DynPG Template-Engine

Daniel Schliebner, ds-develop

Kontakt: mail@ds-develop.de

Stand: 29. Juli 2010

Inhaltsverzeichnis

Kapitel 1

Einführung

Die Templateengine von DynPG besteht im Wesentlichen aus der in der Datei

`(cms)/plugins/templates.php`

definierten Klasse `TTemplate`. Die in ihr enthaltenen Methoden werden innerhlab der Klasse durch Kommentare recht gut dokumentiert.

In jedem Plugin bzw. in der DynPG-Frontend-Engine selbst wird jeweils eine Instanz der Klasse `TTemplate` erzeugt. Standardmäßig lautet diese Instanz innerhalb von Plugins `$this->template`. Im folgenden werden wir nun dieses Property verwenden um Templates mit Inhalt zu füllen.

Die Engine kann im Prinzip vier semantisch verschiedene Platzhalter-Typen dynamisch ersetzen:

1. Variablen: simple Platzhalter für dynmaischen Inhalt, welcher durch den jeweiligen Anwender der Engine mit dynamischen Inhalt gefüllt werden.
2. Alternativen (`if`-Statements): um bestimmte Code-Teile nur in bestimmten Situationen anzeigen lassen zu können, dienen die `if`-Statements.
3. PHP-Codeblöcke: innerhalb dieser Blöcke ist es möglich PHP-Code auszuführen; zu beachten ist dabei jedoch, dass der Block nur *ein* Statement enthalten kann, d.h. mehrere Statements, getrennt durch Semikolons, sind nicht gestattet.
4. Schleifen/Subtemplates: um Listen oder allgemein wiederholte Aufzählungen implementieren zu können, sind Schleifen von hoher Wichtigkeit.

Im folgenden werden wir uns nun mit diesen Elementen beschäftigen und die Verwendung der Template-Klasse erläutern.

Für praktische Beispiele bzw. Erläuterung einiger DynPG-Eigener Platzhalter dienen auch diese Dokumentationen im Internet:

1. http://www.dynpg.ch/doku/ch04s04.html

2. http://www.dynpg.ch/doku/ch05s03.html
3. http://www.dynpg.ch/tool/upload/imgfile128.pdf

Die DynPG-Template Engine ist eine eigenständige Entwicklung und nutzt keinerlei Open-Source Engines oder Auszüge davon, versucht aber Standardtechniken zu implementieren und entsprechende Konzepte umzusetzen.

Kapitel 2

Platzhalter in Template-Dateien

Template-Dateien werden in DynPG-Projekten standardmäßig mit der Dateiendung `*.tpl` angelegt und liegen in Unterordnern der Form

```
templates/(template_set)/template.tpl
```

wobei also `template_set` ein durch den Verwender (Plugin, DynPG-Engine) vorgegebener Name eines Template-Sets ist.

2.1 Variablen

Variablen haben innerhalb von Template-Dateien die Form

```
{$IDENTIFIER}
```

wobei `IDENTIFIER` ein beliebig durch den Verwender der Engine vorgegebener Name ist und je nach Kontext unterschiedliche Werte enthalten kann. Prinzipiell gibt es zwei Typen von Variablen: *globale Variablen*, welche im gesamten Template verfügbar sind (durch `TTemplate::addVar()` angelegt) und *lokale Variablen*, welche als Parameter den Rendering-Methoden `TTemplate::fillTemplate()` bzw. `TTemplate::render()` mitgegeben werden können. Durch den Aufruf von

```
TTemplate::fillTemplate()
```

bzw.

```
TTemplate::render()
```

können die vorher definierten Variablen im Template durch ihren dynamischen Inhalt ersetzt werden. Genaueres werden wir zu diesem Thema in Kapitel 3 erklären.

2.2 Alternativen

Sogenannte `if`-Statements sind sicherlich ein enorm wichtiger Bestandteil von Template-Engines, weil sie ein hohes Maß an Komplexität ermöglichen. Mit ihrer Hilfe können bestimmte Bereiche im Template dynamisch beim Rendering aus- bzw. eingeblendet werden.

Die Syntax in der DynPG Template Engine sieht dabei wiefolgt aus:

```
{$# ASSIGNMENT #} ... {$# End #}
```

Hierbei meint `ASSIGNMENT` einen beliebigen aber booleschen Ausdruck. Insbesondere dürfen diese `ASSIGNMENT`s auch PHP-Code enthalten! Wichtig ist nur, dass ein solches `ASSIGNMENT` einen booleschen Wert zurück liefert (d.h. entweder `true` oder `false` bzw. einen in PHP äquivalenten Wert wie 0 oder `null`).

Beispiel:

```
{$# {$IS_PUBLIC} #} Bereich ist öffentlich {$# End #}
```

2.3 PHP-Codeblöcke

Um beispielsweise dynamisch eingefügten Inhalt noch nachträglich im Template zu bearbeiten, existieren PHP-Codeblöcke. Innerhlab dieser Blöcke dürfen Sie PHP-Code und Template-Variablen vermischen. Dies ermöglicht Ihnen komplexe Nachbearbeitungen der dynmaisch durch die Engine eingefügten Daten (z.B. Abschneiden nach einer bestimmten Anzahl an Zeichen, Großbuchstabenumwandlung usw.). Die Syntax sieht wiefolgt aus:

```
{$#PHP STATEMENT #}
```

Wie bereits erwähnt darf jedoch `STATEMENT` nur ein einzelnes PHP-Statement sein.

Beispiel:

```
{$#PHP strtoupper('{$TITLE_OF_ARTICLE}') #}
```

nicht aber etwa

```
{$#PHP strtoupper('{$TITLE_OF_ARTICLE}'); print 'Hallo Welt'; #}
```

2.4 Schleifen/Subtemplates

Schleifen sollen bestimmte ausgezeichnete Bereiche im Template wiederholen und bei jedem Durchlauf, den entsprechenden Block mit dynamischen Inhalt füllen. Innerhalb der Template-Klasse `TTemplate` heißen diese „Schleifen“ auch Subtemplates, weil sie im Wesentlichen ebenfalls wie eigene kleine Templates interpretiert werden. Die Syntax sieht wiefolgt aus:

```
<!-- RepeatedListBegin name="NAME_OF_LOOP" -->
    ...
<!-- RepeatedListFinish -->
```

Hierbei steht `NAME_OF_LOOP` für den „Namen“ der Schleife bzw. des Subtemplates. Dieser wird später dann bei der Implementierung im Plugin oder der DynPG-Engine referenziert. Den Code innerhlab eines solchen Subtemplates erhalten Sie mit der Methode

`TTemplate::getSubTemplate()`

Das vorbereiten bzw. aufbereiten der Daten für eine Schleife bzw. die Implementierung im Plugin stellt sich dabei als am schwierigsten von den vier soeben vorgestellten Platzhalter-Arten dar, ist aber im Prinzip auch schnell erklärt. Mehr dazu folgt nun im Kaptiel 3.

Kapitel 3

Verwendung

Zum besseren Verständnis sollten Sie bereits Kapitel 2 durchgearbeitet haben, um die hier verwendeten Begriffe besser verstehen zu können.

3.1 Variablen hinzufügen

Wie bereits erwähnt existieren zwei Arten von Variablen: globale und lokale. Globale Variablen können einfach mittels der Methode

```
TTemplate::addVar($id, $var, $overwrite = true)
```

hinzugefügt werden. Dabei ist `$id` der Name der Variablen, `$var` ihr Inhalt und `$overwrite` gibt an, ob eine möglicherweise beretis bestehende, gleichnamige Variable überschrieben werden soll. Globale Variablen werden im Template durch ihren dynmaischen Inhalt `$var` ersetzt, sobald die Methode

```
TTemplate::fillTemplate($template, $file, $subRows)
```

aufgerufen wird und `subRows` ein leeres Array ist bzw. nicht in der Parameterliste angegeben wird.

Lokale Variablen sind Variablen, welche beim Aufruf von `TTemplate::fillTemplate` im Array `$subRows` mitgegeben werden. Beispiele und nährere Informationen zu den soeben angesprochenen Methoden erhalten Sie im nun folgenden Abschnitt.

3.2 Template rendern

Sie haben nun ihrem Template ein paar globale Variablen zuweisen können (anbieten tuen sich hier z.B. Informationen wie aktuelle IP-Adresse, aktueller Seitenpfad etc.). Nun wollen wir einmal ein gegebenes Template rendern, d.h. die in ihm enthaltenen

Platzhalter aus Kapitel 2 durch dynamischen Inhalt ersetzen (in diesem Abschnitt werden die Schleifen noch ausgenommen).

Für das Umsetzen von Alternativen und PHP-Blöcken brauchen Sie natürlich nichts tuen. Sie müssen (wenn wir Schleifen vorerst außen vor lassen) lediglich der Template-Engine globale und lokale Variablen, sowie ihren Inhalt mitteilen, damit diese dann beim Rendering ersetzt werden können.

Als Beispielanwendungsfall nehmen wir einmal an, wir hätten ein Plugin vorliegen, in welchem wir nun ein simples Formulartemplate laden und mit Inhalt füllen wollen. Dieses Template liegt nun unter

`(cms)/plugins/(pluginname)/templates/(templateset)/formular.tpl`.

Wir wollen nun einmal annhemen, das Plugin heiße `DPGplugin` und das aktuelle Templateset `default`. Das Template `formular.tpl` enthalte einmal folgenden Inhalt:

```
<form action="{$Main->Self}" method="post">
    <input type="text" name="email" value="{$EMAIL}" />
    <input type="submit" value="{$SUBMIT_BTN_VALUE}" />
</form>
```

Zum rendern dieses kleinen Templates benötigen wir dann nur folgenden PHP-Code:

```
$this->template->addVar('Main->Self', $_SERVER['PHP_SELF']);
$content = $this->template->fillTemplate(
    dirname(__FILE__) . '/templates/default/formular.tpl',
    true,
    Array(
        'EMAIL' => 'email@domain.com',
        'SUBMIT_BTN_VALUE' => 'Abschicken'
    )
);
$content = $this->template->fillTemplate($content, false);
```

Die Variable `$content` enthält dann den *gerenderten* Inhalt des Templates und kann z.B. zum Test durch `print $content;` ausgegeben werden. Die Variable `Main->Self` ist hier also eine globale und `EMAIL` und `SUBMIT_BTN_VALUE` sind lokale Variablen.

Der doppelte Aufruf von `fillTemplate()` ist notwendig, da beim ersten Aufruf *nur die lokalen Variablen* ersetzt werden (da der dritte Parameter nicht weggelassen bzw. ein leeres Array ist). Alternativ zur letzten Zeile wäre auch der Aufruf

```
$content = $this->template->render($content, false);
```

möglich, mehr dazu jedoch später. Noch eine kleine Anmerkung: der zweite Parameter gibt an, ob der erste Parameter ein String zum rendern (`false`) oder ein Dateipfad zu einer Templatedatei (`true`) ist. Man betrachte hierzu auch die Kommentare der Methoden innerhalb der Klasse `TTemplate`.

Das war letztlich auch schon (fast) alles, was es zum rendern *ohne Schleifen* zu bemerken gibt.

3.3 Schleifen implementieren: Subtemplates

Wie bereits erwähnt sind die „Schleifen“ im wesentlichen Subtemplates. Dies spiegelt sich auch in der Implementierung wieder, wo wir die einzelnen Subtemplates auslesen werden und die „Schleifen“-Funktion (optional) in den Code implementieren. Die Engine macht also aus den in Kapitel 2 beschriebenen Subtemplate-Blöcken nicht „automatisch“ Schleifen.

Wir erweitern den in Abschnitt 3.2 geschilderten Anwendungsfall und das Template `formular.tpl` nun um ein kleines Subtemplate. Der Inhalt des Template sehe daher wiefolgt aus:

```
<form action="{$Main->Self}" method="post">
    <!-- RepeatedListBegin name="formfields" -->
        <input type="text" name="{$NAME}" value="{$DEFAULT_VALUE}" />
    <!-- RepeatedListFinish -->

    <input type="submit" value="{$SUBMIT_BTN_VALUE}" />
</form>
```

Wir wollen also die Formularfelder dynamisch einfügen lassen. Der Einfachheit halber werden wir dazu einfach ein Array `$form_fields` annehmen, welches wiefolgt deklariert wird:

```
$form_fields = Array(
    'name'  => 'Max Mustermann',
    'email' => 'email@domain.com'
);
```

Im folgenden geben wir Ihnen nun ersteinmal den Code zur Implementierung der Schleife, welcher sicher nicht ganz genauso aussehen muss, sich jedoch in seiner Art und Weise bisher so bewährt hat. Anschließend folgen zahlreiche Erläuterungen zu den einzelnen Code-Abschnitten:

```
$this->template->addVar('Main->Self', $_SERVER['PHP_SELF']);

$template     = dirname(__FILE__) . '/templates/default/formular.tpl';
$listing      = '';
$list_index   = 0;
$template_row = $this->template->getSubTemplate(
    'formfields',
    $template
);

foreach ($form_fields as $name => $default_value) {
    $list_index++;
    $listing .= $this->template->fillTemplate(
        $template_row,
        false,
        Array(
            'NAME' => $name,
            'DEFAULT_VALUE' => $default_value,
            'SUBMIT_BTN_VALUE' => 'Abschicken',
            'LIST_INDEX' => $list_index
        )
    );
}

$this->template->addVar('LIST_INDEX', $list_index);
```

```
$content = $this->template->fillTemplate(
    $this->template->replaceSubTemplate('formfields', $listing, $template), false
);
```

Zur Erläuterung: zuerst legen wir vier Variablen an, welche den Code ein wenig schlanker werden lassen:

1. `$template` enthält den Pfad zum Template,
2. `$listing` enthält später den wiederholten Inhalt innerhalb des Subtemplates.
3. `$list_index` enthält die Zahl der Schleifendurchläufe; es hat sich gezeigt, dass dies eine sehr nützliche Variable ist, weshalb wir sie standardmäßig in jeder Auflistung verwenden,
4. `$template_row` enthält den *Code des Subtemplates*
   (hier also: `<input type="text" name="{$NAME}" value="{$DEFAULT_VALUE}" />`).

Das Prinzip hinter diesem kleinen Algorithmus ist das folgende: wir lesen zuerst den Code des Subtemplates ein, d.h. in unserem Fall, den Code, welcher für jedes Fomrularelement im Array `$form_fields` gerendert werden soll. Diesen durch `$this->template->fillTemplate()` gerenderten Code fügen wir dann einer Listenvariable `$listing` an, welche nach Durchlauf der `for`-Schleife schließlich den (hier 2-mal) wiederholten, gerenderten Subtemplate-Code enthält. Nun müssen wir das Subbtemplate mit dem Namen `formfields` lediglich noch mit dem soeben generierten `$listing` ersetzen. So emulieren wir im Prinzip eine Schleife im Template.

Es wird aber natürlich auch deutlich, dass die Subtemplates logischerweise *nicht zwingend* als Schleifen genutzt werden müssen, sondern unter Umständen auch andere Anwendungsmöglichkeiten bieten können. Die Zeile

```
$this->template->addVar('LIST_INDEX', $list_index);
```

ist notwendig, weil wir `LIST_INDEX` vorher stets als lokale Variable deklariert haben. Mithin ist sie nach dem Durchlauf der Schleife nicht mehr im Namespace der Template-Engine, sodass wir sie, um eine Verwendung auch *nach* dem Subtemplate `formfields` zu ermöglichen, als global deklarieren.

Zuletzt gehen wir nocheinmal kurz auf die beidem Methoden `getSubTemplate()` und `replaceSubTemplate()` ein: `getSubTemplate()` liest aus einer im ersten Parameter angegebenen Templatedatei oder Templatestring (selbes Prinzip wie in `fillTemplate()`) einen im dritten Parameter angegebenes Subtemplate und gibt dessen Code zurück. Anschließend können Sie nun mit diesem Code arbeiten und ggf. spezielle, vom Gesamttemplate unabhängige Bearbeitungen durchführen. Nach Bearbeitung des Inhalts können Sie dann durch Verwendung der Methode `replaceSubTemplate()` den Subtemplate-Block mit dem im ersten Parameter spezifizierten Namen ersetzen.

3.4 AutoRendering

Noch zu erwähnen ist die Möglichkeit, innerhalb der Template-Engine das automatische Rendering innerhalb von bestimmten Platzhaltern zu verändern (genauer: in Alternativen und PHP-Blöcken). Innerhalb dieser Blöcke stellt sich die Frage, wann in ihnen enthaltene Variablen bzw. verschachtelte `if`-Statements gerendert werden sollen. Diese Frage ist von der Semantik abhängig und daher existiert in der Klasse `TTemplate` ein Property `TTemplate::autoRender` mit zwei Setter-Methoden `autoRenderOn()` und `autoRenderOff()`.

Steht `TTemplate::autoRender` auf `true` (es wurde aktiviert durch `autoRenderOn()`), so wird versucht, solche Platzhalter innerhalb der Alternativen und PHP-Blöcke direkt beim Rendering zu ersetzen. Dies hat jedoch zur Folge, dass eventuell nicht im Namespace enthaltene Variablen zu Fehlern führen. Ist also `TTemplate::autoRender` aktiviert, so haben Sie unbedingt darauf zu achten, ob beim Rendering mit der Methode `fillTemplate()` alle Variablen im Namespace der Templateendinge enthalten sind!

Es hat sich daher eingebürgert, `TTemplate::autoRender` zu *deaktivieren* (d.h. durch `autoRenderOff()`), nachdem eine Instanz der Klasse `TTemplate` erzeugt wurde.

Es bleibt nun noch die Frage zu klären, wie eventuell noch fehlende Variablen final ersetzt werden können. Dazu existiert in der Klasse `TTemplate` die Methode `render()`. Sie ist im Prinzip identisch mit `fillTemplate()`, jedoch mit dem Unterschied, das `TTemplate::autoRender` aktiviert wird, um ein finales Rendering zu erreichen. Sie sollten daher `render()` am Ende ihrer Templateverarbeitung aufrufen, sofern Sie `TTemplate::autoRender` deaktiviert haben.

Kapitel 4

Ergänzungen

Templatesets können prinzipiell durch den Benutzer Ihres Plugins jederzeit geändert werden. Dies geschieht durch Nutzung der globalen DynPG Instanz bei der Implementierung. Ein Benutzer ruft dazu lediglich die Methode `SetParam_PlugIn_Template`, z.B. wiefolgt:

```
$GLOBALS["DynPG"]->SetParam_PlugIn_Template('DPGplugin' , 'default');
```

Sie sollten daher beim Lesen von Templates darauf achten, dass Sie Templatesets *dynamisch* einbinden, d.h. auf das aktuell eingestellte Templateset zugreifen. In der Plugin-Schablone `DPGplugin` wird dazu die aktuelle Instanz von DynPG im Property `$dynpg_instance` gespeichert. Sie erhalten dann Zugriff auf das aktuelle Templateset durch folgenden Code (innerhalb Ihrer Pluginklasse):

```
$this->dynpg_instance->_PlugIn['dpgplugin']['template']
```

für ein Plugin mit dem Namen `DPGplugin`. Die generische Lösung aus Abschnitt 3.2 sähe dann wiefolgt aus:

```
$this->template->addVar('Main->Self', $_SERVER['PHP_SELF']);
$content = $this->template->fillTemplate(
    dirname(__FILE__) . '/templates/'
        . $this->dynpg_instance->_PlugIn['dpgplugin']['template'] .
        '/formular.tpl',
    true,
    Array(
        'EMAIL' => 'email@domain.com',
        'SUBMIT_BTN_VALUE' => 'Abschicken'
    )
```

```
);
$content = $this->template->fillTemplate($content, false);
```