie dazu das Files-PIP und das Languages-PIP. Wenn Ihr Menüelement beispielsweise folgenden Namen hat, so verwenden Sie diesen bitte auch als Namen der Sprachvariablen:

```
<headermenuitem name="wbb.header.menu.board"></headermenuitem>
<item name="wbb.header.menu.board"><![CDATA[Forum]]></item>
```

Code-Beispiel

Programm 13.11 Exemplarischer Aufbau einer `headermenu.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/headerMenu.dtd">
<data>
  <import>
    <headermenuitem name="wbb.header.menu.board">
      <icon>icon/indexM.png</icon>
      <link>index.php?page=Index</link>
      <showorder>1</showorder>
    </headermenuitem>
    ...
  </import>
</data>
```

13.2.13. Das UserCPMenu-PIP

Verwendungszweck

Das UserCPMenu-PIP wird verwendet, um neue Menüpunkte im Bereich „Mein Profil“ einzufügen.

XML-Code in der `package.xml`

```
<usercpmenu>usercpmenu.xml</usercpmenu>
```

Tags und deren Bedeutung

Dieses PIP ist stark vergleichbar mit dem HeaderMenu-PIP. Folgende Tags kommen hinzu.

<usercpmenuitem> siehe HeaderMenu-PIP

- **<parent>** Im Gegensatz zum HeaderMenu kann das UserCPMenu hierarchisch aufgebaut sein. Mit `<parent>` geben Sie das Elternelement an.
- **<permissions>** wird verwendet, um die Seite mit einem Benutzerrecht zu verknüpfen. Nur wenn der Benutzer das angegebene Recht besitzt, kann er die Seite aufrufen. Um hier mehrere Rechte anzugeben, schreiben Sie diese einfach getrennt mit einem Komma auf.

Code-Beispiel

13.2.14. Das ACPMenu-PIP

Verwendungszweck

Das ACPMenu-PIP wird dazu verwendet neue Menüpunkte im Administrationsbereich anzulegen.

XML-Code in der `package.xml`

```
<acpmenu>acpmenu.xml</acpmenu>
```

Programm 13.12 Exemplarischer Aufbau einer `usercpmenu.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/userCPMenu.dtd">
<data>
  <import>
    <usercpmenuitem name="wcf.user.usercp.menu.link.profile">
      <icon>icon/profileM.png</icon>
      <link>index.php?form=UserProfileEdit</link>
      <showorder>1</showorder>
    </usercpmenuitem>

    <usercpmenuitem name="wcf.user.usercp.menu.link.profile.email">
      <icon>icon/emailS.png</icon>
      <link>index.php?form=EmailChange</link>
      <parent>wcf.user.usercp.menu.link.profile</parent>
      <showorder>2</showorder>
      <permissions>user.profile.canChangeEmail</permissions>
    </usercpmenuitem>
    ...
  </import>
</data>
```

Tags und deren Bedeutung

Dieses PIP ist stark vergleichbar mit dem HeaderMenu- und UserCPMenu-PIP. Der einzige Unterschied zum UserCPMenu-PIP ist das Tag `<acpmenuitem>` welches anstelle von `<usercpmenuitem>` verwendet wird.

13.2.15. Das StyleAttributes-PIP

Verwendungszweck

Das StyleAttributes-PIP wird genutzt, um die Attribute ausgewählter CSS-Selektoren mit Stilvariablen zu verknüpfen. Dadurch wird die Anwendung unabhängig von bestimmten Stilen.

Beispiel: Sie haben in einem Template einen Informationskasten definiert:

```
<div class="infoContainer">Informationen</div>
```

Nun möchten Sie, dass dieser Kasten das Stil-System des WCF nutzt und eine vorhandene Farbe als Hintergrundfarbe verwendet. Sie entscheiden sich für die Variable `container1.background.color`. Dazu erstellen Sie ein StyleAttributes-PIP wie im Codebeispiel 13.13 auf der nächsten Seite abgebildet.

Auf diese Art und Weise haben Sie keine Farbe fest im System kodiert, sondern sind Stil-unabhängig. Ihr Informationskasten passt sich also dem jeweiligen Stil an. Für viele Elemente wurden schon solche Verknüpfungen angelegt. Es ist selten notwendig eigene Verknüpfungen anzulegen.

XML-Code in der `package.xml`

`<styleattributes>styleattributes.xml</styleattributes>`

Tags und deren Bedeutung

<attribute> das Stil-Attribut

- **<selector>** Angabe eines CSS-Selektors[1] – z. B. `.class`, `#id` oder `tag`.
- **<name>** Angabe eines CSS-Attributs – z. B. `color`, `border-style` oder `width`.
- **<value>** Name einer Stilvariablen, dessen Wert hier eingetragen werden soll. Eine Auflistung aller Stilvariablen finden Sie im Anhang.

Code-Beispiel

Programm 13.13 Exemplarischer Aufbau einer `styleattributes.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/styleattributes.dtd">
<data>
  <import>
    <attribute>
      <selector><![CDATA[.infoContainer]]></selector>
      <name><![CDATA[background-color]]></name>
      <value><![CDATA[container1.background.color]]></value>
    </attribute>
    ...
  </import>
</data>
```

13.2.16. Das Languages-PIP

Verwendungszweck

Das Languages-PIP wird zum Installieren von Sprachdateien verwendet.

XML-Code in der `package.xml`

`<languages languagecode="de">de.xml</languages>` Für jede zu installierende Sprache wird ein eigenes Tag verwendet. Der Parameter `languagecode` gibt den Sprachcode der jeweiligen Sprache an.

[1] http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap05.html

Tags und deren Bedeutung

<language> Das Tag enthält eine Liste von Sprachkategorien (`<category>`). Das Attribut `languagecode` erwartet eine korrekte Angabe des Sprachcodes nach ISO 639-1-Standard[2].

· **<category>** Das Tag enthält eine Liste von Sprachvariablen (`<item>`). Das Attribut `name` gibt den Namen der Sprachkategorie an. Kategorien werden benötigt, um die vielen Sprachvariablen zu sortieren.

Wie bei Paketnamen ist es empfehlenswert bestimmte Namensräume zu verwenden z. B. „wcf.acp.global“. Kategorien dürfen dabei maximal aus drei Blöcken bestehen, die mit einem Punkt (.) getrennt werden. Jeder Teil muss aus mindestens einem alphanumerischen[3] oder dem Unterstrich (_) bestehen.

· · **<item>** Jedes <item>-Tag steht für eine eigene Sprachvariable. Das Attribut `name` gibt wie bei den Kategorien Auskunft über den Namen der Variable. Der Name beginnt mit dem Namen der Sprachkategorie. Für die Benennung können die gleichen Zeichen wie bei Kategorien verwendet werden.

Das Tag enthält den Inhalt der Sprachvariable. Dieser sollte innerhalb eines C-DATA[4]-Blocks notiert sein. Darüber hinaus gelten folgende Regeln:

Kodierung beachten Als Kodierung für Sprachdateien ist nur UTF-8 zulässig. Achten Sie auch darauf keine HTML-Entities zu verwenden, beispielsweise um Sonderzeichen zu schreiben. Für diese Regel gibt es nur drei Ausnahmen:

```
&quot; für " verwenden
&lt;   für < verwenden
&gt;   für > verwenden
```

Templatescripting erlaubt Innerhalb der Sprachvariablen darf Templatescripting verwendet werden.

Kontext beachten Bei der Verwendung von Sprachvariablen muss der Kontext beachtet werden, in dem die Variable verwendet wird. Dies gilt sowohl für das Templatescripting als auch für HTML-Entities. Diese sind in Javascript-Anweisungen oder E-Mails nicht gültig. Beim Templatescripting können auf keine Variablen zurückgegriffen werden, die innerhalb des Templates nicht bekannt sind. Wenn Variablen direkt per PHP verwendet werden, ist kein Templatescripting erlaubt[5].

[2] http://de.wikipedia.org/wiki/ISO_639
[3] a bis z, A bis Z oder 0 bis 9
[4] http://de.wikipedia.org/wiki/CDATA
[5] Dies soll in einer späteren Version geändert werden

Code-Beispiel

Programm 13.14 Exemplarischer Aufbau einer deutschen Sprachdatei

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE language SYSTEM "http://www.woltlab.com/DTDs/language.dtd">
<language languagecode="de">
  <category name="example.category.name">
    <item name="example.category.name.variable1"><![CDATA[Beispielinhalt]]></item>
    <item name="example.category.name.variable2"><![CDATA[Beispielinhalt]]></item>
    ...
  </category>
  ...
</language>
```

13.3. Sonstige PIPs

13.3.1. Das SQL-PIP

Verwendungszweck

Das SQL-PIP wird zum Ausführen von SQL Anweisungen während der Installation von Paketen benutzt. Beim typischen Anwendungsfall handelt es sich z. B. um `CREATE TABLE` oder `ALTER TABLE` Befehle. Aber auch andere SQL Befehle können ausgeführt werden.

XML-Code in der `package.xml`

```
<sql>install.sql</sql>
```

Funktionsweise

Dabei überprüft das SQL-PIP vor der Ausführung ob die jeweiligen Anweisungen erlaubt sind. So ist es beispielsweise einem Paket verboten, Tabellen oder Tabellenfelder eines anderen Paketes zu löschen.

Das SQL-PIP loggt alle Anweisungen folgenden Typs, um sie bei der Deinstallation rückgängig machen zu können.

- `CREATE TABLE`
- `DROP TABLE`
- `CREATE INDEX`
- `DROP INDEX`

- `ALTER TABLE`
- `RENAME TABLE`

Warnung: Wie Sie vielleicht bemerkt haben, werden `INSERT INTO`, `DELETE FROM` und `UPDATE TABLE` Anweisungen vom SQL-PIP **nicht** geloggt und können daher nicht rückgängig gemacht werden.

13.3.2. Das Script-PIP

Verwendungszweck

Das Script-PIP wird verwendet, um einmalige und komplexere Operationen während der Installation auszuführen.

XML-Code in der `package.xml`

```
<script>script.php</script>
```

Funktionsweise

Der Aufruf erfolgt innerhalb der Klasse `ScriptPackageInstallationPlugin`. Alle Variablen, die dort genutzt werden können, sind somit auch in der Script-Datei möglich.

Bemerkung: Das PHP-Skript muss vorher mit dem Files-PIP installiert werden.

13.3.3. Das TemplatePatch-PIP

Verwendungszweck

Das TemplatePatch-PIP wird verwendet, um per Plugin bereits vorhandene Templates eines Paketes zu modifizieren. Dabei wird ein sog. *Unified Diff*[6] bzw. *Patch* verwendet. Dieser *Patch* enthält nur die geänderten Zeilen des Templates und nicht das komplette Template.

Ein solcher *Patch* wird mit dem Kommandozeilen Programm *diff*[7] mit dem Befehl

```
diff -u template.tpl.old template.tpl
```

erzeugt.

[6] http://en.wikipedia.org/wiki/Diff
[7] http://www.gnu.org/software/diffutils/ bzw.
http://sourceforge.net/projects/unxutils für Windows

XML-Code in der `package.xml`

```
<templatepatch>patch.diff</templatepatch>
```

Funktionsweise

Das TemplatePatch-PIP versucht den mitgelieferten Patch auf die Templates anzuwenden, indem die einzelnen Teile des Patches, die sog. *Hunks* gesucht werden. Es wird nach den zu ersetzenden Zeilen und dem umgebenden Kontext gesucht (üblich sind drei Zeilen Kontext).

Anwendung von ungenauen Patches Dies funktioniert in der Regel ohne Probleme. Jedoch kann es in manchen Fällen vorkommen, dass das zu patchende Template schon durch ein anderes Plugin verändert wurde und dass deshalb die korrekte Stelle für die Änderungen nicht mehr gefunden werden kann.

Für den Fall, dass „nur“ der Kontext der zu ersetzenden Zeilen nicht gefunden wurde, gibt es im TemplatePatch-PIP den sog. *Fuzz-Faktor*[8]. Der Fuzz-Faktor ist eine natürliche Zahl[9] und bestimmt die maximale Anzahl der Zeilen, die das TemplatePatch-PIP bei der Suche nach der passenden Textstelle ignoriert. Zuerst sucht das TemplatePatch-PIP nach einer Stelle, in der alle Zeilen des Kontextes exakt übereinstimmen. Wird eine solche Stelle nicht gefunden und der Fuzz-Faktor ist größer gleich 1, dann sucht das TemplatePatch-PIP erneut wobei nun aber die erste und die letzte Zeile des Kontextes ignoriert werden. Schlägt auch diese Suche fehl und ist der Fuzz-Faktor größer gleich 2, dann ignoriert das TemplatePatch-PIP bei der erneuten Suche nun die ersten 2 und die letzten 2 Zeilen des Kontextes.

Bemerkung: Der Standardwert des Fuzz-Faktors beträgt 2. Es macht keinen Sinn, den Fuzz-Faktor größer als die Größe des Kontextes (üblicherweise 3) einzustellen. Je größer der Fuzz-Faktor, desto größer ist die Gefahr, dass das TemplatePatch-PIP die falsche Stelle zum Patchen auswählt.

Um den Standardwert des Fuzz-Faktors zu ändern verwendet man in der `package.xml` im Tag <templatepatch> den Parameter fuzzfactor:
`<templatepatch fuzzfactor="3">patch.diff</templatepatch>`

Falls (unabhängig vom Kontext) aber die zu ersetzenden Zeilen nicht gefunden werden, kann der Patch überhaupt nicht angewandt werden. In diesem Fall ist eine manuelle Bearbeitung des Templates nötig.

[8] siehe auch http://www.gnu.org/software/diffutils/manual/html_node/Inexact.html
[9] Die Zahlen 1, 2, 3, ... usw.

Erneutes Patchen nach einem Update Wenn bei einem Update eines Paketes eine neue Version eines Templates, auf das ein Patch angewandt wurde, ausgeliefert wird, dann versucht das TemplatePatch-PIP den Patch nach dem Update auf die neue Version des Templates anzuwenden. Falls dies nicht gelingt, wird eine entsprechende Fehlermeldung auf dem Bildschirm ausgegeben.

Bemerkung: Das TemplatePatch-PIP arbeitet nur mit *Unified Diffs*, normale Diffs oder *Context Diffs* funktionieren nicht.

Bemerkung: Das TemplatePatch-PIP bearbeitet nur Templates im Standardtemplatepack. Änderungen an eigenen Templatepacks müssen manuell vorgenommen werden.

13.3.4. Das ACPTemplatePatch-PIP

Verwendungszweck

Das ACPTemplatePatch-PIP wird zum Anwenden von Patches auf ACP-Templates genutzt.

XML-Code in der `package.xml`

```
<acptemplatepatch>patch.diff</acptemplatepatch>
```

Funktionsweise

Siehe TemplatePatch-PIP 13.3.3 auf Seite 92.

13.4. Eigene PIPs

Es ist möglich das System, um eigene PIPs zu erweitern, wenn die vorhandenen PIPs nicht die gewünschte Funktionalität abdecken. Wie in der Einleitung des Kapitels bereits erwähnt, wird ein PIP durch eine Klasse abgebildet, die das Interface `PackageInstallationPlugin` implementiert.

13.4.1. Das Interface

Der erste Schritt besteht also darin eine entsprechende Klasse anzulegen, die dieses Interface implementiert. Einige abstrakte Klassen, die später vorgestellt werden, nehmen einem bereits einiges an Implementation ab. Zunächst werden die zu implementierenden Methoden vorgestellt:

hasInstall

boolean **hasInstall** ()

`hasInstall()` gibt `true` zurück, wenn die Installation des aktuellen Pakets dieses Plugin ausführen soll. Eine Überprüfung des verwendeten Tags ist hier denkbar.

install

void **install** ()

`install()` führt die Installation des Plugins aus.

hasUpdate

boolean **hasUpdate** ()

`hasUpdate()` gibt `true` zurück, wenn das Update des aktuellen Pakets dieses Plugin ausführen soll. Eine Überprüfung des verwendeten Tags ist hier denkbar.

update

void **update** ()

`update()` führt das Update des Plugins aus.

hasUninstall

boolean **hasUninstall** ()

`hasUninstall()` gibt `true` zurück, wenn die Deinstallation des aktuellen Pakets dieses Plugin ausführen soll. Eine Überprüfung des verwendeten Tags ist hier denkbar.

uninstall

void **uninstall** ()

`uninstall()` führt die Deinstallation des Plugins aus.

13.4.2. Abstrakte Klassen

Da sich die Anforderungen an ein PackageInstallationPlugin häufig ähneln, werden bereits vom WCF einige abstrakte Klassen mitgeliefert, die einige der Methoden implementieren.

AbstractPackageInstallationPlugin

Dies ist die Standard-Implementierung des Interfaces. Es werden alle sechs Methoden implementiert, wobei hier vor allem das Auslösen der entsprechenden Events stattfindet. Die `install()`-Methode ist die einzige die wirklich überschrieben werden muss, um effektiv neue Funktionalität zu implementieren.

AbstractXMLPackageInstallationPlugin

Für die vielen XML-basierten PIPs ist diese abstrakte Klasse in der Regel der Ausgangspunkt, sie erweitert `AbstractPackageInstallationPlugin` um folgende Methoden:

getXML *XML* **getXML** ()

`getXML()` liest die angegebene XML-Datei aus und speichert den Inhalt in einem XML-Object[10], welches zurückgegeben wird.

getShowOrder *integer* **getShowOrder** (*integer* $showOrder, [*string* $parentName = null], [*string* $columnName = null], [*string* $tableNameExtension = ''])

`getShowOrder()` gibt einen `showOrder`-Wert zurück, welcher die Position eines Elements innerhalb einer Ebene bestimmt.

AbstractOptionPackageInstallationPlugin

Diese abstrakte Klasse wird für die zahlreichen Options-PIPs genutzt und implementiert vor allem die `install()`-Methode, wo das XML-Dokument durchwandert wird und die einzelnen Tags ausgelesen werden. Wichtig ist hier vor allem die neue abtrakte Methode `saveOption()`:

[10] Intern wird die *SimpleXML*-Implementierung von PHP 5 verwendet. Die Klasse `XML` ist in `wcf/lib/util/XML.class.php` definiert.

saveOption *void* **saveOption** (`array` $option, `string` $categoryName, [`integer` $existingOptionID = 0])

`saveOption()` speichert eine Option mit den jeweiligen Eigenschaften in der Datenbank. Der Parameter `option` übergibt ein Array mit Werten aus dem XML-Dokument. `categoryName` gibt den Namen der jeweiligen Kategorie an. Sofern bereits eine `optionID` einer Option vorhanden ist, kann diese über `existingOptionID` übergeben werden.

13.4.3. Installation des PIPs

Die Klasse ist nun in ein Tar-Archiv zu packen und kann mit dem PIPs-PIP installiert werden. Dazu mehr in Kapitel 13.1.5 auf Seite 71.

14. Endanwendungen

Wie bereits in vorherigen Kapiteln erwähnt, handelt es sich bei Endanwendungen um spezielle Pakete, die ihre eigene grafische Oberfläche mitbringen und auf vorhandenen Paketen aufbauen. Zusätzlich sei gesagt, dass Endanwendungen zwingend auch einen administrativen Bereich mitbringen. Nachfolgend werden die einzelnen Schritte beschrieben, die notwendig sind, um eine eigene Endanwendung auf WCF-Basis zu erstellen.

14.1. Paket erstellen

Zunächst muss natürlich erst einmal ein entsprechendes Paket erstellt werden. Hierzu erfahren Sie in Kapitel 12 auf Seite 63 mehr. In der `package.xml` geben Sie bei einer Endanwendung im `<packageinformation>`-Block folgende Anweisung an:

```
<standalone>1</standalone>
```

Die weiteren Angaben können Sie frei wählen. Bei den benötigten Paketen sind alle Pakete anzugeben, die von Ihrer Endanwendung vorausgesetzt werden.

Bemerkung: Die Abhängigkeit zu `com.woltlab.wcf` muss nur dann angegeben werden, wenn eine Abhängigkeit zu einer speziellen Version besteht. Sonst sind alle Pakete automatisch von diesem Paket abhängig.

14.2. Ableitung der Klassen `WCF` und `WCFACP`

Der Aufbau des WCF ist dem *MVC-Pattern*[1] angelehnt. Hierbei entspricht die Klasse `WCF` (bzw. `WCFACP`) der zentralen Steuerungseinheit der Anwendung (*Controller*). Diese kann direkt verwendet werden. Sie müssen jedoch eine eigene Ableitung der Klasse erstellen, wenn Sie z. B. auf eigene Cache-Resourcen oder Templates zurückzugreifen möchten. Es empfiehlt sich, die `WCF`-Klasse und deren Methoden genau anzusehen. (Gleiches gilt für `WCFACP`.)

[1] http://de.wikipedia.org/wiki/MVC

14.3. Erstellung einer `IndexPage`-Klasse

Die `IndexPage`-Klasse wird standardmäßig vom `RequestHandler` aufgerufen, wenn keine Parameter über die URL mitgegeben wurden. Es handelt sich also um die Startseite Ihrer Endanwendung bzw. der Eingangsseite im Administrationsbereich.

Hinweise zum `RequestHandler` und zum `Page`-Interface finden Sie in Kapitel 11 auf Seite 59.

14.4. Erstellung einer `index.php`-Datei

Innerhalb der `index.php`-Datei werden wichtige Dateien eingebunden und die Anwendung initialisiert. Einige der nachfolgenden Anweisungen können auch in eine separate Datei ausgelagert werden, die dann wiederum eingebunden werden muss. Darauf wird an dieser Stelle verzichtet. Im folgenden Code-Beispiel wird davon ausgegangen, dass die Endanwendung `com.application.test` heißt.

Programm 14.1 Beispiel für eine `index.php`-Datei

```
// initialize package array
$packageDirs = array();
// include config
require_once(dirname(__FILE__).'/config.inc.php');

// include WCF
require_once(RELATIVE_WCF_DIR.'global.php');
//
if (!count($packageDirs)) $packageDirs[] = TEST_DIR;
$packageDirs[] = WCF_DIR;

// starting test application
require_once(TEST_DIR.'lib/system/Test.class.php');
new Test();

RequestHandler::handle(ArrayUtil::appendSuffix($packageDirs, 'lib/'));
```

Als erstes wird das Array `$packageDirs` initialisiert. Dieses wird in der danach eingebundenen `config.inc.php`-Datei verwendet. Die Datei wird automatisch vom WCF angelegt und enthält einige Konstanten-Definitionen. Unter anderem wird der relative Pfad zum Ordner der Endanwendung in einer Konstanten abgespeichert. Der Name der Konstanten wird automatisch aus dem Paketnamen generiert. Hierbei wird der letzte Block verwendet. Bei `com.application.test` wäre dies *test*. Die Konstante lautet dann `RELATIVE_TEST_DIR` und die des absoluten Pfads `TEST_DIR`.

Bemerkung: In unserem Beispiel müsste man eventuelle eigene Datenbanktabellen dieses Paketes innerhalb der `install.sql`-Datei mit dem Präfix *test1_1_* benennen.

Als nächstes wird das WCF eingebunden und das Array `$packageDirs` mit Werten gefüllt. Zum einen wird der absolute Pfad zur aktuellen Endanwendung eingetragen. Zum

anderen der Pfad zum WCF-Verzeichnis. Jetzt kann die Anwendung gestartet werden. Die Klasse `Test` ist von WCF abgeleitet.

Als letztes wird dem `RequestHandler` das Array `$packageDirs` übergeben. An die beiden Pfadangaben wird noch der `lib`-Ordner gehängt. Dieser enthält die eigentlichen Klassen, sowohl im WCF, als auch bei der Endanwendung.

Teil III.

Anhänge

15. Events

15.1. Events der freien WCF-Pakete

Ort des Events	Name des Events
com.woltlab.wcf.data.message.bbcode · URLParser	
parse()	didParse shouldParse
com.woltlab.wcf.page.util.menu · HeaderMenu	
loadCache()	loadCache
buildMenu()	buildMenu
com.woltlab.acp.package.plugin · StylePackageInstallationPlugin	
uninstall()	uninstall
com.woltlab.wcf.acp.form · UserSearchForm	
search()	buildConditions
com.woltlab.wcf.acp.package.plugin · AbstractPackageInstallationPlugin	
__construct()	construct
hasInstall()	hasInstall
install()	install
hasUpdate()	hasUpdate
update()	update
hasUninstall()	hasUninstall
uninstall()	uninstall
com.woltlab.wcf.action · AbstractAction	
readParameters()	readParameters
execute()	execute
executed()	executed
com.woltlab.wcf.page.util.menu	

· TreeMenu	
loadCache()	loadCache
buildMenu()	buildMenu
com.woltlab.wcf.page	
· AbstractForm	
submit()	submit
readFormParameters()	readFormParameters
validate()	validate
save()	save
saved()	saved
· AbstractPage	
readParameters()	readParameters
readData()	readData
assignVariables()	assignVariables
show()	show
· MultipleLinkPage	
calculateNumberOfPages()	calculateNumberOfPages
countItems()	countItems
· SortablePage	
validateSortField()	validateSortField
validateSortOrder()	validateSortOrder
com.woltlab.wcf.system.auth	
· UserAuth	
getInstance()	loadInstance
com.woltlab.wcf.system.session	
· SessionFactory	
get()	shouldInit
	didInit
com.woltlab.wcf.system.template	
· Template	
display()	shouldDisplay
	didDisplay

Tabelle 15.1.: Events der freien WCF-Pakete

16. Stilvariablen

In diesem Kapitel werden alle Variablen beschrieben, die innerhalb der `variables.xml`-Datei verwendet werden dürfen. Die Variablen wurden so sortiert, wie sie auch innerhalb der grafischen Oberfläche im Administrationsbereich zu bearbeiten sind. Die angegebenen Standard-Werte beziehen sich auf den Stil „WoltLab Basic“.

Bitte beachten Sie, dass sich die Werte, die man bei den einzelnen Variablen angeben kann, von denen unterscheiden können, die im Stileditor einzutragen sind.

Bei der Auflistung der Variablen wird zunächst der **Name der Variable** angegeben. Es folgt der `Wert`, so wie er im „WoltLab Basic“-Stil verwendet wird. Es kann daher auch vorkommen, dass kein Wert angegeben ist. In eckigen Klammern steht der *Wertebereich*, der Auskunft gibt, welche Angaben man hier machen kann. Alle Werte sind innerhalb eines CDATA-Blocks anzugeben. Abgeschlossen wird die Angabe durch eine kurze Beschreibung der Variablen.

16.1. Global

16.1.1. Allgemein

Darstellung

page.alignment `center` [*align*] Ausrichtung der Seite – Der Befehl funktioniert nur in Verbindung mit der nächsten Variable.

page.alignment.margin `margin-left:auto;margin-right:auto;` [*custom*] Ausrichtung der Seite (links: „margin-left: auto; margin-right: 0“, rechts: „margin-left: 0; margin-right: auto“ und zentriert „margin-left: auto; margin-right: auto“)

page.width [*lengths*] Feste Seitenbreite (css-Befehl `width`) für eine statische Breite

page.width.max `80%` [*lengths*] Max. Seitenbreite (css-Befehl `max-width`) für eine flexible Breite – sollte nicht zusammen mit der statischen Breite genutzt werden.

page.width.min `760px` [*lengths*] Min. Seitenbreite (css-Befehl `min-width`) für eine flexible Breite – sollte nicht zusammen mit der statischen Breite genutzt werden.

Speicherort für Grafiken

global.icons.location `icon/` [*path*] Pfad zum Iconordner (die Angabe wird momentan noch ignoriert)

global.images.location `images/` [*path*] Pfad zum Bilderordner (eigene Bilder am besten in einen Unterordner von images legen)

Favoriten-Icon

global.favicon `grey` [*favicon*] Name

16.1.2. Seite

Seitenkopf

page.header.background.color `#777` [*bcolor*] Hintergrundfarbe

page.header.height `90px` [*lengths*] Höhe

page.header.background.image [*burl*] Hintergrundbild-URL

page.header.background.image.alignment [*balign*] Hintergrundbild Ausrichtung

page.header.background.image.repeat [*brepeat*] Hintergrundbild wiederholen

Logo

page.logo.image `images/wbb3-header-logo.png` [*path*] Pfad zum Bild

page.logo.image.alignment `left` [*align*] Ausrichtung

page.logo.image.padding.top `5px` [*lengths*] Innerer Abstand (oben)

page.logo.image.padding.right `0px` [*lengths*] Innerer Abstand (rechts)

page.logo.image.padding.left `13px` [*lengths*] Innerer Abstand (links)

Globaler Titel

global.title.hide `position: absolute; top: -9000px; left: -9000px;` [*hide*] Globalen Titel anzeigen

global.title.font [*font*] Schriftart

global.title.font.style [*custom*] Stil (font-style) – die Werte normal, italic und oblique sind möglich.

global.title.font.weight [*custom*] Stil (font-weight) – die Werte normal und bold werden von allen Browsern unterstützt.

global.title.font.size [*lengths*] Größe

global.title.font.color [*color*] Farbe

global.title.font.alignment [*align*] Ausrichtung

global.title.font.padding.top [*lengths*] Innerer Abstand (oben)

global.title.font.padding.right [*lengths*] Innerer Abstand (rechts)

global.title.font.padding.left [*lengths*] Innerer Abstand (links)

Hintergrund

page.background.color `#fff` [*bcolor*] Hintergrundfarbe

page.background.image [*burl*] Hintergrundbild-URL

page.background.image.attachment [*bfix*] Hintergrundbild fixieren

page.background.image.alignment [*balign*] Hintergrundbild Ausrichtung

page.background.image.repeat [*brepeat*] Hintergrundbild wiederholen

16.1.3. Kästen

Kasten 1

container1.background.color `#f7f7f7` [*bcolor*] Hintergrundfarbe

container1.font.color `#666` [*color*] Textfarbe

container1.font.2nd.color `#888` [*color*] Zweite Textfarbe

container1.link.color `#666` [*color*] Linkfarbe

container1.link.color.hover `#333` [*color*] Linkfarbe (Hover)

Kasten 2

container2.background.color `#efefef` [*bcolor*] Hintergrundfarbe

container2.font.color `#666` [*color*] Textfarbe

container2.font.2nd.color `#888` [*color*] Zweite Textfarbe

container2.link.color `#666` [*color*] Linkfarbe

container2.link.color.hover `#333` [*color*] Linkfarbe (Hover)

Kasten 3

container3.background.color `#e0e0e0` [*bcolor*] Hintergrundfarbe

container3.font.color `#333` [*color*] Textfarbe

container3.font.2nd.color `#777` [*color*] Zweite Textfarbe

container3.link.color `#666` [*color*] Linkfarbe

container3.link.color.hover `#333` [*color*] Linkfarbe (Hover)

16.1.4. Rahmen

Rahmenkopf

container.head.font.color `#fff` [*color*] Textfarbe

container.head.font.2nd.color `#fff` [*color*] Zweite Textfarbe

container.head.link.color `#fff` [*color*] Linkfarbe

container.head.link.color.hover `#fff` [*color*] Linkfarbe (Hover)

container.head.background.color `#777` [*bcolor*] Hintergrundfarbe

container.head.background.image [*burl*] Hintergrundbild-URL

Rahmen

container.border.outer.width `1px` [*lengths*] Äußere Randbreite

container.border.outer.style `solid` [*style*] Äußerer Randstil

container.border.outer.color `#999` [*color*] Äußere Randfarbe

container.border.inner.color `#fff` [*color*] Innere Randfarbe

divider.width `1px` [*lengths*] Randbreite (Trennlinie)

divider.style `solid` [*style*] Randstil (Trennlinie)

divider.color `#bbb` [*color*] Randfarbe (Trennlinie)

16.1.5. Formulare

Text

input.font `'Trebuchet MS', Arial, sans-serif` [*font*] Schriftart

input.font.size `.85em` [*lengths*] Schriftgröße

input.font.color `#333` [*color*] Schriftfarbe

input.font.color.focus `#000` [*color*] Schriftfarbe (Fokus)

Hintergrund

input.background.color `#fff` [*bcolor*] Hintergrundfarbe

input.background.color.focus `#ffd` [*bcolor*] Hintergrundfarbe (Fokus)

Rand

input.border.width `1px` [*lengths*] Randbreite

input.border.style `solid` [*style*] Randstil

input.border.color `#999` [*color*] Randfarbe

input.border.color.focus `#08f` [*color*] Randfarbe (Fokus)

16.2. Text

16.2.1. Textarten

Texte

page.font `'Trebuchet MS', Arial, sans-serif` [*font*] Schriftart

page.font.size `.8em` [*lengths*] Textgröße

page.font.2nd.size `.85em` [*lengths*] Zweite Textgröße

page.font.line.height `1.5` [*lengths*] Zeilenhöhe

page.font.color `#333` [*color*] Textfarbe

page.font.2nd.color `#888` [*color*] Zweite Textfarbe

Überschrift

page.title.font `'Trebuchet MS', Arial, sans-serif` [*font*] Schriftart

page.title.font.style `normal` [*custom*] Stil (font-style) – die Werte normal, italic und oblique sind möglich.

page.title.font.weight `normal` [*custom*] Stil (font-weight) – die Werte normal und bold werden von allen Browsern unterstützt.

page.title.font.size `1.3em` [*lengths*] Textgröße

page.title.font.color `#333` [*color*] Textfarbe

16.2.2. Verweise

Verweise

page.link.color `#666` [*color*] Linkfarbe

page.link.color.hover `#333` [*color*] Linkfarbe (Hover)

Externe Verweise

page.link.external.color `#333` [*color*] Externer Link-Farbe

page.link.external.color.hover `#08f` [*color*] Externer Link-Farbe (Hover)

Aktive Verweise

page.link.color.active `#08f` [*color*] Linkfarbe (Aktiv)

16.3. Buttons

16.3.1. Kleine Buttons

Beschriftung

buttons.small.caption.hide [*hide*] Beschriftung der Buttons anzeigen

buttons.small.caption.color `#666` [*color*] Textfarbe

buttons.small.caption.color.hover `#333` [*color*] Textfarbe (Hover)

Außenrahmen

buttons.small.border.outer.width `1px` [*lengths*] Äußere Randbreite

buttons.small.border.outer.style `solid` [*style*] Äußerer Randstil

buttons.small.border.outer.color `#999` [*color*] Äußere Randfarbe

buttons.small.border.outer.color.hover `#999` [*color*] Äußere Randfarbe (Hover)

Innenrahmen

buttons.small.border.inner.width `1px` [*lengths*] Innere Randbreite

buttons.small.border.inner.style `solid` [*style*] Innerer Randstil

buttons.small.border.inner.color `#fff` [*color*] Innere Randfarbe

buttons.small.border.inner.color.hover `#fff` [*color*] Innere Randfarbe (Hover)

Hintergundfarbe

buttons.small.background.color `#e8e8e8` [*bcolor*] Hintergrundfarbe

buttons.small.background.color.hover `#fff` [*bcolor*] Hintergrundfarbe (Hover)

Hintergrundgrafik

buttons.small.background.image [*burl*] Hintergrundbild-URL

buttons.small.background.image.hover [*burl*] Hintergrundbild-URL (Hover)

16.3.2. Große Buttons

Beschriftung

buttons.large.caption.hide [*hide*] Beschriftung der Buttons anzeigen

buttons.large.caption.color `#fff` [*color*] Textfarbe

buttons.large.caption.color.hover `#333` [*color*] Textfarbe (Hover)

Außenrahmen

buttons.large.border.outer.width `1px` [*lengths*] Äußere Randbreite

buttons.large.border.outer.style `solid` [*style*] Äußerer Randstil

buttons.large.border.outer.color `#999` [*color*] Äußere Randfarbe

buttons.large.border.outer.color.hover `#999` [*color*] Äußere Randfarbe (Hover)

Innenrahmen

buttons.large.border.inner.width `1px` [*lengths*] Innere Randbreite

buttons.large.border.inner.style `solid` [*style*] Innerer Randstil

buttons.large.border.inner.color `#fff` [*color*] Innere Randfarbe

buttons.large.border.inner.color.hover `#fff` [*color*] Innere Randfarbe (Hover)

Hintergundfarbe

buttons.large.background.color `#777` [*bcolor*] Hintergrundfarbe

buttons.large.background.color.hover `#cecece` [*bcolor*] Hintergrundfarbe (Hover)

Hintergrundgrafik

buttons.large.background.image [*burl*] Hintergrundbild-URL

buttons.large.background.image.hover [*burl*] Hintergrundbild-URL (Hover)

16.4. Menüs

16.4.1. Hauptmenü

Buttons

menu.main.position `text-align:left;margin:0 auto 0 0` [*custom*] Ausrichtung der Buttons (links: „text-align: left; margin: 0 auto 0 0“, rechts: „text-align: right; margin: 0 auto 0 0“ und zentriert „textalign: center; margin: 0 auto 0 auto;“)

menu.main.bar.hide `#f7f7f7` [*bcolor*] Hintergrundfarbe

menu.main.bar.show `#f7f7f7` [*bcolor*] Hintergrundfarbe

menu.main.bar.divider.show `1px` [*custom*] Soll eine Trennline angezeigt werden (ja: „1px“ und nein „0“)

Beschriftung

menu.main.caption.hide [*hide*] Wenn Beschriftung nicht angezeigt werden soll dann „position: absolute; top: -9000px; left: -9000px;“, ansonsten frei lassen.

menu.main.caption.color `#666` [*color*] Textfarbe

menu.main.caption.color.hover `#333` [*color*] Textfarbe (Hover)

menu.main.active.caption.color `#fff` [*color*] Textfarbe (Aktiv)

menu.main.active.caption.color.hover `#000` [*color*] Textfarbe (Aktiv Hover)

Hintergrundfarbe

menu.main.background.color `#efefef` [*bcolor*] Hintergrundfarbe

menu.main.background.color.hover `#fff` [*bcolor*] Hintergrundfarbe (Hover)

menu.main.active.background.color `#777` [*burl*] Hintergrundfarbe (Aktiv)

menu.main.active.background.color.hover `#cecece` [*burl*] Hintergrundfarbe (Aktiv Hover)

Hintergrundgrafik

menu.main.background.image [*burl*] Hintergrundbild-URL

menu.main.background.image.hover [*burl*] Hintergrundbild-URL (Hover)

16.4.2. Tabs

Beschriftung

menu.tab.caption.color `#666` [*color*] Textfarbe

menu.tab.caption.color.hover `#333` [*color*] Textfarbe (Hover)

menu.tab.active.caption.color [*color*] Textfarbe (Aktiv)

menu.tab.active.caption.color.hover [*color*] Textfarbe (Aktiv Hover)

Hintergrundfarbe

menu.tab.background.color `#e8e8e8` [*bcolor*] Hintergrundfarbe

menu.tab.background.color.hover `#fff` [*bcolor*] Hintergrundfarbe (Hover)

menu.tab.active.background.color [*bcolor*] Hintergrundfarbe (Aktiv)

menu.tab.active.background.color.hover [*bcolor*] Hintergrundfarbe (Aktiv Hover)

Hintergrundgrafik

menu.tab.background.image [*burl*] Hintergrundbild-URL

menu.tab.background.image.hover [*burl*] Hintergrundbild-URL (Hover)

16.4.3. Tab-Buttons

Beschriftung

menu.tab.button.caption.color `#ddd` [*color*] Textfarbe

menu.tab.button.caption.color.hover `#fff` [*color*] Textfarbe (Hover)

menu.tab.button.active.caption.color `#fff` [*color*] Textfarbe (Aktiv)

menu.tab.button.active.caption.color.hover `#fff` [*color*] Textfarbe (Aktiv Hover)

Hintergrundfarbe

menu.tab.button.background.color [*bcolor*] Hintergrundfarbe

menu.tab.button.background.color.hover `#666` [*bcolor*] Hintergrundfarbe (Hover)

menu.tab.button.active.background.color `#444` [*bcolor*] Hintergrundfarbe (Aktiv)

menu.tab.button.active.background.color.hover `#666` [*bcolor*] Hintergrundfarbe (Aktiv Hover)

Außenrahmen

menu.tab.button.border.style `solid` [*style*] Randstil

menu.tab.button.border.color `#aaa` [*color*] Randfarbe

menu.tab.button.border.color.hover `#bbb` [*color*] Randfarbe (Hover)

16.4.4. Spaltenköpfe

Beschriftung

table.head.caption.color `#666` [*color*] Textfarbe

table.head.caption.color.hover `#333` [*color*] Textfarbe (Hover)

table.head.active.caption.color `#333` [*color*] Textfarbe (Aktiv)

table.head.active.caption.color.hover `#333` [*color*] Textfarbe (Aktiv Hover)

Unterstreichung

table.head.border.bottom.style `solid` [*style*] Stil

table.head.border.bottom.color `#999` [*color*] Randfarbe

table.head.border.bottom.color.hover `#999` [*color*] Randfarbe (Hover)

table.head.active.border.bottom.color `#08f` [*color*] Randfarbe (Aktiv)

table.head.active.border.bottom.color.hover `#08f` [*color*] Randfarbe (Aktiv Hover)

Hintergrundfarbe

table.head.background.color `#cecece` [*bcolor*] Hintergrundfarbe

table.head.background.color.hover `#e8e8e8` [*bcolor*] Hintergrundfarbe (Hover)

table.head.active.background.color `#e8e8e8` [*bcolor*] Hintergrundfarbe (Aktiv)

table.head.active.background.color.hover `#efefef` [*bcolor*] Hintergrundfarbe (Aktiv Hover)

Hintergrundgrafik

table.head.background.image [*burl*] Hintergrundbild-URL

table.head.background.image.hover [*burl*] Hintergrundbild-URL (Hover)

16.4.5. Extras

Dropdown- & Listenmenüs

menu.dropdown.link.color `#555` [*color*] Textfarbe

menu.dropdown.link.color.hover `#000` [*color*] Textfarbe (Hover)

menu.dropdown.background.color `#f7f7f7` [*bcolor*] Hintergrundfarbe

menu.dropdown.background.color.hover `#e0e0e0` [*bcolor*] Hintergrundfarbe (Hover)

menu.dropdown.background.image [*burl*] Hintergrundbild-URL

menu.dropdown.background.image.hover [*burl*] Hintergrundbild-URL (Hover)

Ausgewählte Elemente

selection.font.color `#333` [*color*] Textfarbe

selection.font.2nd.color `#333` [*color*] Zweite Textfarbe

selection.link.color `#666` [*color*] Linkfarbe

selection.background.color `#def` [*bcolor*] Hintergrundfarbe

selection.border.width `1px` [*lengths*] Randbreite

selection.border.style `solid` [*style*] Randstil

selection.border.color `#08f` [*bcolor*] Randfarbe

selection.background.image [*burl*] Hintergrundbild-URL

16.5. Erweitert

16.5.1. Nachrichtendarstellung

Seitenleisten

messages.sidebar.alignment `left` [*custom*] Ausrichtung (links: „left“, rechts: „right“ oder oben: „top“)

messages.sidebar.text.alignment `center` [*align*] Textausrichtung

messages.sidebar.avatar.framed `0` [*boolean*] Gerahmte Ansicht

messages.sidebar.color.cycle `0` [*boolean*] Hintergrundfarbe abwechseln

messages.sidebar.divider.use `1` [*boolean*] Trennlinie

Nachrichtenfeld

messages.framed `0` [*boolean*] Gerahmte Ansicht

messages.color.cycle `1` [*boolean*] Hintergrundfarbe abwechseln

messages.boxes.background.color [*bcolor*] Hintergrundfarbe (Boxen)

messages.boxes.font.color [*color*] Textfarbe (Boxen)

messages.footer.alignment `right` [*custom*] Ausrichtung der Buttons (links: „left“ oder rechts: „right“)

16.5.2. Zusätzliche CSS-Deklarationen

Zusätzliche CSS-Deklarationen 1

user.additional.style.input1 [*custom*] Eigener CSS-Code 1

user.additional.style.input1.use `0` [*boolean*] Soll eigener CSS-Code 1 verwendet werden

Zusätzliche CSS-Deklarationen 2

user.additional.style.input2 [*custom*] Eigener CSS-Code 2

user.additional.style.input2.use `0` [*boolean*] Soll eigener CSS-Code 2 verwendet werden

16.5.3. Kommentare

Kommentare

user.comment `"WoltLab Basic"[...]` [*custom*] Kommentar zum Stil

16.6. Wertebereiche

Innerhalb der Auflistung der Stilvariablen wurde in eckigen Klammern immer angegeben, welcher Wertebereich für diese Angaben in Frage kommt. Diese sollen hier in alphabetischer Reihenfolge erklärt werden. Beachten Sie, dass es auch häufig möglich ist keine Angabe zu machen und dabei dennoch ein gewünschtes Resultat zu erzielen – da bei CSS das Prinzip der Vererbung häufig eine Rolle spielt. Auch kann es unter Umständen vorkommen, dass eine Angabe nicht wirksam ist, weil sie in einem Browser nicht unterstützt wird oder an anderer Stelle wieder überschrieben wird.

align

Hier ist die Ausrichtung von Text gemeint. Browser wie der Internet Explorer richten auch Elemente mit `text-align` aus:

`left` Links ausrichten

`center` Zentriert ausrichten

`right` Rechts ausrichten

`justify` Blocksatz - sollte nur mit Vorsicht angewendet werden und funktioniert dann nicht, wenn es um die Ausrichtung von Elementen geht.

Weitere Informationen: http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap16.html#heading-16.2%A0

balign

Für die Ausrichtung von Hintergrundbildern kombinieren Sie Angaben für die vertikale Ausrichtung (oben-unten) mit denen zur horizontalen Ausrichtung:

`top` Oben ausrichten

`bottom` Unten ausrichten

`center` Mittig ausrichten – kann sowohl für horizontale als auch vertikale Ausrichtung verwendet werden.

`left` Links ausrichten

`right` Rechts ausrichten

Geben Sie immer erst den Wert zur vertikalen und dann zur horizontalen Ausrichtung an, z. B. `top left` um ein Bild oben links auszurichten.

Weitere Informationen: http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap14.html#heading-14.2.1%A0

bcolor

Die Hintergrundfarbe kann mit den unter color beschriebenen Farbangaben gesteuert werden. Zusätzlich steht das Schlüsselwort `transparent` zur Verfügung, um den Hintergrund durchscheinend zu machen.

bfix

Um ein Hintergrundbild zu fixieren, geben Sie den Wert `fixed` an. Beachten Sie, dass ein fixierter Hintergrund die Bedienung der Seite verlangsamen kann (z. B. beim Scrollen).

boolean

Geben Sie eine `1` an für `true` und eine `0` für `false`.

brepeat

Hintergrundbilder können auch wiederholt (gekachelt) werden. Sie können dies über die folgenden Werte steuern:

`repeat` Bild wird in alle Richtungen wiederholt.

`repeat-x` Bild wird in X-Richtung (links-rechts) wiederholt.

`repeat-y` Bild wird in Y-Richtung (oben-unten) wiederholt.

`no-repeat` Bild wird einmal angezeigt, aber nicht wiederholt.

burl

Die URL von Hintergrundbildern wird über das Schlüsselwort `url()` angegeben. Innerhalb der Klammern steht der Pfad zu einer Bilddatei.

Weitere Informationen: http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap04.html#heading-4.3.4%A0

color

Farben können als Hexadezimalzahl oder in RGB-Notation angegeben werden. Zusätzlich steht eine Liste mit Schlüsselwörtern wie red, green oder blue zur Verfügung. Die Farbe rot kann also wie folgt angegeben werden:

```
#f00
#ff0000
rgb(255,0,0)
rgb(100%, 0%, 0%)
red
```

Weitere Informationen: http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap04.html#heading-4.3.6%A0

custom

Alle Variablen, die eine eigene sehr spezielle Angabe benötigen, wurden als custom deklariert. Hier finden Sie eine passende Beschreibung jeweils bei der entsprechenden Stilvariablen.

favicon

Bei der Angabe des Favoriten-Icons können Sie einen Namen aus der folgenden Liste angeben. Es handelt sich dabei um vorgefertigte Favicons:

- black
- blue
- blueExtra
- brown
- brownExtra
- darkBlue
- darkGreen
- darkRed
- darkViolet
- green
- greenExtra
- grey
- greyExtra
- lightBlue
- lightGreen
- lightGrey
- ochre
- orange
- pink
- red
- redExtra
- violet
- violetExtra
- yellow

Möchten Sie gerne ein eigenes Favicon verwenden, so lassen Sie die Variable leer und überschreiben Sie die Datei `favicon.ico` im Ordner der Endanwendung.

font

Bei der Angabe von Schriftarten gibt es zwei Möglichkeiten:

1. Name der Schriftart - dieser Name sollte in Anführungszeichen gesetzt sein. Es ist darauf zu achten, dass nur Schriften verwendet werden, die auf einem Großteil der Rechner installiert sind. Exotische Schriftarten funktionieren vielleicht auf dem eigenen Rechner, nicht jedoch auf denen Ihrer Benutzer.

2. Angabe einer generischen Familie: `serif`, `sans-serif`, `cursive`, `fantasy` und `monospace`

Mehrere Schriftarten sind durch Komma zu trennen. Es empfiehlt sich immer als letztes eine generische Schriftart anzugeben.

Weitere Informationen: http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap15.html#heading-15.2.2%A0

hide

Bestimmte Variablen sind dazu da Elemente auszublenden. Wenn die Variable leer ist, wird das Element angezeigt. Um es auszublenden verwenden Sie bitte den folgenden Code:

```
position: absolute; top: -9000px; left: -9000px;
```

lengths

Zur Angabe von Größen, Längen oder Breiten stehen Ihnen die Einheiten `em`, `ex`, `px` und `%` zur Verfügung. Von der Verwendung absoluter Größen wie `pt` oder `cm` wird abgeraten.

Mehr Informationen: http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap04.html#heading-4.3.2%A0

path

Pfadangaben zu Dateien und Ordnern sollten relativ zum WCF-Ordner gemacht werden.

style

Rahmen sind in ihrem Stil änderbar. Beachten Sie, dass nicht alle Browser die nachfolgenden Stile unterstützen:

- none
- hidden
- dotted
- dashed
- solid
- double
- groove
- ridge
- inset
- outset

Weitere Informationen: http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap08.html#heading-8.5.3%A0

Index

@, 28
#, 28

include, 32

sandbox, 33
section, 34

Template
append, 30
assign, 30
else, 32
elseif, 32
foreach, 34
Funktionen, 28
if, 32
include, 32
sandbox, 33
Kommentare, 28
Modifikatoren, 29
section, 34
Variablen, 27
@, 28
#, 28
zuweisen, 27