Dokumentation der TagLibTags

Folgende TagLib-Tags stehen zur Verfügung. In nicht selbstschließenden Tags sind alle HTML- Tags und -Formatierungen erlaubt. Die Tags können mit den üblichen HTML-Attributen ausgestattet werden, sofern diese für die Ausgabe oder Funktion relevant sind.

1. Core

1.1. Addtaglib

TagLibs sind nur innerhalb eines Dokuments des Objektbaums gültig. Um eine selbst implementierte TagLib im aktuellen Dokument einbinden zu können wird folgender Tag verwendet. Dieser sollte möglichst am Anfang des Dokuments plaziert sein, da Tags erst nach erfolgreicher Einbindung einer TagLib geparst werden können.

```
<core:addtaglib namespace="" prefix="" class="" />
```

Beschreibung der Attribute:

- **namespace**: Ein mit "::" getrennter Namespace-Pfad. (**Zeichen**: `[A-Za-z0-9_-:]`)
- **prefix**: XML-Prefix (**Zeichen**: `[a-z]`)
- **class**: XML-Klasse (**Zeichen**: `[a-z]`)

Die vom Entwickler erzeugte TagLib-Klasse muss dir Form "<prefix>_taglib_<class>" haben. Die innerhalb der Klasse zu implementierenden Methoden sind der API-Dokumentation zu entnehmen. In der Regel wird das Tag für die Einbindung der Form-TagLib verwendet. Eigene Taglibs machen vor allem dann Sinn, wenn eigene abgetrennte Bereiche einer Applikation mehrfach verwendbar gemacht werden wollen (z.B. Warenkorb).

1.2. Importdesign

Möchte der Template-Bauer an einer definierten Stelle ein weiteres Document einfügen, so kann der Tag

```
<core:importdesign namespace="" template="" [incparam=""] />
```

Beschreibung der Attribute:

- **namespace**: Ein mit "::" getrennter Namespace-Pfad. (**Zeichen**: `[A-Za-z0-9_-:]`)
- **template**: Name Eindeutige ID innerhalb eines Formulars. (**Zeichen**: `[A-Za-z0-9-_:]`)
- **incparam**: URL-Parameter der abgefragt wird, welches Template eingebunden werden soll.

verwendet werden. Wird bei der Definition des Tags ein sog. pagepart-Template definiert, kann ein

Template abhängig vom URL-Parameter eingebunden werden. Standard-Parameter ist "pagepart", dieser kann jedoch mit dem optionalen Attribut "incparam" angepasst werden. Beispiel für die Einbindung mit pagepart-Parameter:

```
<core:importdesign namespace="sites::testsite::pres::templates"
template="[pagepart = test]" />
<core:importdesign namespace="sites::testsite::pres::templates"
template="[cmsmodule = test]" incparam="cmsmodule" />
```

Das eingebundene Template erbt automatisch die Eigenschaften des Vater-Objekts. Insbesondere Context und Sprache werden übertragen.

1.3. Setproperty

Um eine Eigenschaft des aktuellen Documents im Template setzen zu können steht dem, Entwickler der "core:setproperty"-Tag zur Verfügung. Typischer Anwendungsfall ist es dein Context oder die Sprache eines Documents mit diesem Text zu setzen um auf eine anderssprachige Seite zu wechseln.

```
<core:setproperty name="" value="" />
```

Eine Liste der Eigenschaften eines von der Klasse "Document" erbenden Klasse entnehmen Sie bitte der API-Dokumentation.

Um das Tag verwenden zu können muss die TagLib "core:setproperty" erst via

```
<core:addtaglib namespace="core::pagecontroller" prefix="core"
class="setproperty" />
```

eingebunden werden.

Beschreibung der Attribute:

- **name**: Name der zu befüllenden Eigenschaft. (**Zeichen**: `[A-Za-z0-9_-]`)
- **value**: Wert der Eigenschaft. (**Zeichen**: `[A-Za-z0-9-_:.]`)

1.4. Setattribute

Der Tag "core:setattribute" verhält sich analog zu "core:setproperty", nur wird bei diesem der Wert eines Attributes des Dokuments gesetzt.

```
<core:setattribute name="" value="" />
```

Um das Tag verwenden zu können muss die TagLib "core:setattribute" erst via

```
<core:addtaglib namespace="core::pagecontroller" prefix="core"
class="setattribute" />
```

eingebunden werden.

Beschreibung der Attribute:

- **name**: Name des Attributes. (**Zeichen**: `[A-Za-z0-9_-]`)
- **value**: Wert des Attributes. (**Zeichen**: `[A-Za-z0-9-_:.]`)

2. Html

2.1. Placeholder

Um in einem DocumentController Inhalte dynamisch füllen zu können gibt es den html:placeholder- Tag.

```
<html:placeholder name="" />
```

Beschreibung der Attribute:

- **name**: Name des Platzhalters. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)

Um einen Platzhalter mit Inhalt zu füllen kann man in einem DocumentController folgendes einbauen:

```
$this->setPlaceHolder('NameDesPlatzhalters','ContentDesPlatzHalters');
```

2.2. Template

Um weitere Elemente für die Ausgabe-Steuerung definieren zu können gibt es so genannte Templates. Diese können innerhalb eines DocumentControllers manipuliert und verarbeitet werden. In einem Template gibt es wiederum Platzhalter, die dynamisch gesetzt werden können.

```
<html:template name="">
  [<template:placeholder name="" />]
</html:template>
```

Beschreibung der Attribute:

- **name**: Name des Templates bzw. des Platzhalters innerhalb des Templates. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)

Zugriff auf ein Template gewährt die private Methode "__getTemplate()" eines DocumentControllers. Mit Hilfe dieser Funktion kann eine Referenz auf ein Template-Objekt zurückgeliefert werden. Ein typischer Anwendungsfall sieht so aus:

Template:

```
<@controller namespace="" file="" class="" @>
<html:placeholder name="Content" />

<html:template name="MyTemplate">
  <template:placeholder name="MyPlaceHolder">
</html:template>
```

Controller:

```
class myDocumentController extends baseController
{
   function myDocumentController(){
   }

   function transformContent(){
      $Template__MyTemplate = &$this->__getTemplate('MyTemplate');
      $Template__MyTemplate->setPlaceHolder('MyPlaceHolder','MyContent');
      $this->setPlaceHolder('Content',$Template__MyTemplate-
>transformTemplate());
   }
}
```

Hier wird in der XML-Datei ein Document-Controller, ein HTML-Platzhalter und ein Template mit zugehörigem Platzhalter definiert. Im Controller kann man nun mittels der Methoden *__getTemplate()* eine Referenz auf das Template holen und mit *setPlaceHolder()* den Platzhalter im Template füllen. Anschließend wird das Template transformiert und die Ausgabe in den HTML-Platzhalter mittels *setPlaceHolder()* eingesetzt. Es ist möglich mehrere Templates auf einer Seite zu platzieren. Oft wird das für die Ausgabe von unterschiedlichen Meldungen bei unterschiedlichen Events eingesetz.

2.3. Form

Zur dynamischen Generierung und Handhabung von Formularen gibt es das html:form-Tag. Mit diesem können Formulare innerhalb eines Dokuments beschreiben werden. Ähnlich wie Templates sind diese im DocumentController verfügbar uns können entsprechend in das Dokument eingesetzt werden.

```
<html:form name="">
  ...
</html:form>
```

Beschreibung der Attribute:

- **name**: Name des Formulars. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)

Innerhalb eines Formulars stehen folgende Tags zu Verfügung. Diese können in unterschiedlichen Kombinationen auftreten. Für ein Anwendungsbeispiel wird auf das Kontaktformular der Beispiel-Seite auf die API-Dokumentation verwiesen.

Des Weiteren stehen für Eingabe-Felder folgende Attribute zur Aktivierung und Konfiguration der Validierung von Benutzereingaben zur Verfügung (z.B. bei einer Textarea):

```
<form:area name="" [..] [validate="" button="" [validator=""]]/>
```

Beschreibung der Attribute:

- **validate**: Ist der Wert des Attributes mit "true" gefüllt, so wird der Eingabe- Werte beim Absenden überprüft. Fehlt das Attribut oder ist es mit dem Wert "false" versehen wird keine Validierung vorgenommen. Der Wert "true" bedingt jedoch das Attribut "button". (**Zeichen**: `[true|false]`)
- **button**: Name des Buttons, bei dessen Klicken das Feld validiert werden soll. (**Zeichen**: `[A-Za-z0-9-_]`)
- **validator**: Name des Validators. Standard ist "Text" und prüft auf Vorhandensein von mind. einem beliebigen Zeichen. Die hier möglichen Validatoren ergeben sich aus den Methoden der Klasse "myValidator" durch weglassen des Keywords "validate". Beispiel: Möchte eine E-Mail validiert werden, so ist hierfür die Methode "validateEMail" zuständig. Im Feld "validate" kann damit ein "EMail" notiert werden. Sollen weitere Validatoren möglich gemacht werden muss die Klasse "myValidator" erweitert werden. (**Zeichen**: `[A-Za-z]`)

2.3.1. Textarea

```
<form:area name="" [id=""] [class=""] [style=""] />
```

Beschreibung der Attribute:

- **name**: Name des Textarea. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)
- **id**: Eindeutige ID innerhalb eines Formulars. (**Zeichen**: `[A-Za-z0-9-_]`)
- **class**: CSS-Klasse des Feldes.
- **style**: CSS-Klasse des Feldes.

2.3.2. Textarea mit Content

```
<form:area name="" [id=""] [class=""] [style=""]>
</form:area>
```

Beschreibung der Attribute:

- **name**: Name des Textarea. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)
- **id**: Eindeutige ID innerhalb eines Formulars. (**Zeichen**: `[A-Za-z0-9-_]`)
- **class**: CSS-Klasse des Feldes.
- **style**: CSS-Klasse des Feldes.

2.3.3. Button

```
<form:button name="" [id=""] [class=""] [style=""] />
```

Beschreibung der Attribute:

- **name**: Name des Buttons. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)
- **id**: Eindeutige ID innerhalb eines Formulars. (**Zeichen**: `[A-Za-z0-9-_]`)
- **class**: CSS-Klasse des Feldes.
- **style**: CSS-Klasse des Feldes.

2.3.4. Checkbox

```
<form:checkbox name="" [id=""] [class=""] [style=""] [checked=""] />
```

Beschreibung der Attribute:

- **name**: Name der Checkbox. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)
- **id**: Eindeutige ID innerhalb eines Formulars. (**Zeichen**: `[A-Za-z0-9-_]`)
- **class**: CSS-Klasse des Feldes.
- **style**: CSS-Klasse des Feldes.
- **checked**: Zeigt an, ob der Radio-Button vorselektiert ist.

2.3.5. Datums-Control

```
```

```
<form:date name="" [id=""] [class=""] [style=""] [yearrange=""]
[offsetnames=""]/>
```

Beschreibung der Attribute:

- **name**: Name des Datum-Controlls. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)
- **id**: Eindeutige ID innerhalb eines Formulars. (**Zeichen**: `[A-Za-z0-9-_]`)
- **class**: CSS-Klasse des Feldes.
- **style**: CSS-Klasse des Feldes.
- **yearrange**: Range des Jahres-Feldes. Beispiel: `1990-2007`. (**Zeichen**: `[0-9-]`)
- **offsetnames**: Namen der Felder für Tag, Monat und Jahr. Einzelne Felder müssen durch ";" getrennt werden. Beispiel: `Tag;Monat;Jahr`. (**Zeichen**: `[A-Za-z]`)

2.3.6. Dateiupload-Feld

```
<form:file name="" [id=""] [class=""] [style=""] />
```

Beschreibung der Attribute:

- **name**: Name des Dateiupload-Felds. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)
- **id**: Eindeutige ID innerhalb eines Formulars. (**Zeichen**: `[A-Za-z0-9-_]`)
- **class**: CSS-Klasse des Feldes.
- **style**: CSS-Klasse des Feldes.

2.3.7. Hidden-Feld

```
<form:hidden name="" value="" />
```

Beschreibung der Attribute:

- **name**: Name des Hidden-Felds. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)
- **value**: Wert des Hidden-Feldes.

2.3.8. Multiselect-Feld

```
<form:multiselect name="" [id=""] [class=""] [style=""] />
```

Beschreibung der Attribute:

- **name**: Name des Text-Felds. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)
- **id**: Eindeutige ID innerhalb eines Formulars. (**Zeichen**: `[A-Za-z0-9-_]`)
- **class**: CSS-Klasse des Feldes.
- **style**: CSS-Klasse des Feldes.

2.3.9. Multiselect-Feld mit Optionen

```
<form:multiselect name="" [id=""] [class=""] [style=""]>
  <select:option value="" [selected=""]></select:option>
</form:multiselect>
```

Beschreibung der Attribute:

- **name**: Name des Text-Felds. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)
- **id**: Eindeutige ID innerhalb eines Formulars. (**Zeichen**: `[A-Za-z0-9-_]`)
- **class**: CSS-Klasse des Feldes.
- **style**: CSS-Klasse des Feldes.
- **value**: Wert der Option.

2.3.10. Passwort-Feld

```
<form:password name="" [id=""] [class=""] [style=""] />
```

Beschreibung der Attribute:

- **name**: Name des Passwort-Felds. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)
- **id**: Eindeutige ID innerhalb eines Formulars. (**Zeichen**: `[A-Za-z0-9-_]`)
- **class**: CSS-Klasse des Feldes.
- **style**: CSS-Klasse des Feldes.

2.3.11. Platzhalter

```
<form:placeholder name="" />
```

Beschreibung der Attribute:

- **name**: Name des Hidden-Felds. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)

2.3.12. Radio-Button

```
<form:radio name="" id="" [class=""] [style=""] [checked=""]/>
```

Beschreibung der Attribute:

- **name**: Name des Radio-Buttons. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)
- **id**: Eindeutige ID innerhalb eines Formulars. (**Zeichen**: `[A-Za-z0-9-_]`)
- **class**: CSS-Klasse des Feldes.
- **style**: CSS-Klasse des Feldes.
- **value**: Wert der Option.
- **checked**: Zeigt an, ob der Radio-Button vorselektiert ist.

2.3.13. Select-Feld

```
<form:select name="" [id=""] [class=""] [style=""] />
```

Beschreibung der Attribute:

- **name**: Name des Text-Felds. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)
- **id**: Eindeutige ID innerhalb eines Formulars. (**Zeichen**: `[A-Za-z0-9-_]`)
- **class**: CSS-Klasse des Feldes.
- **style**: CSS-Klasse des Feldes.

2.3.14. Select-Feld mit Optionen

```
<form:select name="" [id=""] [class=""] [style=""]>
  <select:option value="" [selected=""]></select:option>
</form:select>
```

Beschreibung der Attribute:

- **name**: Name des Text-Felds. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)
- **id**: Eindeutige ID innerhalb eines Formulars. (**Zeichen**: `[A-Za-z0-9-_]`)
- **class**: CSS-Klasse des Feldes.
- **style**: CSS-Klasse des Feldes.
- **value**: Wert der Option.

2.3.15. Text-Feld

```
<form:text name="" [id=""] [class=""] [style=""]/>
```

Beschreibung der Attribute:

- **name**: Name des Text-Felds. Über den Namen kann auf das Element zugegriffen werden. (**Zeichen**: `[A-Za-z0-9-_]`)
- **id**: Eindeutige ID innerhalb eines Formulars. (**Zeichen**: `[A-Za-z0-9-_]`)
- **class**: CSS-Klasse des Feldes.
- **style**: CSS-Klasse des Feldes.

2.3.16. Validator

```
<form:validate validator="" button="" field="" [type=""] [msginputreq=""]
[msginputwrg=""] />
```

Beschreibung der Attribute:

- **validator**: Name des Validators, mit dem der Wert geprüft werden soll. (**Zeichen**: `[A-Za-z]`)
- **button**: Name des Button des Formulars.
- **field**: Name des Feldes, das validiert werden soll.
- **type**: Ausgabe-Typ des Validators. Bei "text" gibt der Validator-Tag einen in der Konfigurations-Datei angegebenen Text aus, bei "css" eine CSS-Formatierung. (**Werte:** `text|css`).
- **msginputreq**: Eintrag der Konfigurations-Datei, der bei Angabe des "type"s "text" statt der Standard-Meldung ausgegeben werden soll um anzuzeigen, dass das mit "field" gekennzeichnete Feld nicht gefüllt wurde. Wird der Parameter nicht angegeben werden Standard-Werte verwendet. (**Zeichen:** `[A-Za-z0-9_-]`).
- **msginputwrg**: Eintrag der Konfigurations-Datei, der bei Angabe des "type"s "text" statt der Standard-Meldung ausgegeben werden soll um anzuzeigen, dass das mit "field" gekennzeichnete Feld mit dem falschen Wert gefüllt wurde. Wird der Parameter nicht angegeben werden Standard-Werte verwendet. (**Zeichen:** `[A-Za-z0-9_-]`).

2.3.17. Validator-Gruppe

```
<form:valgroup name="">
  <valgroup:validate validator="" button="" field="" [type=""]
[msginputreq=""] [msginputwrg=""] />
  [<valgroup:placeholder name="" />]
</form:valgroup>
```

Beschreibung der Attribute:

- **name**: Name der Validator-Gruppe. Es kann mehrere Gruppen in einem Formular geben.
- **validator**: Name des Validators, mit dem der Wert geprüft werden soll. (**Zeichen**: `[A-Za-z]`)
- **button**: Name des Button des Formulars.
- **field**: Name des Feldes, das validiert werden soll.
- **type**: Ausgabe-Typ des Validators. Bei "text" gibt der Validator-Tag einen in der Konfigurations-Datei angegebenen Text aus, bei "css" eine CSS-Formatierung. (**Werte:** `text|css`).
- **msginputreq**: Eintrag der Konfigurations-Datei, der bei Angabe des "type"s "text" statt der Standard-Meldung ausgegeben werden soll um anzuzeigen, dass das mit "field" gekennzeichnete Feld nicht gefüllt wurde. Wird der Parameter nicht angegeben werden Standard-Werte verwendet. (**Zeichen:** `[A-Za-z0-9_-]`).
- **msginputwrg**: Eintrag der Konfigurations-Datei, der bei Angabe des "type"s "text" statt der Standard-Meldung ausgegeben werden soll um anzuzeigen, dass das mit "field" gekennzeichnete Feld mit dem falschen Wert gefüllt wurde. Wird der Parameter nicht angegeben werden Standard-Werte verwendet. (**Zeichen:** `[A-Za-z0-9_-]`).

Optional besitzt die Validator-Gruppe Platzhalter, die mit der Methode *setPlaceHolder()* auf das Validator-Gruppen.Objekt gesetzt werden können. Attribute sind den *html:placeholder*-Tag zu entnehmen.

3. Document

3.1. createcontentobject

```
<document:createcontentobject requestparam="" defaultvalue="" />
```

Beschreibung der Attribute:

- **requestparam**: URL-Parameter, der angibt, welche Content-Datei gezogen wird. (**Zeichen**: `[A-Za-z0-9]`)
- **defaultvalue**: Default-Wert für den Parameter *requestparam*.

4. DocumentController

Um einem Dokument einen Controller zu deklarieren muss folgender Tag am Anfang des Templates platziert werden:

```
<@controller namespace="" file="" class="" @>
```

Beschreibung der Attribute:

- **namespace**: Namespace-Pfad zum Controller. (**Zeichen**: `[A-Za-z0-9:]`)
- **file**: Datei-Namen in dem der Controller residiert. (**Zeichen**: `[A-Za-z0-9_]`)
- **class**: Klassen-Namen des Controllers. (**Zeichen**: `[A-Za-z0-9_]`)

Die Implementierung des DocumentControllers muss immer vom *baseController* ableiten. Dieser ist bereits im PageController definiert und stellt grundlegende Methode für das Handling von DOM- Objekten bereit. Beispiel hierfür ist die __*getForm()*-Methode, die eine Referenz auf ein Formular zurückgibt.