Programmierhandbuch Imperia 8.6

Imperia AG
Leyboldstr. 10
D - 50354 Hürth
http://www.imperia.net

Programmierhandbuch
Imperia 8.6

Auflage: 01.12.2009 19:56

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1. Einführung

1.1 Konventionen in dieser Dokumentation

Die vorliegende Dokumentation benutzt folgende Konventionen:

- Dialoge, Abschnitte, Eingabefelder, Markierungsfelder, kurz alle Programmbestandteile werden **fett** gedruckt dargestellt.
- Auf besondere Zusammenhänge wird mit der fett gedruckten Überschrift **Achtung:** hingewiesen.
- Hinweise werden **kursiv** dargestellt.
- Platzhalter innerhalb von Code-Beispielen werden **kursiv** dargestellt.

1.2 Dokumenten-Modi

Imperia-Dokumente befinden sich abhängig vom aktuellen Workflow-Schritt bzw. von der aktuellen User-Aktion in unterschiedlichen Modi. Dies hängt davon ab, ob das Dokument zum Beispiel gerade bearbeitet wird oder in einer Vorschau betrachtet wird. Ein Dokument ist letztendlich der Container für alle Metadaten, die innerhalb eines Workflows gesammelt werden. Dies kann interaktiv durch den User über entsprechende Formulare geschehen (Metafiles, Templates, Flexmodule, Workflow-Plug-Ins) oder automatisiert ohne Interaktion direkt über Workflow-Plug-Ins bzw. "hidden"-Felder in Metafiles, Templates und Flexmodulen. Nachdem ein Dokument den Workflow durchlaufen hat, generiert das System über ein oder mehrere Templates Dateien, die im DOC-Root des Imperia-Webservers abgelegt werden. Dabei entscheiden die Templates, in welchem Format die Dateien abgelegt werden (textbasierte Formate wie HTM*, XML, etc.) und welche der im Workflow gesammelten Metadaten mit welchen Layout visualisiert werden. Beim Freischalten werden diese Dateien über das entsprechend ausgewählte Protokoll auf ein oder mehrere Zielsysteme kopiert. Optional können beim Freischalten über entsprechende Plug-Ins auch nur ausgewählte Metadaten des zugrundeliegenden Dokuments in SQL-Datenbanken exportiert werden.

Man unterscheidet folgende Dokumenten-Modi:

- META-Mode
- EDIT-Mode
- SAVE-Mode
- PREVIEW-Mode
- REPARSE-Mode

Imperia-Modi werden dazu verwendet, bestimmte Teile des Dokuments oder des entsprechenden Templates auszublenden oder bestimmte Ergänzungen und Aktionen nur in bestimmten Modi auszuführen.

META-Mode

Im META-Mode befindet sich ein Dokument genau dann, wenn es erzeugt wird. In diesem Modus wird das Dokument initialisiert und die Meta-Variablen `directory`, `template`, `filename` und `metafile` werden mit den entsprechenden Werten aus der Rubrik angelegt.

Wurde der Workflow-Schritt *Meta-Edit* verwendet und über den Button **Okay** abgeschlossen, kann sich ein Dokument zu keinem späteren Zeitpunkt wieder im META-Mode befinden, auch dann nicht, wenn es zurück an den Workflow-Schritt Meta-Edit verwiesen wird.

Hinweis:

Wurde der Workflow-Schritt Meta-Edit nicht abgeschlossen, sondern über den Button **Abbruch** *abgebrochen, dann befindet sich das Dokument beim nächsten Bearbeiten wieder im META-Mode.*

Der META-Mode wird häufig dazu verwendet, das Verzeichnis aus der Rubrik durch ein weiteres nummeriertes Verzeichnis zu erweitern, so dass jedes in einer Rubrik erzeugte Dokument ein eigenes Verzeichnis erhält. Ein solches Vorgehen bietet sich an, wenn viele Dokumente in einer Rubrik angelegt werden, sich aber nicht gegenseitig überschreiben dürfen, zum Beispiel bei aktuellen Nachrichten. Lesen Sie hierzu auch Abschnitt 2.3.3 **IF-Abfragen** auf Seite 7.

Eine Erweiterung im Meta-Edit findet jedoch nur dann statt, wenn dieser Workflow-Schritt über den Button **Okay** abgeschlossen wurde.

Hinweis:

Neben dem Meta-Edit Schritt gibt es noch weitere Möglichkeiten, den Dateipfad eines Dokumentes im Verlauf des Workflows zu ändern. Dies kann beispielsweise im Bearbeiten-Schritt oder auch durch Workflow-Plug-Ins geschehen, die die Metainformationen ändern können, wie den Meta-Setter.

EDIT-Mode

Im EDIT-Mode befindet sich ein Dokument dann, wenn das entsprechende Template angezeigt wird und der User Inhalt eingibt oder ändert. Ein Dokument kann sich beliebig oft im EDIT-Mode befinden.

Über die Prüfung, ob sich ein Dokument im EDIT-Mode befindet, können bestimmte Teile des Templates, meist beschreibende Informationen für den User, in anderen Dokumenten-Modi ausgeblendet werden. Lesen Sie hierzu Abschnitt 3.4.1 **Elemente ausblenden** auf Seite 53.

SAVE-Mode

Ein Dokument befindet sich im SAVE-Mode, wenn es fertig ist und den Workflow verlässt.

Eine praktische Anwendung des SAVE-Modus besteht darin, ein Dokument erst dann durch bestimmte Teile zu ergänzen, beispielsweise die Navigation, wenn es den Workflow verlässt. Dadurch würde vermieden, dass ein User während der Eingabe von Inhalt über die eventuell bereits vorhandene Navigation entweder andere Dokumente öffnet oder Fehlermeldungen erhält.

PREVIEW-Mode

In diesem Modus befindet sich ein Dokument immer dann, wenn es in einer Vorschau betrachtet wird. Dies kann über den Button **Vorschau** während des Bearbeitens vom Schreibtisch aus über die Spalten **Titel** und **URL** oder von der Freischalt-Liste aus, wenn das Dokument den Workflow bereits verlassen hat, geschehen.

Eine praktische Anwendung für den PREVIEW-Mode ist das Ergänzen des Dokuments zum Beispiel mit einem Navigations-Dummy, so dass der User, wenn die Navigation im Edit-Modus nicht vorhanden ist, zumindest einen Eindruck erhält, wie das fertige Dokument angezeigt wird.

REPARSE-Mode

Ein Dokument befindet sich im REPARSE-Mode, wenn es mit Hilfe des Template-Reparsers erneut geparst wurde.

Hinweis:

Im Gegensatz zur Abfrage des Modus mit "<!--mode--> EQ SAVE", der sich ausschließlich auf den oben beschriebenen SAVE-Mode bezieht, beinhaltet "<!--XX-savemode-->" auch die Modi PREVIEW und REPARSE. Der Grund hierfür ist, dass bei einer Abfrage von <!--XX-savemode--> lediglich geprüft wird, ob der gegenwärtige Modus nicht EDIT ist. Lesen Sie hierzu auch Abschnitt 3.4.5 **Modus abfragen und überprüfen** *auf Seite 55*

1.3 Variablen

In Imperia gibt es Meta-Variablen und als besondere Untergruppe davon die so genannten Imperia-Variablen. Meta-Variablen nehmen zum Beispiel den Inhalt auf, den ein User in ein Dokument eingibt. Auch Steuerungsvariablen, mit deren Hilfe der Weg eines Dokuments durch den Workflow bestimmt wird oder das Datum des automatischen Freischaltens, werden in Meta-Variablen gespeichert.

Imperia-Variablen, wie die Variable `__imperia_modified`, werden vom System erzeugt und mit Werten versehen. Man sollte davon absehen, diese Variablen zu verändern. Eine Liste aller relevanten Imperia-Variablen finden Sie in Abschnitt 9.2 **Imperia-Metafelder** auf Seite 170.

Ein Imperia-Dokument besteht aus Schlüssel-Wert-Paaren, wobei der Schlüssel der Name einer Variablen ist. Dies können sowohl Imperia- als auch Meta-Variablen sein. Imperia-Variablen enthalten Dokument-Eigenschaften, wie das Erzeugungs-Datum, das Datum der letzten Änderung usw. Meta-Variablen enthalten Werte, die durch User eingegeben oder über das Template bzw. die Meta-Datei gespeichert wurden.

Kapitel 2. Metadateien

Metadateien dienen dazu, Eigenschaften und Zusatzinformationen eines Dokumentes festzulegen und zu speichern, die nicht unmittelbar den Inhalt des Dokuments betreffen bzw. für die Darstellung relevant sind. Dazu gehören beispielsweise folgende Informationen:

- der Dateiname des Dokumentes
- das Verzeichnis, in dem das Dokument gespeichert werden soll
- das Template zum Eingeben von Inhalt
- Keywords für Suchmaschinen
- Freischalt- und Löschinformationen
- Steuerungsinformationen für den Workflow
- zusätzliche Informationen für die Veröffentlichung
- Steuerungsinformationen für den Workflow

Viele dieser Informationen erbt ein Dokument bereits über die Rubrik, in der es erzeugt wird, wie zum Beispiel das zu verwendende Template und das Verzeichnis, in dem die Dokumente letztendlich abgelegt werden.

Das System erzeugt während des Workflow-Schrittes **MetaEdit** aus einer Metadatei ein Eingabeformular ähnlich einem Imperia-Template. Der Benutzer erhält, wenn er diesen Schritt bearbeitet, das entsprechende Formular mit den in der Metadatei angelegten Eingabe- und Auswahl-Elementen.

Metadateien erzeugen Eingabe- und Auswahl-Elemente, die der Benutzer bearbeiten muss. Neben diesen sichtbaren Elementen können in Metadateien Zuweisungselemente enthalten sein, mit deren Hilfe Meta-Variablen ohne Benutzereingaben angelegt und gespeichert werden können. Zusätzlich stehen Kontrollstrukturen (z.B.: IF-Abfragen) zur Überprüfung von Bedingungen zur Verfügung.

Welche Meta-Datei für ein Dokument verwendet werden soll, wird in der Rubrik eingestellt. Bei der Bearbeitung des Workflow-Schrittes **MetaEdit** können in der Eingabemaske die bereits in der Rubrik definierten Meta-Variablen angezeigt und editiert werden. Alle in die Eingabemaske einer Metadatei eingegebenen Informationen werden im Dokument gespeichert.

Die Verwaltung der Metadateien geschieht über die Metadatei-Verwaltung, die im Kapitel **Metadateien** im Administrationshandbuch beschrieben wird.

2.1 Struktur einer Metadatei

Die Eingabemaske einer Metadatei bietet eine Reihe von Anzeige-, Bedienungs- und Zuweisungselementen. Mit ihnen lassen sich entweder bereits vorhandene Informationen anzeigen oder dem Benutzer Eingabemöglichkeiten für Informationen bereitstellen. Bei diesen Elementen handelt es sich um bekannte Formularelemente, wie zum Beispiel:

- Eingabefelder
- Drop-Down-Listen
- Checkboxen
- Mehrfachauswahllisten

Eine Beschreibung aller Elemente finden Sie in Abschnitt 2.3 **Syntax-Referenz** auf Seite 6.

Die Informationen, die der Benutzer in ein Bedienungselement eingibt, werden in einer für jedes Bedienungselement eindeutigen Variablen gespeichert. Der Name der Variablen wird durch den Namen des Bedienungselements bestimmt.

Neben den oben genannten Elementen können in Metadateien auch Kontrollstrukturen wie IF-Abfragen verwendet werden, um den Variablen in Abhängigkeit von einer Bedingung Werte zuweisen zu können.

Im folgenden ein Beispiel einer Metadatei:

```
❶TITLE "erste Metadatei"
❷AUTHOR "P. Kuhn"
❸HELPTEXT "Bitte tragen Sie unten einen Titel für das Dokument ein."

❹#IF ("<!--XX-METAMODE-->")
❺ HIDDEN "directory:<!--XX-directory-->/<!--count-->"
❻ PRINT "erzeugte Datei:<!--XX-directory-->/<!--copycount-->/index.html"
#ELSE
❼ PRINT "erzeugte Datei:<!--XX-directory-->/index.html"
#ENDIF

❽INPUT "35:title::Bitte den Titel eingeben:"
```

❶ Die erste Zeile einer Metadatei enthält ihren Titel, den Sie auch in der Metadatei-Verwaltung sehen (siehe Kapitel **Struktur / Metadateien** im Administrationshandbuch).

❷ In der zweiten Zeile steht der Name des Autors der Metadatei. Er dient jedoch lediglich der Information und ist für den weiteren Ablauf des Workflows nicht wichtig.

❸ Mit dieser Zeile beginnt der Code, der die Eingabemaske gestaltet. Zunächst ist das ein Hilfetext, der während der Eingabe von Informationen in die Maske zu sehen ist.

❹ Anschließend folgt eine IF-Abfrage, mit der in Abhängigkeit des Dokumenten-Modus die Variable `directory` gefüllt wird. Der erste Zweig der Anweisung wird nur ausgeführt, wenn das betreffende Dokument sich im META-Modus befindet (siehe Abschnitt 1.2 **Dokumenten-Modi** auf Seite 1).

❺ In dieser Zeile wird der Pfad zum Dokument erstellt. Dieser besteht aus der Angabe *<!--XX-directory-->*, dem Pfad unter dem alle Dokumente der betreffenden Rubrik liegen, und einem Verzeichnis, dessen Name aus einer Zahl besteht, die mit *<!--count-->* automatisch generiert wird.

❻ Der soeben erzeugte Dateipfad einschließlich des Dateinamens wird ausgegeben. Mit *<!--copycount-->* referenzieren Sie auf die zuletzt mit *<!--count-->* generierte Zahl. In diesem Fall ist das der in der Zeile darüber generierte Verzeichnisname.

❼ Anweisungen in diesem Zweig werden nur ausgeführt, wenn sich das betreffende Dokument nicht im META-Modus befindet. Hier erfolgt die Ausgabe des kompletten Pfades des Dokuments einschließlich des Dateinamens. Das im META-Modus automatisch erzeugte Verzeichnis, das den Dateipfad der Dokumente dieser Rubrik erweitert, ist bei allen folgenden Durchläufen des Meta-Edit-Schrittes über <!--XX-directory--> abrufbar. Es muss also nicht wie im META-Modus eigens referenziert werden.

❽ Zuletzt erhält der Benutzer die Möglichkeit, den Titel des Dokumentes in ein INPUT-Feld einzutragen.

2.2 Allgemeines zur Syntax

Jedes Auswahl- oder Eingabe-Element in einer Metadatei definiert eine neue Variable oder verändert eine bereits vorhandene Variable. Zu jedem Auswahl- oder Eingabe-Element gehören ein Schlüsselwort und eine Reihe von Parametern. Welche Parameter das im Einzelnen sind, entnehmen Sie bitte Abschnitt 2.3 **Syntax-Referenz** auf Seite 6.

Schlüsselwörter werden in der Regel groß geschrieben, die dazugehörigen Parameter werden in Anführungsstriche gesetzt und durch Doppelpunkte voneinander getrennt. Auch wenn ein Parameter nicht verwendet wird (zum Beispiel der Default-Wert), muss der dazugehörige Doppelpunkt angegeben werden.

Mit der folgenden Zeile erzeugen wir ein Eingabefeld. Die Benutzereingabe wird hier in der Variablen `keywords` gespeichert.

```
INPUT "60:keywords::Schlüsselwörter"
```

Das Schlüsselwort `INPUT` erzeugt an der gewünschten Stelle ein Eingabefeld. Nach dem Gleichheitszeichen wird die Breite des Eingabefeldes (60) definiert, gefolgt von einem Doppelpunkt.

Nun folgt der Name der Variablen, ebenfalls gefolgt von einem Doppelpunkt. In unserem Fall ist dies `keywords:`. In diesem Beispiel wird kein Default-Wert für die Variable gesetzt, so dass nach `keywords:` wieder ein Doppelpunkt folgt.

Als letztes wird der beschreibende Text angegeben, der in der Eingabemaske der Metadatei neben dem Eingabe-Element erscheinen soll.

Andere Eingabe-Elemente verhalten sich ähnlich. Lesen Sie hierzu Abschnitt 2.3 **Syntax-Referenz** auf Seite 6.

Neben diesem Verfahren können Variablen auch versteckt definiert werden, ohne dass der Benutzer eingreifen muss. Diese Methode kann beispielsweise verwendet werden, um das zu verwendende Template, das Zielverzeichnis oder den Dateinamen der Dokumente festzulegen. Die Syntax für diese Methode ist wie folgt:

```
HIDDEN "filename:index.html"
HIDDEN "directory:/docs/news/<!--count-->"
HIDDEN "template:myImperia"
```

Lesen Sie hierzu auch Abschnitt 2.3.2 **Versteckte Zuweisungen** auf Seite 7.

2.3 Syntax-Referenz

Imperia generiert aus der beschriebenen Syntax ein HTML-Formular mit den bekannten Eingabe- und Auswahl-Feldern.

2.3.1 Eingabefelder

Mit Hilfe von Eingabefeldern können Sie Variablen erzeugen und speichern. Die Werte einer Variablen stehen an jeder Stelle des Workflows zur Verfügung und die darin enthaltene Information kann abgerufen, geändert oder anderweitig weiterverarbeitet werden.

Die Syntax für ein Eingabefeld ist wie folgt:

`INPUT = "`*Anzahl*`:`*Variable*`:`*Wert*`:`*Beschreibung*`"`

Die Platzhalter im obigen Beispiel werden wie folgt ersetzt:

- *Anzahl* durch die gewünschte Darstellungs-Breite des Eingabefelds in der Eingabemaske.
- *Variable* durch den Namen der Variablen, die den vom Benutzer eingegebenen Wert enthalten soll.
- *Wert* durch den Text, der als Default-Wert der Variablen gespeichert werden soll. Diesen Wert hat die Variable dann, wenn der Benutzer keinen Wert eingibt. Wird kein Default-Wert vereinbart und der Benutzer gibt keinen Wert an, bleibt die Variable leer.

Hinweis:

Soll der Inhalt einer Variablen hier eingesetzt werden, so ist die Variable als META zu escapen, z.B. so:

```
INPUT = "60:title:<!--XX-META:title-->:Titel"
```

Anderenfalls kann es, wenn der Variablenwert Anführungszeichen enthält, zu Datenverlust kommen.

- *Beschreibung* durch den Text, der in der Eingabemaske als Beschreibung neben dem Eingabefeld erscheint.

Beispiel:

```
INPUT = "60:keywords:Bundesliga, Fußball:Schlüsselworte für Suchmaschinen"
```

Diese Zeile erzeugt ein Eingabefeld mit einer sichtbaren Breite von 60 Zeichen. Mit diesem Eingabefeld füllt der Benutzer die Variable `keywords`. Wenn er keine Eingaben macht, enthält es als Default-Wert `"Bundesliga, Fußball"`.

Auf den Inhalt dieser Variablen kann über die Syntax <!--XX-keywords--> beliebig oft an jeder Stelle im Workflow zugegriffen werden.

2.3.2 Versteckte Zuweisungen

Versteckte Zuweisungen verwendet man, um eine Variable zu füllen, ohne dass ein Benutzer eingreifen kann oder muss.

Das Schlüsselwort für eine versteckte Zuweisung ist `HIDDEN`. Es erwartet als Parameter den Namen einer Variablen und den Wert, den diese Variable zugewiesen bekommen soll:

```
HIDDEN "Variable:Wert"
```

Beispiel:

```
HIDDEN "filename:default.htm"
```

Mit dieser Zeile wird der Variablen `filename` der Wert `default.htm` zugewiesen.

2.3.3 IF-Abfragen

Mit Hilfe von IF-Abfragen können Bedingungen überprüft werden. Abhängig von der Bedingung können Variablen verändert werden.

Eine mögliche Anwendung ist die Bildung des Verzeichnisses in Abhängigkeit vom Dokumenten-Modus:

```
#IF ("<!--XX-METAMODE-->")
     HIDDEN "directory:<!--XX-directory-->/<!--count-->"
#ELSE
     HIDDEN "directory:<!--XX-directory-->"
#ENDIF
```

2.3.4 Drop-Down-Listen

Mit Hilfe einer Drop-Down-Liste kann der Benutzer eine von mehreren vorgegebenen Optionen aus einer Liste wählen. Jede Option weist der Variablen, die über die Drop-Down-Liste gefüllt wird, einen anderen Wert zu.

Die Syntax für eine Drop-Down-Liste ist wie folgt:

```
SELECTION "Meta-Variable:Beschreibung"
    OPTION "Wert 1:Text 1"
    OPTION "Wert 2:Text 2"
    OPTION "Wert 3:Text 3"
    OPTION "Wert 4:Text 4"
ENDSEL
```

Dieses Beispiel verwendet Platzhalter. Ersetzen Sie

- *Meta-Variable* durch den Namen der Variablen, die mit Hilfe der Drop-Down-Liste gefüllt werden soll.
- *Beschreibung* durch den Text, der in der Eingabemaske neben der Drop-Down-Liste erscheinen soll.
- *Wert 1* bis *Wert 4* durch den Wert, den die Variable bei Auswahl der entsprechenden Option annehmen soll.

- *Text 1* bis *Text 4* durch den Text, der für die entsprechende Option in der Drop-Down-Liste angezeigt werden soll.

Beispiel:

```
SELECTION "bgcolor:Hintergrundfarbe"
    OPTION "#FFFFFF:weiß"
    OPTION "#000000:schwarz"
    OPTION "#FF0000:rot"
ENDSEL
```

Mit dieser Drop-Down-Liste wird in der Variablen `bgcolor` der vom Benutzer gewählte Farbcode gespeichert. Man könnte diesen Farbcode beispielsweise wie folgt als Hintergrundfarbe des Dokumentes setzen:

```
<body bgcolor="<!--XX-bgcolor-->">
...
```

2.3.5 Mehrfachauswahl-Listen

Mit Hilfe einer Mehrfachauswahl-Liste kann der Benutzer aus einer Reihe von Optionen mehrere auswählen.

Die Syntax für eine Mehrfachauswahl-Liste ist wie folgt:

```
MULTIPLE_SELECTION "Zahl:Meta-Variable:Beschreibung"
    OPTION "Wert 1:Text 1"
    OPTION "Wert 2:Text 2"
    OPTION "Wert 3:Text 3"
    OPTION "Wert 4:Text 4"
ENDSEL
```

Dieses Beispiel verwendet Platzhalter. Ersetzen Sie

- *Zahl* durch die Anzahl der Zeilen der Mehrfachauswahl-Liste.
- *Meta-Variable* durch den Namen der Variablen, die mit Hilfe der Mehrfachauswahl-Liste gefüllt werden soll.
- *Beschreibung* durch den Text, der in der Eingabemaske neben der Mehrfachauswahl-Liste erscheinen soll.
- *Wert 1* bis *Wert 4* durch den Wert, den die Variable bei Auswahl der entsprechenden Option annehmen soll.
- *Text 1* bis *Text 4* durch den Text, der für die entsprechende Option in der Drop-Down-Liste angezeigt werden soll.

2.3.6 Checkboxen

Mit Hilfe von Checkboxen kann ein Benutzer Optionen aktivieren oder deaktivieren. So kann er zum Beispiel eine bestimmte Funktion an- oder abschalten.

Die Syntax für eine Checkbox ist wie folgt:

```
CHECKBOX "Name:Wert:linker Text:rechter Text:Parameter"
```

Das Syntaxbeispiel verwendet Platzhalter. Ersetzen Sie

- *Name* durch den Namen der Variablen, die durch die Checkbox mit einem wert gefüllt werden soll.
- *Wert* durch den Wert, der in der Variablen gespeichert werden soll.
- *linker Text* durch den Text, der in der Eingabemaske links neben der Checkbox erscheinen soll.
- *rechter Text* durch den Text, der in der Eingabemaske rechts neben der Checkbox erscheinen soll.

- *Parameter* durch einen beliebigen String, um die Checkbox initial auszuwählen.

Beispiel:

```
CHECKBOX "lang_de:1:Sprache:deutsch:selected"
CHECKBOX "lang_en:1:Sprache:english:"
CHECKBOX "lang_fr:1:Sprache:francaise:"
```

Dieser Code erzeugt drei Checkboxen, mit denen der Benutzer Sprachversionen auswählen kann. Die erste Checkbox ist standardmäßig selektiert (anstelle des Strings `selected` kann ein beliebiger String verwendet werden).

2.3.7 Radiobuttons

Mit Hilfe von Radiobuttons kann der Benutzer aus einer Reihe von Optionen eine auswählen. Mehrere Radiobuttons können über den Namen zu einer Gruppe zusammengefasst werden, in der ein zuvor ausgewählter Radiobutton durch das Auswählen eines anderen Radiobuttons deaktiviert wird.

Die Syntax für Radiobuttons ist wie folgt:

```
RADIO "Name:Wert:linker Text:rechter Text"
```

Das Syntaxbeispiel verwendet Platzhalter. Ersetzen Sie

- *Name* durch den Namen der Variablen, die durch den Radiobutton gefüllt werden soll.
- *Wert* durch den Wert, der in der Variablen gespeichert werden soll.
- *linker Text* durch den Text, der in der Eingabemaske links neben dem Radiobutton erscheinen soll.
- *rechter Text* durch den Text, der in der Eingabemaske rechts neben dem Radiobutton erscheinen soll.
- *Parameter* durch einen beliebigen String, um den Radiobutton initial auszuwählen.

Beispiel:

```
RADIO "radio:0:Wetter:regnerisch:"
RADIO "radio:1:Wetter:sonnig:selected"
RADIO "radio:2:Wetter:nebelig:"
```

Dieser Code erzeugt eine Gruppe von drei Radiobuttons, von denen der Mittlere initial selektiert ist (anstelle des Strings `selected` kann ein beliebiger String verwendet werden).

2.3.8 Template-Auswahl

Mit Hilfe der Template-Auswahl kann der Benutzer während des MetaEdit-Schrittes das Template für das Dokument auswählen. Die Auswahl des Benutzers wird in der Variablen `template` gespeichert.

Hinweis:

Eine Template-Auswahl ist in diesem Schritt nicht unbedingt notwendig, da bereits über die Rubrik ein Template festgelegt wurde.

Wenn Sie dem Benutzer in diesem Schritt keine Möglichkeit zur Auswahl eines Templates geben möchten, lassen Sie dieses Bedienelement einfach weg.

Wählt der Benutzer eine Rubrik aus, um ein neues Dokument zu erstellen, erhält in diesem Moment die Variable `template` bereits als Wert den Namen des in dieser Rubrik eingestellten Templates (siehe Kapitel **Metadatei und Default-Template bestimmen** im Administrationshandbuch).

Die Syntax für eine Template-Auswahl ist wie folgt:

```
TEMPLATESELECT "X,Y,Z"
```

Das Syntaxbeispiel verwendet Platzhalter. Ersetzen Sie *X*, *Y* und *Z* durch die Nummern oder Namen der Templates.

Beispiel:

```
TEMPLATESELECT "01,911,1002,myTemplate"
```

Diese Codezeile erzeugt in der Eingabemaske eine Drop-Down-Liste, in der die Beschreibungen der angegebenen Templates angezeigt werden. Der Benutzer kann dann einen Eintrag auswählen und das System verwendet das entsprechende Template für das Dokument.

Warnung:

Geben Sie mindestens ein Template an. Eine leere Template-Auswahl in der Metadatei verursacht einen Fehler, sobald das Dokument gespeichert wird.

2.3.9 Sprachauswahl

Wenn Sie mehrsprachige Inhalte publizieren wollen, bietet sich die in Imperia implementierte Multilanguage-Lösung an. Lesen Sie Abschnitt 9.4 **Mehrsprachige Dokumente mit Imperia-Multilanguage verwalten** auf Seite 173 für eine vollständige Anleitung zur Nutzung dieser Lösung. Die Auswahl der Sprachen, in denen ein Dokument verfügbar sein soll, nehmen Sie hierbei im Meta-Edit-Schritt vor. Notieren Sie die folgende Anweisung im Metafile, um eine Auswahl der verfügbaren Sprachen einzublenden:

```
MULTILANG_SELECTION
```

Daraufhin erscheint im Meta-Edit-Schritt für jede auswählbare Sprache eine Landesflagge mit einer Checkbox zum An- bzw. Abwählen der betreffenden Sprache. Imperia generiert dann automatisch für jede Sprache eine Copy-Seite (siehe Abschnitt 2.4.1 **copy** auf Seite 19). Die Anweisung *copy* brauchen Sie jedoch in diesem Fall nicht in der Metadatei zu notieren.

Die Inhalte für die einzelnen Sprachversionen können Sie dann im Edit-Schritt des Dokuments eingeben. Lesen Sie hierzu auch Abschnitt 3.4.13 **Mehrsprachige Inhalte eingeben** auf Seite 60. Welche Sprachversionen zur Auswahl stehen, legen Sie mit der Rubrik-Meta-Variablen `linguas` fest. Geben Sie die gewünschten Sprachen mit ihrem jeweiligen ISO 639 Sprach-Code an (**de**, **en**, **fr** etc.). Lesen Sie hierzu auch den Abschnitt "**Rubriken bearbeiten**" im Kapitel **Struktur** des Administrationshandbuchs.

2.3.10 Hilfetext

Hilfetext wird dazu verwendet, der Eingabemaske erklärenden Text für den Benutzer hinzuzufügen.

Die Syntax für Hilfetext ist wie folgt:

```
HELPTEXT "Dies ist der Hilfetext."
```

Dieser Code erzeugt einen einzeiligen Hilfetext. Ein Hilfetext wird nicht automatisch umbrochen. Längere Hilfetexte müssen auf mehrere solcher Zeilen umgebrochen werden.

2.3.11 Kommentare

Um bestimmte Abschnitte des Codes in der Metadatei zu kommentieren, verwenden Sie folgende Syntax:

```
// Dies ist ein Kommentar
```

Jede Kommentarzeile muss mit zwei Schrägstrichen beginnen.

2.3.12 Automatisches Löschen und Freischalten

Sie können Dokumente automatisch freischalten und vom Zielsystem löschen lassen. Die Zeitpunkte hierfür können Sie in einem Metafile mit speziellen Eingabeelementen definieren. Eingegebene Werte speichert Imperia in den Variablen `publish_date` und `expiry_date`.

Hinweis:

Die Meta-Variablen `publish_date` *und* `expiry_date` *können Sie zu jedem Zeitpunkt innerhalb eines Workflows setzen bzw. ändern, beispielsweise durch ein Metasetter-Plug-In. Freischalt- und Löschdatum müssen Sie also nicht im Metaedit-Schritt des Workflows bestimmen.*

Notieren Sie im Metafile Folgendes:

```
PUBLISH_DATE "Beschreibung"
```

für das Eingabeelement zur Festlegung des Freischaltdatums und

```
EXPIRY_DATE "Beschreibung"
```

für das Eingabeelement zur Festlegung des Löschdatums.

In der Eingabemaske sehen Sie dann jeweils ein Icon , mit dem Sie einen Kalender zum Einstellen des jeweiligen Datums aufrufen können. Dieser ist per Default mit einer Datumsangabe vorbelegt, die jeweils um ein Jahr bzw. vier Jahre versetzt in der Zukunft liegt.

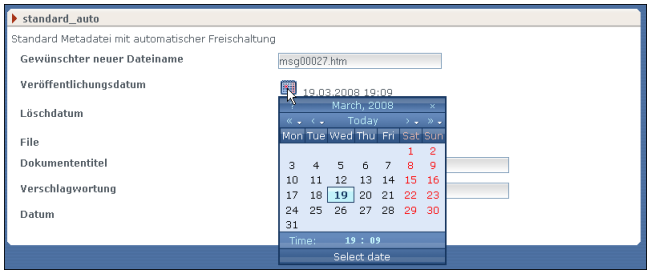

Abb. 2.1: Der Kalender zur Bestimmung des Freischaltdatums

Zeitversatz, bzw. Datumsangabe können Sie beliebig vorbelegen. Lesen Sie hierzu auch Abschnitt 2.3.12.1 **Voreinstellung für Datumsangaben festlegen** auf Seite 11. Das voreingestellte Datum erscheint neben dem Icon zum Aufruf des Kalenders. Geben Sie anstelle des Platzhalters *Beschreibung* einen beschreibenden Text ein. Dieser erscheint dann neben dem Icon zum Aufruf des Kalenders. Dazu ein Beispiel:

```
//Freischaltdatum mit Vorbelegung
    PUBLISH_DATE "Freischaltdatum:2007.01.11 23:05"
```

Obenstehendes Codebeispiel erzeugt im Metaedit-Schritt folgendes Bild:

Abb. 2.2: Festlegung des Freischaltdatums mit veränderter Vorbelegung

2.3.12.1 Voreinstellung für Datumsangaben festlegen

Den vorgegebenen Zeitversatz für automatische Freischalt- oder Löschdaten können Sie auch selbst beeinflussen. Hierzu stehen Ihnen folgende Möglichkeiten zur Verfügung:

Feststehende Datumsangabe

Eine feststehende Datumsangabe notieren Sie in der folgenden Form:

```
PUBLISH_DATE "Beschreibung:JJJJ.MM.TT HH:MM"

EXPIRY_DATE "Beschreibung:JJJJ.MM.TT HH:MM"
```

Im Meta-Edit-Schritt können Sie das Datum durch Auswahl eines anderen Datums über den Kalender ändern.

Zeitversetzte Datumsangabe

Eine flexiblere Variante als die Vorgabe eines feststehenden Zeitpunkts ist die Angabe des Versatzes der Datumsvorbelegung. Diesen definieren Sie mithilfe des Kommandos *Xstrftimeoff*:

```
PUBLISH_DATE "Beschreibung:<!--Xstrftimeoff:offset:Locale:%Y-%m-%d %H %M-->"

EXPIRY_DATE "Beschreibung:<!--Xstrftimeoff:offset:Locale:%Y-%m-%d %H %M-->"
```

Setzen Sie für `offset` den gewünschten Zeitversatz ein. Dieser besteht aus einem Operator, einer Zahl und einer optionalen Einheit. Als Operatoren kommen das Pluszeichen und das Minuszeichen in Frage. Die möglichen Einheiten sind in der folgenden Tabelle aufgeführt:

Einheit	Erklärung
`Y`	Jahre
`M`	Monate
`W`	Wochen
`D`	Tage
`h`	Stunden
`m`	Minuten

Tabelle 2.1. Einheiten für Zeit-Versatz

Wenn Sie keine Einheit angeben, wertet das System die Angabe automatisch als Sekunden aus. Anstelle des Platzhalters *Locale* geben Sie an, in welchem landesspezifischen Format das Datum ausgegeben werden soll, zum Beispiel *de_DE* für das deutsche Datumsformat oder *en_EN* für das englische.

Beispiel:

```
PUBLISH_DATE "Beschreibung:<!--Xstrftimeoff:+1h+30m:de_DE:%Y-%m-%d %H %M-->"
```

Das Dokument wird mit obenstehender Angabe eine Stunde und 30 Minuten nach der Erstellung veröffentlicht.

Es ist nicht vorgesehen, diese Zeitpunkte außerhalb des Workflows zu verändern.

Beim Reimport eines Dokuments aus dem Archiv ist eine Anpassung des Freischaltdatums nicht zwingend notwendig, denn das System schaltet Dokumente sofort frei, wenn deren Freischaltdatum in der Vergangenheit liegt.

23.12.2 Weitere Voreinstellungen

Auf Metaseiten können alle in Templates über Processing Instructions mögliche Voreinstellungen für den Kalender vorgenommen werden.

Beispiel:

```
PUBLISH_DATE = "label=Auto publish date:startDate=<!--Xstrftime:C:%Y-%m-%d %H:%M-->:startYear=
```

Eine Liste der möglichen Parameter finden Sie im Abschnitt 3.4.14.1 **Datumseingaben mit dem Kalender-Tool** auf Seite 60.

2.3.13 Datumsangaben für beliebige Metafelder

Ergänzend zur Angabe eines Freischalt- oder Löschdatums können Sie noch Datumsangaben in beliebigen Metafeldern abspeichern. Diese beliebigen Datumsangaben bestimmen Sie wie Freischalt- und Löschdaten mithilfe eines einblendbaren Kalenders. Im Metafile erzeugen Sie das entsprechende Eingabeelement mit folgender Anweisung:

```
DATE "Name:Beschreibung"
```

Mit dem Parameter *Name* geben Sie hierbei den Namen des Metafeldes an, in dem die Datumsangabe abgespeichert wird.

2.3.14 Ausgabe von Text

Informationen, die mit Hilfe von Imperia eigener Syntax aus dem System ausgelesen werden können, zum Beispiel Datum und Urzeit, oder sonstiger Text, der angezeigt werden soll, werden in der Eingabemaske wie folgt dargestellt:

```
PRINT "beschreibender Text:gewünschte Daten"
```

Text wird in der Eingabemaske zweispaltig ausgegeben, wobei die linke Spalte grundsätzlich fett gedruckt und die rechte Spalte grundsätzlich normal gedruckt dargestellt wird. Der Doppelpunkt trennt den Text, der fett gedruckt wird vom Text, der normal gedruckt wird. Daraus ergeben sich folgende Gestaltungsmöglichkeiten:

Doppelpunkt ganz links: `PRINT ": ..."`	Der gesamte Text wird in der rechten Spalte normal gedruckt dargestellt.
Doppelpunkt trennt den Text in zwei Teile: `PRINT "... : ...".`	Der Text links vom Doppelpunkt wird in der linken Spalte fett gedruckt, der Text rechts vom Doppelpunkt wird in der rechten Spalte normal gedruckt dargestellt.
Doppelpunkt ganz rechts oder kein Doppelpunkt: `PRINT "... :".`	Der gesamte Text wird in der linken Spalte fett gedruckt dargestellt.

Tabelle 2.2. Gestaltungsmöglichkeiten mit PRINT

Beispiel:

```
PRINT "Heute ist:<!--DATE:day-->"
```

Das Resultat in der Eingabemaske ist wie folgt:

Abb. 2.3: Resultat einer PRINT-Anweisung in einer Metadatei

2.3.15 Ausgabe von HTML-Code

Mit der Anweisung *HTML* können Sie in einem Metafile HTML-Tags ausgeben lassen.

```
HTML "<img src='/images/template_screenshots/news.png' />"
```

In Verbindung mit einem Templateselect (siehe Abschnitt 2.3.8 **Template-Auswahl** auf Seite 9) könnten Sie beispielsweise Vorschaubilder für die auswählbaren Templates in das Metafile integrieren.

2.3.16 Javascript in Metafiles

Es ist auch möglich, beim Abschluss des Meta-Edit Schritts Javascript-Funktionen auszuführen. Den Javascript-Code dafür können Sie mit der Anweisung *ONSUBMITSRC* im Metafile integrieren.

```
ONSUBMITSRC "checkFormValues();"
```

Hinweis:

Die Tags zur Kennzeichnung von Javascript-Bereichen oder zum Verweis auf externe Javascript-Dateien in einem HTML-Dokument müssen Sie mit der Anweisung HTML ins Metafile einbinden.

2.3.17 Benutzerdaten auslesen

Mit dieser Funktion ist es möglich, Daten des aktuellen Benutzers aus der Benutzer-Verwaltung auszulesen und in einer Variablen zu speichern.

Hinweis:

Man kann diese Benutzerinformation auch im Template auslesen und dort in einer Variablen speichern.

Die Syntax, um auf Benutzerdaten zugreifen zu können, lautet wie folgt:

```
<!--USER_CONF:Feld-->
```

Achtung:

Die alte Syntax mit Bindestrich (USER-CONF) wird nicht mehr unterstützt!

Folgende Felder können anstelle des Platzhalters *Feld* eingesetzt werden:

Name	Beschreibung
homepage	Startseite
country	Land
telnumber	Telefonnummer
language	Sprache
cellular	Mobiltelefon-Nummer
name	Name
fname	Vorname
city	Stadt
faxnumber	Faxnummer
comment	Kommentar
zip	PLZ
login	Anmeldekennung
email	E-Mail-Adresse
street	Straße

Tabelle 2.3. Felder der Benutzer-Verwaltung

Beispiel:

```
PRINT "Sie sind eingeloggt als:<!--USER_CONF:login-->"
```

2.3.18 Systemparameter auslesen

Diese Funktion erlaubt es, Systemparameter aus der Systemkonfiguration auszulesen und in einer Variablen zu speichern. Es können alle Parameter ausgelesen werden, die in der Datei `/site/config/system.conf` definiert sind.

Die Syntax, um Systemparameter auszulesen, ist wie folgt:

```
<!--SYSTEM_CONF:Parameter-->
```

Der Platzhalter *Parameter* muss mit dem Namen des entsprechenden Parameters aus der Datei `system.conf` ersetzt werden.

Beispiel:

```
PRINT "Document-Root:<!--SYSTEM_CONF:DOCUMENT-ROOT-->"
```

Eine Liste der Konfigurationsparameter finden Sie in Kapitel **Konfiguration von Imperia** im Administrationshandbuch.

2.3.19 Rubrikeneigenschaften auslesen

Es können verschiedene Eigenschaften der Rubrik, in der ein Dokument erzeugt wurde, über die Metadatei ausgelesen und in einer Variablen gespeichert werden.

Die Syntax ist wie folgt:

```
<!--SECTION:Schlüsselwort:Rubriklevel-->
```

Folgende Schlüsselwörter können verwendet werden:

NAME

Name der Rubrik.

DESCR

Beschreibung der Rubrik.

DIRECTORY

Das Verzeichnis, das in der Rubrik festgelegt wird.

TEMPLATE

Das Template, das in der Rubrik eingestellt wurde.

FILENAME

Der Dateiname der erzeugten Dokumente, der in der Rubrik festgelegt wurde.

Die Werte der Schlüsselwörter DIRECTORY, TEMPLATE und FILENAME können durch Variablen in der Metadatei übeschrieben werden. Diese Funktion greift trotzdem auf die entsprechenden Werte in der Rubrik zu. Verwenden Sie diese Schlüsselwörter nur dann, wenn Sie sicherstellen können, dass die Werte aus der Rubrik nicht an anderer Stelle geändert werden.

Soll nur der unterste Rubriklevel abgefragt werden, muss der Rubriklevel nicht angegeben werden:

```
<!--SECTION:NAME-->
```

Beispiel 1

Angenommen, es existiert eine Hauptrubrik `movies` mit einer Unterrubrik `action`, die wiederum eine Unterrubrik `martial arts` hat.

Um nun den Namen der Unterrubrik `action` zu erhalten, muss folgender Code eingegeben werden:

```
PRINT "Unterrubrik:<!--SECTION:NAME:2-->"
```

Beispiel 2

Sie verwalten zu jeder Rubrik den Namen des Autors in den zusätzlichen Metainformationen. Mit dem Ausdruck

```
PRINT "Rubrik-Autor:<!--SECTION:META_INFO_autor-->"
```

geben Sie den Inhalt des Meta-Feldes „autor“ der Rubrik im Metafile aus.

2.3.20 Dokumentenpfad auslesen

Es können Teile des Dokumentenpfades ausgelesen werden. Es wird der Name des gewünschten Verzeichnisses in einer Variablen gespeichert.

Die Syntax ist wie folgt:

```
<!--dirlevel:xxx-->
```

Der Platzhalter *xxx* muss durch den Wert der Position des gewünschten Verzeichnisses im Pfad ersetzt werden. Die Positionen beginnen mit dem obersten Verzeichnis, das durch den Wert `1` referenziert wird.

Beispiel

Angenommen ein Artikel liegt im Verzeichnis `/sport/Juni/16`. Die Code-Zeile

```
PRINT "Dieser Artikel ist vom <!--dirlevel:3-->. <!--dirlevel:2-->."
```

erzeugt in der Eingabemaske folgendes:

```
Dieser Artikel ist vom 16. Juni.
```

2.3.21 Datum und Zeit einfügen

Es gibt mehrere Möglichkeiten, ein Datum und eine Uhrzeit in einer bestimmten Form in einer Variablen zu speichern.

Die Syntax ist wie folgt:

```
<!--Xdate:Format-->
```

Der Platzhalter *Format* wird durch eins der folgenden Elemente ersetzt:

Format	Ergebnis
default	15.08.2005
full	15.08.2005
inverse	2005-08-15
afiles	2005-08-15 12:31
iso	20050815
normal	15.08.2005
american	08/15/2005
imperia	15.08.2005 12:31
finddate	2005.08.15 12:31

Tabelle 2.4. Datumsformate

Ein Datum kann in einem INPUT-Feld als Default-Wert vorgeschlagen werden:

```
<input name="IMPERIA:datum" type="text" size="8"
value="<!--Xdate:default-->">
```

Um das Datum lediglich auszugeben, wird folgende Syntax verwendet:

```
PRINT "Datum:<!--Xdate:default-->"
```

2.3.21.1 Auf Datumselemente zugreifen

Für den Zugriff auf einzelne Datumselemente wie Tag, Monat oder Jahr wird folgende Syntax verwendet:

<!--DATE:*Datumselement*-->

Der Platzhalter *Datumselement* wird durch eins der folgenden Elemente ersetzt:

Element	Ergebnis
day	gibt den Tag numerisch aus
month	gibt den Klartext des Monats aus
mon	gibt den Monat numerisch aus
year	gibt das Jahr aus

Tabelle 2.5. Datumselemente

Beispiel:

```
PRINT ":Wir schreiben das Jahr <!--DATE:year-->"
```

2.3.21.2 Uhrzeitformate

Um die Uhrzeit in einer Variablen zu speichern, verwende Sie folgende Syntax:

<!--Xtime:*Format*-->

Der Platzhalter *Format* wird durch eins der folgenden Elemente ersetzt:

Element	Ergebnis
default	12:31
normal	12:31:22
full	12:31:22
compact	12:31
hour	12
minute	31
second	22

Tabelle 2.6. Elemente der Uhrzeit

Beispiel:

```
HIDDEN "time: <!--Xtime:compact-->"
```

2.3.21.3 Datum und Uhrzeit lokalisiert

Formatierungen von Datum und Uhrzeit über die Systemfunktion `strftime` nehmen Sie mit der folgenden Syntax vor:

`<!--Xstrftime:`*Locale*`:`*Format*`-->`

Der Platzhalter *Locale* steht für die systemabhängige LOCALE-Variable. Lesen Sie hierzu die Dokumentation Ihres Systems.

Beispiel:

```
<!--Xstrftime:de_DE: Heute ist %A, der %d.%B %Y.-->
```

Diese Zeile wird zu:

```
Heute ist Dienstag, der 06. Januar 2004.
```

Hinweis:

HTML-Markup (kleiner als, größer als, Ampersand, doppelte und einfache Anführungszeichen), der in dem von `strftime` *gelieferten String enthalten ist, wird korrekt maskiert.*

23.21.4 Lokalisiertes Datum mit Offset

Um ein zukünftiges oder vergangenes Datum bzw. einen zukünftigen oder vergangenen Zeitpunkt zu erhalten, können Offsets angegeben werden. Die Syntax ist wie folgt:

`<!--Xstrftimeoff:`*Offset*`:`*Locale*`:`*Format*`-->`

Die Platzhalter *Locale* und *Format* sind wie zuvor zu ersetzen. Der Platzhalter *Offset* steht für eine beliebige Anzahl von Parametern der Form *Operator Wert Einheit*.

Folgende Operatoren stehen zur Verfügung:

Operator	Funktion
+	addiert den Wert zum / zur aktuellen Datum / Uhrzeit
-	subtrahiert den Wert vom aktuellen Datum / von der aktuellen Uhrzeit

Tabelle 2.7. Operatoren

Folgende Werte können verwendet werden:

Parameter	Bedeutung
`s`	Sekunden
`m`	Minuten (immer 60 Sekunden)
`h`	Stunden (immer 3600 Sekunden)
`D`	Tage (immer 86400 Sekunden)
`W`	Wochen (immer 604800 Sekunden, z.B. 7 Tage)
`M`	Monate (immer 2592000 Sekunden, z.B. 30 Tage)
`Y`	Jahre (immer 378432000 Sekunden, z.B. 365 Tage)

Tabelle 2.8. Datums- u. Zeit-Parameter

Schaltsekunden und Schalttage werden ignoriert, desgleichen die unterschiedlichen Anzahlen der Tage eines Monats im Jahr.

Beispiele:

Code	Effekt
`+1W-1D`	Heute plus eine Wochen minus ein Tag
`600`	Jetzt plus 600 Sekunden
`+1M`	Heute plus einen Monat

Tabelle 2.9. Beispiele für lokalisierte(s) Zeit/Datum

Der verantwortliche Algorithmus versucht, erwartete Ergebnisse zu liefern. Wird an einem 13. eines Monats ein Offset von drei Monaten eingestellt, ist das Ergebnis der 13. des entsprechenden Monats. Existiert das Zieldatum nicht (z.B. 31. Februar) wird der erste existierende Tag vor dem Zieldatum verwendet.

2.4 Funktionen und Konstanten in Metadateien

Im Folgenden finden Sie eine Liste aller Funktionen und Konstanten, die in Metadateien verwendet werden können. Grundsätzlich müssen alle Funktionen von Kommentarzeichen eingefasst werden. Beispiel:

```
<!--count-->
```

Die Anweisung aus dem folgenden Abschnitt bildet eine Ausnahme von dieser Regel.

2.4.1 copy

Eigentlich handelt es sich bei copy nicht um eine Funktion, sondern um ein spezielles Metafeld, das z.B. mit einer HIDDEN-Anweisung befüllt wird. Da Sie bei der Verwendung dieses Metafeldes die Erzeugung von Seitenkopien initialisieren, kann man aber auch von einer Funktion sprechen. Inhalt des copy-Feldes sind einige Parameter, mit denen Sie die Erzeugung der Kopie steuern. Für jedes Copy-Element erzeugt Imperia ein eigenes Dokument, bzw. eine eigene Datei. Dies geschieht in dem Moment, wenn das Hauptdokument den Workflow verlässt.

In den Copy-Seiten sind dabei alle Metainformationen des ursprünglichen Hauptdokuments verfügbar, das deswegen auch als **Masterdokument** bezeichnet wird. Welche Informationsbestandteile in welcher Kopie angezeigt werden, legen Sie in den jeweils verwendeten Templates fest. Während das Masterdokument den Workflow durchläuft, sammeln Sie die Informationen für alle Kopien, bzw. geben diese ein. Erst am Schluss, wenn das Dokument den Workflow verlässt, werden die Inhalte und Informationsbestandteile auf die einzelnen Kopien verteilt.

Befindet sich das Dokument in einem Bearbeiten-Schritt, steht Ihnen ein zusätzliches Auswahlfeld zur Verfügung. Hiermit bestimmen Sie, ob in der Vorschau das Masterdokument oder eine der Kopien erscheinen soll.

Hinweis:

Die Copy-Anweisung muss nicht zwangsläufig im Meta-Edit-Schritt erfolgen. Ebensogut können Sie das entsprechende Metafeld auch erst im Bearbeiten-Schritt des Workflows generieren und befüllen, beispielsweise mit versteckten Formularfeldern im Template. Beachten Sie aber, dass dort dann die Meta-Edit Funktionen count, bzw. copycount nicht zur Verfügung stehen.

In einer Copy-Anweisung legen Sie neben dem Dateinamen der Kopie eine Reihe weiterer Parameter fest. Diese notieren Sie als Schlüssel-Wert-Paar. Die Werte müssen dabei URL-encodiert sein. Ein Schlüssel kann immer nur einen Wert haben. Sind mehrere Werte notiert, wird nur der zuletzt notierte gespeichert.

Die Syntax für eine Copy-Anweisung lautet folgendermaßen:

```
HIDDEN "copy[Suffix]:Pfad/dateiname.end:[Parameter]"
```

Suffix

Optional lässt sich die Anweisung *copy* mit einem Suffix versehen. Dieser kann aus alphanumerischen Zeichen bestehen. Durch die Verwendung eines Suffix unterdrücken Sie die Anzeige eines Auswahlfeldes zur Bestimmung einer Kopie für die Vorschau im Edit-Modus eines Dokuments.

Pfad und Dateiname

Geben Sie an, unter welchem Pfad und Dateinamen Sie die Kopie abspeichern möchten. Achten Sie dabei auf die Eindeutigkeit des Dateinamens.

Tipp

Die Funktionen count (siehe Abschnitt 2.4.2 **count** *auf Seite 21) und copycount (siehe Abschnitt 2.4.5* **copycount:Zähldatei** *auf Seite 22) finden hierbei häufig Verwendung.*

TEMPLATE

Mit diesem Parameter legen Sie das Template für die Kopie fest. Wahlweise können Sie ein eigenes Template bestimmen oder das des Masterdorkuments verwenden.

Wichtig:

Beachten Sie, dass Imperia die Zuordnung der Informationen zu den Kopien über die Benennung der Metafelder vornimmt.

Per Default erscheint der Templatename als Beschriftung für die entsprechende Kopie im Auswahlfeld für die Vorschau im Bearbeiten-Schritt.

publish_date

Das Veröffentlichungsdatum einer Copy-Seite können Sie mit dem Parameter *publish_date* bestimmen. Imperia speichert den eingetragenen Wert in das gleichnamigen Metafeld des Dokuments. Die Datumsangabe muss in der Form **JJJJ-MM-TT+HH%3aMM** erfolgen. Das Leerzeichen zwischen Datums- und Zeitangabe sowie der Doppelpunkt zwischen Stunden und Minuten sind hierbei bereits URL-encodiert.

Hinweis:

An dieser Stelle notierte Werte können Sie in späteren Workflow-Schritten überschreiben.

expiry_date

Mit diesem Parameter bestimmen Sie das Löschdatum einer Copy-Seite. Das Datumsformat entspricht dem für das Veröffentlichungsdatum (siehe Abschnitt 2.4.1, „publish_date“). Imperia speichert den eingetragenen Wert im gleichnamigen Metafeld der Copy-Seite.

Hinweis:

An dieser Stelle notierte Werte können Sie in späteren Workflow-Schritten überschreiben.

__imperia_preview_select_name

Wenn Sie alle Kopien mit dem selben Template generieren, ermöglicht Ihnen dieser Parameter die Vergabe individueller Namen für die einzelnen Copy-Seiten im Auswahlfeld für die Vorschau. Ein Beispiel für Copy-Seiten mit identischem Template und unterschiedlichen Namen in der Vorschauauswahl:

```
HIDDEN "copy:<!--XX-directory-->/<!--copycount-->/de_<!--KK-filename-->:
                TEMPLATE=_multilang_de_en_fr:__imperia_preview_select_name=deutsch"
HIDDEN "copy:<!--XX-directory-->/<!--copycount-->/en_<!--KK-filename-->:
                TEMPLATE=_multilang_de_en_fr:__imperia_preview_select_name=english"
HIDDEN "copy:<!--XX-directory-->/<!--copycount-->/fr_<!--KK-filename-->:
                TEMPLATE=_multilang_de_en_fr:__imperia_preview_select_name=francais"
```

Hinweis:

Obenstehender Quellcode ist aus Darstellungsgründen umgebrochen.

Setzen Sie diesen Parameter auf *none*, um die betreffende Copy-Seite aus dem Vorschaufeld auszublenden. Die Anzeige des Masterdokuments im Auswahlfeld unterdrücken Sie folgendermaßen:

```
HIDDEN "__imperia_preview_select_name:none"
```

Beliebiges Metafeld

Neben den vorgenannten Parametern lassen sich in einer Copy-Anweisung beliebige Metafelder für eine Copy-Seite definieren. Diese können Sie dann beispielsweise als Freischalttrigger nutzen.

Wichtig:

Metainformationen, die Sie über eine Copy-Anweisung in ein Dokument schreiben, sind nur in der betreffenden Kopie abrufbar, wenn der Templateprozessor diese verarbeitet hat.

Bei einem Dokument mit Kopien sehen und bearbeiten Sie im Workflow immer das Masterdokument. Informationen, die für eine bestimmte Kopie spezifisch sind, können Sie dort nicht abrufen. Auch der Metaviewer zeigt Ihnen ausschließlich die Meta-Informationen des Masterdokuments.

Wenn Sie die spezifischen Informationen einer bestimmten Kopie überprüfen möchten, müssen Sie sich diese in der Vorschau anzeigen lassen.

Beispiel

Dazu ein Beispiel mit Parametern:

```
HIDDEN "copy_de:<!--XX-directory-->/<!--copycount-->/de_<!--XX-filename-->:
               TEMPLATE=_multilang_de_en_fr:publish_date=2007-01-22+12%3a22:
               expiry_date=2008-01-22+12%3a22:intranet=1"
```

Hinweis:

Der Text in diesem Beispiel ist aus Darstellungsgründen mehrfach umgebrochen.

In der Praxis kommen Copy-Seiten häufig bei der Erzeugung mehrsprachiger Dokumente zum Einsatz. Ein Besipiel finden Sie im Anhang unter Abschnitt 9.5 **Copy-Seiten, Beispiel mehrsprachige Dokumente** auf Seite 174.

2.4.2 count

Erzeugt eine fortlaufende Nummerierung, die mit jedem Aufruf der Variablen hochgezählt wird. Wird diese Funktion zur Bildung des Verzeichnisses verwendet, muss zusätzlich eine IF-Abfrage eingebaut werden, die den Modus des Dokumentes abfragt. Lesen Sie hierzu Abschnitt 2.5.1 **Zähler im Verzeichnisnamen** auf Seite 23.

Hinweis:

Aktualisiert der Benutzer das Browserfenster (F5 bzw. Strg+R), wird die Nummerierung erhöht. Es werden jedoch keine neuen Dokumente angelegt. Diese Variable ist bevorzugt im Meta-Mode einer Metadatei zu nutzen.

2.4.3 count*Zähler*

Erzeugt eine Nummerierung, die mit jedem inkrementellen Schritt um <Zahl> erhöht wird. Beispiel:

```
<!--count5-->
```

erhöht den Zähler um 5.

2.4.4 count:*Zähldatei*

Diese Funktion verwendet eine andere Zähldatei als die Standard-Zähldatei. Zähldateien liegen im Verzeichnis `/site/counter`. Die vorhandenen Dateien dürfen nicht verändert werden. Stattdessen muss eine neue Zähldatei mit beliebigem Namen ohne Endung angelegt werden.

Die Syntax zur Verwendung einer eigenen Zähldatei ist folgendermaßen:

```
<!--count:Dateiname-->
```

Der Aufbau einer Zähldatei ist wie folgt:

```
$start = '1'
$increment = '1'
$minvalue = '1'
$maxvalue = '2147483647'
$cycle = '1'
$currval = '41'
```

Schlüsselwort	Bedeutung
`$start`	Enthält den Startwert.
`$increment`	Enthält den Wert, um den inkrementiert wird.
`$minvalue`	Enthält den Wert, mit dem der Zähler nach einem kompletten Durchlauf wieder beginnt (wenn $cycle =1).
`$maxvalue`	Enthält den maximalen Wert, den der Zähler annehmen kann.
`$cycle`	Bestimmt, ob der Zählmechanismus wieder bei `$start` beginnt, wenn der Zähler `$maxvalue` erreicht hat. Kann die Werte `0` oder `1` annehmen.

Tabelle 2.10. Schlüsselwörter einer Zähldatei

2.4.5 copycount:Zähldatei

Diese Funktion liefert den gleichen Wert wie die Funktion `count`, jedoch ohne den Zähler zu inkrementieren. Sie kann zum Beispiel für mehrsprachige Dokumente verwendet werden. Wenn Sie diese Funktion zur Bildung des Verzeichnisses nutzten, müssen Sie zusätzlich eine IF-Abfrage einbauen, die den Modus des Dokumentes abfragt. Lesen Sie hierzu Abschnitt 2.5.1 **Zähler im Verzeichnisnamen** auf Seite 23

Die Syntax zur Verwendung der Kopie einer Zähldatei ist folgendermaßen:

```
<!--copycount:Dateiname-->
```

Beispiel:

```
#IF ("<!--XX-METAMODE-->")
    HIDDEN "directory:<!--XX-directory-->/<!--count-->"
#ELSE
    // No need to set the hidden field again if not first run.
#ENDIF

#IF ("<!--XX-METAMODE-->")
    PRINT "All languages:<!--XX-directory-->/<!--copycount-->/all.html"
    PRINT "English:<!--XX-directory-->/<!--copycount-->/index.html.en"
    PRINT "Deutsch:<!--XX-directory-->/<!--copycount-->/index.html.de"
    PRINT "Français:<!--XX-directory-->/<!--copycount-->/index.html.fr"
    PRINT "Italiano:<!--XX-directory-->/<!--copycount-->/index.html.it"
    PRINT "Nederlands:<!--XX-directory-->/<!--copycount-->/index.html.pt"
#ELSE
    PRINT "All languages:<!--XX-directory-->/all.html"
    PRINT "English:<!--XX-directory-->/index.html.en"
    PRINT "Deutsch:<!--XX-directory-->/index.html.de"
```

```
        PRINT "Français:<!--XX-directory-->/index.html.fr"
        PRINT "Italiano:<!--XX-directory-->/index.html.it"
        PRINT "Nederlands:<!--XX-directory-->/index.html.pt"
    #ENDIF
```

2.4.6 no_publish

Mit Hilfe dieser Funktion können Sie steuern, ob ein Dokument freigeschaltet werden soll oder nicht. Die Funktion füllt eine Variable mit dem gleichen Namen, die vom System geprüft wird. Ist die Variable leer oder hat den Wert `0`, wird das Dokument freigeschaltet. Jeder andere Wert der Variable führt dazu, dass das Dokument nicht freigeschaltet wird bzw. dass kein Dokument erzeugt und im DOC-Root des Webservers abgelegt wird.

Sie können damit beispielsweise steuern, ob bei einer Reihe von Copy-Dokumenten auch das Masterdokument veröffentlicht werden soll.

2.4.7 fullyear

Diese Konstante liefert das aktuelle Datum im Format YYYY.

2.4.8 year

Dies Konstante liefert das aktuelle Datum im Format YY.

2.4.9 fullmon

Diese Konstante liefert den Klartextnamen des aktuellen Monats (Januar, Februar, ...).

2.4.10 mon

Diese Konstante liefert den aktuellen Monat als Zahl im Format MM.

2.4.11 day

Diese Konstante liefert den aktuellen Wochentag als Zahl im Format DD.

2.4.12 fullday

Diese Konstante liefert den Klartextnamen des aktuellen Wochentags (Montag, Dienstag, ...).

2.4.13 hour

Diese Konstante liefert die aktuelle Stunde als Zahl im Format hh.

2.4.14 minute

Diese Konstante liefert die aktuelle Minute als Zahl im Format mm.

2.4.15 second

Diese Konstante liefert die aktuelle Sekunde als Zahl im Format ss.

2.5 Anwendungsbeispiele

In diesem Kapitel werden einige häufig verwendete Funktionen in Metadateien anhand von Beispielen erläutert.

2.5.1 Zähler im Verzeichnisnamen

Manchmal ist es erforderlich, dass jedes erzeugte Dokument in einem eigenen Verzeichnis abgelegt wird. Beispielsweise dann, wenn diese Dokumente aktuelle Nachrichten zum gleichen Bereich enthalten, man jedoch mit einem gerade neu freigeschalteten Dokument eine ältere Nachricht nicht überschreiben möchte.

Damit jedes Dokument in einem eigenen Verzeichnis landet, wird das bereits in der Rubrik vereinbarte Verzeichnis um ein weiteres Verzeichnis erweitert, dessen Name mit Hilfe eines Zählers gebildet wird.

Vor dem Hintergrund, dass ein Benutzer den MetaEdit-Schritt über den Workflow mehrfach ausführen kann, muss verhindert werden, dass in einem solchen Fall der Zähler zur Verzeichnisbildung weiter inkrementiert wird. Aus diesem Grund wird die Bildung des Verzeichnisses in einer IF-Abfrage gekapselt.

Diese prüft, ob sich das Dokument im METAMODE befindet. In diesem Modus befindet sich ein Dokument nur, wenn es direkt nach der Erzeugung den MetaEdit-Schritt durchläuft (siehe auch Abschnitt 1.2 **Dokumenten-Modi** auf Seite 1). Ein Dokument ist aber nicht im METAMODE, wenn es von einem Benutzer über den Workflow wieder an diesen Schritt zurückgeschickt wird.

Beispiel:

```
#IF ("<!--XX-METAMODE-->")
    HIDDEN "directory:<!--XX-directory-->/<!--count-->"
#ELSE
    HIDDEN "directory:<!--XX-directory-->"
#ENDIF
```

Hinweis:

Wird ein neue Metadatei über die Oberfläche erzeugt (Link **Neue Metadatei anlegen** *im Metadatei-Manager) enthält die neue Metadatei eine solche IF-Abfrage. Natürlich kann die Zählerfunktionalität wie auch die Node-ID eines Dokuments genutzt werden, um den Dateinamen eindeutig zu vergeben.*

2.5.2 Dateinamen im Meta-Edit-Schritt bestimmen

Wenn Sie Redakteuren ermöglichen wollen, selbst zu bestimmen, unter welchem Dateinamen ein Dokument abgespeichert wird, können Sie dies am einfachsten über ein Eingabefeld im Metafile lösen. Der Dateiname eines Dokuments ist in der Variablen *filename* gespeichert. Diese Variable ist über die Rubrikeinstellungen vorbelegt (siehe Abschnitt 5.1.1 **Rubriken-Informationen angeben** im Administrationshandbuch).

```
INPUT "30:filename::Gewünschter neuer Dateiname"
```

Beim Abspeichern der erfassten Meta-Informationen überschreibt der vom Redakteur eingegebene Name den vorbelegten Wert aus den Rubrikeinstellungen. Der neue Dateiname wird anschließend auf dem Schreibtisch angezeigt. Diese einfache Lösung hat allerdings einige Nachteile. Das Eingabefeld lässt sich nicht mit eigenem Text vorbelegen. Im Metaedit-Schritt erscheint dort grundsätzlich der Dateiname aus den Rubrikeinstellungen. Ferner sind im Metaedit-Schritt auf das Feld *filename* keine Zuweisungen über HIDDEN-Anweisungen möglich, wenn ein Redakteur den Inhalt dieses Felds gleichzeitig durch eine Eingabe verändert. Sie können also beispielsweise nicht gleichzeitig die Funktion *count* zum Anhängen eines Zählers nutzen, um eindeutige Dateinamen sicherzustellen.

```
//funktioniert nicht:

INPUT "30:filename:Bitte Dateinamen und -endung eintragen:Gewünschter neuer Dateiname"
HIDDEN "filename:<!--count--><!--KK-filename-->"
```

Weder die Vorbelegung des INPUT-Felds noch das Anhängen des Zählers mit der HIDDEN-Anweisung werden ausgewertet. Lediglich der vom Redakteur eingebene Text wird in der Variablen gespeichert. Mit einem kleinen Trick können Sie diese Funktionalität aber umsetzen:

```
INPUT "30:new_filename:Bitte Dateinamen ohne Endung eingeben:Neuer Dateiname"
HIDDEN "filename:<!--KK-new_filename--><!--count-->.htm"
```

Definieren Sie im Metafile ein zusätzliches Metafeld, dessen Inhalt der Redakteur durch ein vorbelegtes Eingabefeld ändern kann. Im obigen Codebeispiel ist dies das Metafeld *new_filename*. Anschließend weisen Sie den Inhalt dieses Metafeldes mit einer HIDDEN-Anweisung dem Feld *filename* zu. In der gleichen Anweisung generieren Sie mit der Funktion *count* einen eindeutigen Zähler und hängen die Dateiendung an.

Da die HIDDEN-Anweisung der einzige Zugriff auf filename ist, funktioniert diese Variante. Sobald der Redakteur den Meta-Edit-Schritt mit Speichern verlässt, ist der neue Dateiname des Dokumentes auf dem Schreibtisch abrufbar.

Tipp

Verwenden Sie als Vorbelegung für den neuen Dateinamen besser einen sinnvollen Default-Namen für den Fall, dass der Redakteur vergisst einen eigenen Namen einzugeben.

Hinweis:

Beachten Sie, dass Sie für den Zugriff auf das Metafeld new_filename eine KK-Variable (siehe Abschnitt 6.5 **KK-Variablen** *auf Seite 139) verwenden müssen, damit der soeben eingetragene Wert beim Abspeichern verfügbar ist. Ebenso ist zu beachten, dass die Vergabe von Dateinamen durch den Redakteur auch Gefahren hinsichtlich der Gültigkeit und Eindeutigkeit birgt (Sonderzeichen, Leerzeichen, etc.).*

Kapitel 3. Templates

Zu den Aufgaben eines Content-Management-Systems gehört es, Redakteuren die Eingabe von Inhalten für Websites zu ermöglichen, ohne dass diese über HTML- oder Programmierkenntnisse verfügen. Es muss dabei einerseits eine Eingabemaske für die Inhalte bereitstellen und andererseits die eingegebenen Inhalte in das bestehende Layout der Site einfügen. Das Instrument zur Erfüllung dieser beiden Aufgaben sind Templates. Die wörtliche Übersetzung des Begriffs, „Schablone“, beschreibt treffend, wozu ein Template dient. Es ist das Rohgerüst für ein Dokument. Es enthält feststehende und veränderliche Elemente. So können Sie gewährleisten, dass ein einheitliches Grundlayout gewahrt bleibt, während Inhalte variabel sind. Mit einem Template legen Sie also Folgendes fest:

- Größe, Position und Art von Grafiken
- Aussehen und Position des Textes
- Größe, Position und Aussehen der Navigation
- Gesamtgestaltung des Dokuments (Logo, Hintergrundbilder und -farben etc.)

Der Redakteur kann sich so auf die Eingabe des Texts und ggf. die Auswahl von Grafiken zur Illustration des Texts konzentrieren.

Technisch gesehen ist ein Template eine HTML-Datei mit Formularfeldern zur Texteingabe bzw. dem HTML-Gerüst für die fertige Seite. Dabei können die Erfassung der Inhalte und deren Ausgabe in einer Layoutvorlage wahlweise mit einem einzigen oder zwei getrennten Templates erfolgen. Bei der ersten Variante arbeiten die Redakteure mit dem Look-And-Feel einer WYSIWYG-Applikation. Anstelle der Eingabeelemente erscheint in der fertigen Seite der formatierte Inhalt. So haben die Benutzer schon beim Bearbeiten eine klare Vorstellung davon, wie das veröffentlichte Dokument aussehen wird. Wollen Sie hingegen einmal eingegebenen Inhalt in unterschiedlichen Layouts mehrfach präsentieren, bietet sich die zweite Variante an: ein layout-neutrales strukturiertes Eingabetemplate und spezifisch gestalteten Ausgabetemplates für die Darstellung der erfassten Inhalte.

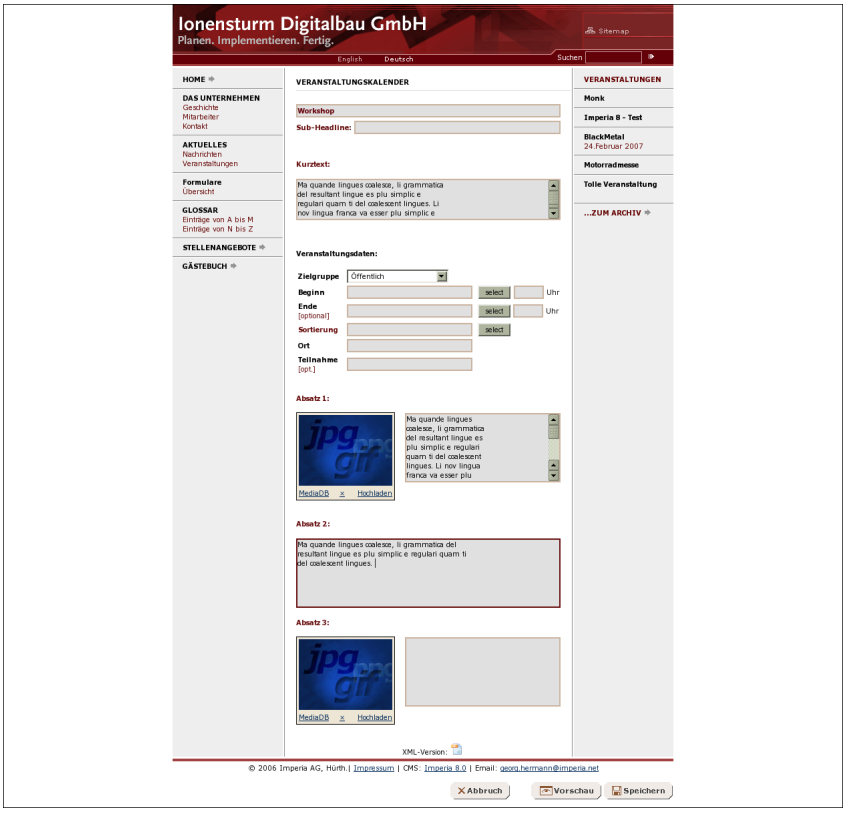

Abb. 3.1: Ein Template mit Eingabeelementen

Welches Template an welcher Stelle im Workflow zum Einsatz kommt, entscheiden Sie in Imperia in den Einstellungen der jeweiligen Rubrik oder auch im Workflow. Aus administrativer Sicht gibt es in der Templateverwaltung keinen Unterschied zwischen Ein- und Ausgabetemplates.

Welche Elemente einer Seite vorgegeben sind und welche Elemente der Redakteur hinzufügen oder bearbeiten kann, bestimmt der Entwickler des Templates. Außer den bereits erwähnten Eingabefeldern für Text können Sie dem Benutzer die Möglichkeit geben, Grafiken auszuwählen und in seinem Artikel an definierten Stellen zu platzieren. Optional können außerdem externe Applikationen darin integriert sein, z.B. externer Code wie PHP, JSP etc.

Mit Imperia können Sie darüberhinaus dem Benutzer, der Inhalte eingibt, mehr Entscheidungsfreiheit über die Darstellung der von ihm eingegebenen Inhalte gewähren. Sie können hierzu in Imperia-Templates variable Kombinationen von Strukturelementen bereitstellen. Auf diese Weise gewinnt ein Redakteur mehr Flexibilität. Dennoch braucht er keinerlei HTML- oder Programmierkenntnisse und dennoch ist durch die Templates ein einheitliches Layout der erstellten Dokumente sichergestellt.

Templates nehmen nicht nur den Inhalt für ein Dokument auf, sondern geben Ihnen auch die Möglichkeit zur Ausführung weiterer Funktionen:

- Variablen füllen
- Variablen auswerten
- Möglichkeiten zum Einfügen von Bildern und anderen Medien bereitstellen
- Perl-Code ausführen

Bei seiner Fertigstellung verlässt das Dokument den Workflow. Der Imperia Templateprozessor fügt in diesem Moment die im Verlaufe des Workflows gesammelten Inhalte in das Ausgabetemplate ein und erzeugt daraus eine Datei. Diese Datei können Sie anschließend auf das Zielsystem bzw. die Zielsysteme freischalten.

3.1 Konventionen für Dateinamen und Schlüsselwörter

Folgende Konventionen gelten in Imperia für Dateinamen und Schlüsselwörter:

Dateinamen

Der Dateiname eines Templates ist beliebig, darf aber keine Sonderzeichen außer dem Unterstrich und dem Bindestrich enthalten. Das aus älteren Imperia-Versionen bekannte Präfix `template` ist nicht mehr notwendig. Speichern Sie Ihre Templates mit der Dateiendung `.htms` ab.

Veränderliche Meta-Variablennamen

Imperia verändert die Meta-Variablennamen in Flexmodulen und Imperia-Blocks durch das Hinzufügen der entsprechenden Indizes. Diese werden durch Unterstriche voneinander getrennt an den Meta-Variablennamen angehängt. In einem Template dürfen keine Meta-Variablennamen der Form `name_1_2_3` verwendet werden, wenn außerdem Flexmodule oder Imperiablocks verwendet werden, die eine solche Meta-Variable (`name`) enthalten. Dies kann zu Überlappungen und Datenverlust führen.

Die folgenden Namen sind als Schlüsselwörter reserviert und dürfen nicht oder nur eingeschränkt als Meta-Variablennamen verwendet werden:

__imperia, _washed

Schlüsselwörter, die mit `__imperia` und `_washed` beginnen, übergeben intern Parameter zwischen den verschiedenen Layern von Imperia und dem Browser.

copy

Schlüsselwörter, die mit `copy` beginnen, sind für Copypages reserviert.

directory, filename, template, author

Diese Schlüsselwörter werden vom Publisher und der Workflow-Engine verwendet.

publish_date, expiry_date, no_publish, __publish_..., __expiry...

Diese Schlüsselwörter übergeben Parameter an den Publisher.

title

Dieser Schlüssel kann eingeschränkt verwendet werden. Es wird in Imperia unter anderem als Dokumenten-Titel auf dem Schreibtisch verwendet.

language, languages

Diese Schüssel werden vom Lokalisierungsmodul verwendet.

imperiaflex, imperiablocks

Diese Schlüsselwörter werden anonymen Meta-Variablen aus Flexmodulen und Imperia-Blocks vorangestellt.

3.2 Template-Grundlagen

Ein Template kann in verschiedenen Modi operieren:

- EDIT
- PREVIEW
- REPARSE
- SAVE

Die verschiedenen Modi lassen sich über Variablen abfragen. Das ermöglicht Ihnen, bestimmte Template-Elemente in verschiedenen Ansichten auszublenden bzw. anzeigen zu lassen. Ein gutes Beispiel hierfür ist das Ausblenden bestimmter Elemente (Hinweis-Texte etc.) im PREVIEW-Modus, damit die Vorschau dem fertigen Dokument entspricht.

Templates sind wie die Metadateien und Workflows an Rubriken gebunden. Daher empfiehlt sich die eingehende Beschäftigung mit diesen Themen, bevor Sie Templates erstellen. Näheres über die Konfiguration von Rubriken und Workflows erfahren Sie im Administrationshandbuch, das Thema Metadateien wird in Kapitel 2 **Metadateien** behandelt.

Alle in ein Template eingegebenen Daten können Sie über Variablen abfragen und an anderer Stelle wieder verwenden. Dazu müssen Sie alle Eingabeelemente eines Templates, wie zum Beispiel Eingabezeilen und Eingabefelder, eindeutig benennen.

Welches Template als Vorlage für ein Dokument dient, bestimmt in Imperia die Meta-Variable `template`. Bereits beim Erstellen des Dokuments bekommt diese Variable ihren Wert aus den Rubrikeneinstellungen. Im Verlauf des an die Rubrik geknüpften Workflows kann sich diese Meta-Variable durch Workflow-Plug-Ins ändern. Lesen Sie hierzu auch beispielsweise Abschnitt 2.3.8 **Template-Auswahl** auf Seite 9.

3.2.1 Erlaubte Zeichen in Variablenamen

Variablennamen dürfen in Imperia folgende Zeichen enthalten:

- Zahlen
- Buchstaben
- Unterstriche

Achtung:

Doppelpunkte können Sie in Variablennamen nicht verwenden, da dies zu unerwartetem Verhalten führt. Verwenden Sie bei der Programmierung eines Templates einen Doppelpunkt in einem Variablennamen, z.B.: `name="IMPERIA:txt:myname"`*, lautet der Name der Variablen* `txt:myname` *statt* `myname`*. Die Vorschau des Dokuments erscheint korrekt, jedoch sind weder über* `<!--XX-myname-->` *noch über* `<!--XX-txt:myname-->` *Zugriffe auf den Inhalt der Meta-Variablen möglich.*

3.2.2 Skripte, Formulare, Frames und Layer/ILayer in Templates

Standardmäßig werden im Template enthaltene Skripte, Frames und Formulare im EDIT-Modus unterdrückt. Dieses Verhalten kann jedoch systemweit eingestellt werden. Lesen Sie hierzu das Kapitel **Templateprozessor** im Administrationshandbuch und Abschnitt 3.8 **Der Templateprozessor** auf Seite 87.

3.3 Syntax-Referenz

Im Folgenden finden Sie Erläuterungen zur Syntax von Templates. Imperia erwartet relevanten Code zwischen dem `formstart` -Kommentar und dem `formend`-Kommentar (siehe auch Abschnitt 3.3.1 **Rumpf eines Templates** auf Seite 29).

In einem Imperia-Template vorhandene Content-Eingabemöglichkeiten befinden sich in einem HTML-Formular. Die Benutzer geben den Inhalt über Formular-Elemente ein. Imperia speichert anschließend die eingegebenen Inhalte in Variablen. Der Name einer Variablen resultiert dabei aus dem Namen des entsprechenden Formular-Elements.

Solche Variablen können Sie mit Hilfe einer bestimmten Syntax an jeder beliebigen Stelle abrufen und in das Dokument eingefügen (siehe Abschnitt 3.4.3 **Meta-Variablen referenzieren** auf Seite 54).

Neben den aus normalen HTML-Formularen bekannten Elementen, wie zum Beispiel `<input>` oder `<textarea>`, stellt Imperia über eine Reihe von eigenen Tags besondere und komplexe Funktionen zur Verfügung. Hierzu gehören beispielsweise Flexmodule, ImperiaBlocks, der Aufruf der Mediendatenbank, etc.

Damit Imperia ein HTML-Formularelement als relevant einstuft, müssen Sie das `name`-Attribut dieses Elements als erstes notieren und wie folgt ergänzen:

```
<input name="IMPERIA:Variablenname" type="text" size="35" />
```

Funktionen und Konstanten in Templates werden in Abschnitt 3.4 **Funktionen und Konstanten in Templates** auf Seite 53 beschrieben.

Die Funktionen zur Verwaltung von Templates in Imperia werden im Kapitel **Templates** des Administrationshandbuchs beschrieben.

3.3.1 Rumpf eines Templates

Templates können Sie mit einem beliebigen Editor erstellen. Welche Elemente Sie einfügen müssen, um aus einer HTML-Datei ein Imperia-Template zu machen, zeigt das folgende Beispiel:

```
<html>
  <head>
    <title>Leeres Template</title>
  </head>

  <body>
    <!--formstart-->❶
    <!--template-description:Test-Template-->❷

    <!--controls-->❸
    <!--formend-->❹
  </body>
</html>
```

❶ Imperia ersetzt die Zeile `<!--formstart-->` im Bearbeitungsmodus durch ein Formular-Tag mit den zugehörigen Attributen. Dieses öffnet das Formular, das bei der Erstellung des Dokumentes zur Eingabe von Content in das Template dient. Alle Eingabeelemente des Templates müssen innerhalb dieses Formularbereiches liegen. Wenn Sie diesen Kommentar nicht selbst in Ihrem Template notieren, ergänzt der Templateprozessor das einleitende Formular-Tag automatisch direkt hinter dem einleitenden Body-Tag des HTML-Dokuments.

❷ In der Zeile `<!--template-description:Test-Template-->` bestimmen Sie die Bezeichnung des Templates, die in den Imperia-Dialogen, beispielsweise bei den Rubrikeneinstellungen erscheint.

❸ Das Element *<!--controls-->* wird im Bearbeitungsmodus durch die Steuerelemente *Abbrechen*, *Vorschau* und *Speichern* ersetzt. Wenn Sie diese drei Buttons also am Kopf, am Fuß oder in der Mitte des Templates positionieren wollen, setzen Sie dieses Element an entsprechender Stelle im Quellcode ein. Es ist auch möglich, die Steuerelemente mehrfach innerhalb eines Templates anzeigen zu lassen.

❹ Mit der Zeile *<!--formend-->* erzeugen Sie im Bearbeitungsmodus das schließende Form-Tag. Wenn Sie diesen Kommentar nicht selbst in Ihrem Template notieren, ergänzt der Templateprozessor das einleitende Formular-Tag automatisch direkt vor dem schließenden Body-Tag des HTML-Dokuments.

Hinweis:

Die Template-Rumpf-Elemente sind im fertigen Dokument nicht mehr zu sehen. Der Templateprozessor entfernt die Kommentare, wenn das Dokument den Workflow verlässt.

Speichern Sie die Template-Dateien im Verzeichnis `/site/templates` unter der Endung `.htms`. Beachten Sie, dass der Name der Datei mit dem Wort `template` beginnen muss, zum Beispiel `templateNewsaktuell.htms`. Imperia ergänzt dieses Präfix beim Upload und Anlegen von Templates über das Template-Management automatisch.

3.3.2 HTML-Escaping-Modi

Mit Hilfe eines Escaping-Modus kann der Template-Programmierer bestimmen, auf welche Weise sich HTML-Code, den ein Benutzer in ein Eingabe-Element eingibt, im fertigen Dokument auswirkt. Grundsätzlich wird eingegebener HTML-Code immer unverändert in der entsprechenden Meta-Variablen gespeichert. Der Escaping-Modus des entsprechednen Eingabe-Elements bestimmt, ob dieser Code als funktionierendes HTML oder funktionsloser Text erscheint.

Die Eingabe-Elemente, für die Modi vereinbart werden können, sind:

- Input-Feld
- Textarea

In einem fertigen Dokument werden die Eingabe-Elemente des Templates durch den Inhalt der entsprechenden Meta-Variablen ersetzt, so dass ein Benutzer bei entsprechendem (oder falschem) Escaping-Modus des Eingabe-Elements auch komplexe Tabellen in eine Textarea eingeben und somit das im Template vorgesehene Layout massiv verändern könnte.

Auch bei der Referenzierung des Inhalts einer Meta-Variablen (XX-Syntax) kann ein Escaping-Modus angegeben werden, wodurch der Inhalt dieser Meta-Variablen entsprechend im fertigen Dokument maskiert wird. Die Escaping-Modi, die Ihnen für Referenzierungen zur Verfügung stehen, finden Sie in Abschnitt 6.21 **Escaping Modes** auf Seite 149.

Achtung:

Wird ein Eingabe-Element mit einem ungültigen Modus vereinbart (z.B.: `name="IMPERIA:TEXT:text1"`*), ist der Name der daraus resultierenden Variablen nicht* `text1`*, sondern* `TEXT:text1`*! Der Zugriff auf eine solche Variable ist jedoch weder mit* `<!--XX-text1-->` *noch mit* `<!--XX-TEXT:text1-->` *möglich.*

Der Escaping-Modus wird als Parameter an das `name`-Attribut des entsprechenden Tags angehängt. Das folgende Beispiel demonstriert dies anhand eines einfachen Eingabefeldes:

```
<input name="IMPERIA:HTML:Name" type="text" size="30" />
```

Im Beispiel wird eventuell eingegebener HTML-Code im fertigen Dokument berücksichtigt und verändert das Layout. Dies kann gewünscht sein (fette oder kursive Text-Auszeichnung), birgt aber auch die Gefahr, dass Benutzer komplexere Layout-Änderungen vornehmen könnten.

Folgende Escaping-Modi stehen zur Verfügung:

kein Modus

Dies ist der Standardmodus, der für alle relevanten Eingabe-Elemente verwendet werden kann. In diesem Modus werden eingegebene Zeilenschaltungen als Newlines (\n) in der Meta-Variablen gespeichert. Im fertigen Dokument werden bei der Darstellung des Inhalts diese Newlines automatisch in
-Tags umgewandelt und erscheinen entsprechend im fertigen Dokument.

Eingegebener HTML-Code wird maskiert. Beispielsweise wird aus `<b>` im fertigen Dokument `&lt;b&gt;`.

HTML-Modus

Dieser Modus kann in allen relevanten Eingabe-Elementen verwendet werden. In diesem Modus wird eingegebener HTML-Code nicht maskiert und beeinflusst das Layout des fertigen Dokuments. Vom Benutzer eingegebene Zeilenschaltungen in einer Textarea werden **nicht** in der Meta-Variablen gespeichert und erscheinen **nicht** im fertigen Dokument.

HTMLBR-Modus

Dieser Modus kann nur in Verbindung mit Textareas verwendet werden. Er verhält sich in Bezug auf eingegebenen HTML-Code analog zum HTML-Modus, jedoch werden zusätzlich eingegebene Zeilenschaltungen berücksichtigt und im fertigen Dokument bei der Darstellung des Meta-Variablen-Inhalts automatisch in `<br />`-Tags umgewandelt.

TEXT-Modus

In diesem Modus werden alle Sonderzeichen durch SGML-Entities ersetzt.

TEXTBR-Modus

Die Ersetzung erfolgt analog zum TEXT-Modus, wobei zuerst alle Zeilenumbrüche durch <br /> ersetzt werden.

URI-Modus

Dieser Modus konvertiert die Sonderzeichen durch die für den MIME-Typ „application/x-www-form-urlencoded“ notwendige Schreibweise mit Prozent-Zeichen.

3.3.3 Input-Feld

Input-Felder werden verwendet, um einzeilige Texte einzugeben, zum Beispiel Artikel-Überschriften oder Bildunterschriften.

Die Syntax eines Input-Felds kann durch die Angabe eines Escape-Modus' ergänzt werden, um eingegebenen HTML-Code auf verschiedene Weisen behandeln zu können. Lesen Sie hierzu Abschnitt 3.3.2 **HTML-Escaping-Modi** auf Seite 30

Die Syntax für ein Input-Feld ohne Escaping-Modus ist wie folgt:

```
<input name="IMPERIA:Name" type="text" size="30" />
```

Die Syntax für ein Input-Feld mit Escaping-Modus ist wie folgt:

```
<input name="IMPERIA:Modus:Name" type="text" size="30" />
```

3.3.4 Textarea

Textareas werden verwendet, um mehrzeilige Texte aufzunehmen, so dass der Benutzer lange Fließtexte eingeben und speichern kann.

Die Syntax einer Textarea kann durch die Angabe eines Escaping-Modus' ergänzt werden, um eingegebenen HTML-Code auf verschiedene Weisen behandeln zu können. Lesen Sie hierzu Abschnitt 3.3.2 **HTML-Escaping-Modi** auf Seite 30.

Die Syntax für ein Textarea-Feld ohne Escaping-Modus ist wie folgt:

```
<textarea name="IMPERIA:Name" rows="5" cols="40"></textarea>
```

Die Syntax für ein Textarea-Feld mit Escaping-Modus ist wie folgt:

```
<textarea name="IMPERIA:Modus:Name" rows="5" cols="40"></textarea>
```

3.3.5 Hyperlinks

Die Syntax für einen Hyperlink ist wie folgt:

```
<a href="IMPERIA:Name">Linktext</a>
```

Im `EDIT`-Modus wird ein Eingabefeld angezeigt, in das der URI des Hyperlink-Ziels eingegeben wird. Um auch den Linktext eingeben zu können, kann folgende Syntax verwendet werden:

```
<a href="IMPERIA:link">
  #IF <!--XX-editmode-->
    Linktext:
  #ENDIF
  <input name="IMPERIA:linktext">
  </a>
```

3.3.6 Combo-Box

Combo-Boxen geben dem Benutzer die Möglichkeit, aus mehreren Optionen die gewünschte auszuwählen. Die Syntax für eine Combo-Box ist wie folgt:

```
<select name="IMPERIA:combo">
   <option value="option1">erste Option</option>
   <option value="option2">zweite Option</option>
</select>
```

Tipp

Im Gegensatz zu früheren Imperia-Versionen muss der Code nicht mehr in eine Zeile geschrieben werden.

Im Template wird eine Combo-Box angezeigt, in der genau eine Option ausgewählt werden kann. Dieses Beispiel speichert den Wert `option1` in der Meta-Variablen `combo` des Dokuments, im fertigen Dokument wird die ausgewählte Option angezeigt. Dadurch ist es möglich, den angezeigten Text im Nachhinein zu modifizieren, ohne die Meta-Variablen ändern zu müssen.

Soll eine der Optionen als Default-Wert voreingestellt werden, muss die Syntax dieser Option ergänzt werden:

```
  <option value="option1" selected="selected">erste Option</option>
```

3.3.7 MultiSelect-Elemente

MultiSelect-Felder geben dem Benutzer die Möglichkeit, mehrere Optionen auszuwählen. Die Syntax ist wie folgt:

```
<select name="IMPERIA:Name" size="4" multiple="multiple">
  <option value="Wert">Label</option>
</select>
```

Die in einem MultiSelect-Element ausgewählten Werte werden in einem Array gespeichert. Die Nummerierung der Array-Elemente beginnt mit `0`. Jedes Array-Element kann mit Hilfe einer Klammersyntax referenziert werden. Hierbei enthält die Klammer die Position der gewünschten Array-Instanz.

Beispiel:

```
<select name="IMPERIA:multiselect" size="4" multiple="multiple">
  <option value="1">1. Option</option>
  <option value="2">2. Option</option>
  <option value="3">3. Option</option>
  <option value="4">4. Option</option>
  <option value="5">5. Option</option>
  <option value="6">6. Option</option>
  <option value="7">7. Option</option>
</select>
```

Um zum Beispiel den Wert der zweiten ausgewählten Option zu erhalten, wird folgende Syntax verwendet:

```
<!--XX-multiselect[1]-->
```

Siehe dazu auch Abschnitt 3.3.12 **Arrayblocks** auf Seite 36.

3.3.8 Checkbox

Die Syntax einer Checkbox ist wie folgt:

```
<input name="IMPERIA:Name" type="checkbox" value="Wert">
```

Es ist möglich, mehrere Checkboxen zu einer Gruppe zusammenzufassen. Dies geschieht über das `name`-Attribut (siehe Beispiel).

Beispiel:

```
<input name="IMPERIA:check3" type="checkbox" value="one" />
<input name="IMPERIA:check3" type="checkbox" value="two" />
<input name="IMPERIA:check3" type="checkbox" value="three" />
<input name="IMPERIA:check3" type="checkbox" value="four" />
```

Werden mehrere Checkboxen aus einer Gruppe ausgewählt, speichert Imperia die entsprechenden Werte in einem Array, von wo aus sie mit Hilfe der Klammersyntax referenziert werden können. Um zum Beispiel den dritten ausgewählten Wert aus dem Beispiel zu referenzieren, wird folgende Syntax verwendet:

```
  <!--XX-check3[2]-->
```

Siehe dazu auch Abschnitt 3.3.12 **Arrayblocks** auf Seite 36.

3.3.9 Radio-Buttons

Die Syntax für einen Radio-Button ist wie folgt:

```
<input name="IMPERIA:Name" type="radio" value="Wert" checked="checked" />
```

Das Attribut `checked="checked"` bestimmt, ob eine Option standardmäßig ausgewählt ist.

Beispiel:

```
<input name="IMPERIA:radio1" type="radio" value="one" />
<input name="IMPERIA:radio1" type="radio" value="two" checked="checked" />
<input name="IMPERIA:radio1" type="radio" value="three" />
```

3.3.10 Datei-Upload (Media-Asset-Management)

Um ein Template für den Upload von Assets in das Media-Asset-Management zu erstellen, benötigen Sie ein spezielles Eingabefeld, mit dem Sie die gewünschte Datei zum Upload auswählen können. Dieses Feld erzeugen Sie im Template mit dem folgenden Kommentar:

```
<!--upload_start-->
<!--upload_end-->
```

Hinweis:

Gegenwärtig ignoriert der Templateprozessor Text, den Sie zwischen Start- und Endkommentar notieren. HTML-Tags zwischen den beiden Kommentaren verhindern allerdings die Anzeige des Upload-Feldes.

Das Workflow-Plug-In **Upload** interpretiert diesen Kommentar und generiert daraufhin folgendes Eingabeelement:

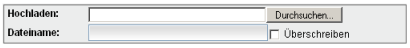

Abb. 3.2: Eingabeelement zum Asset-Upload

Imperia speichert die ausgewählte Datei sowie die Informationen, die Sie über etwaige andere Eingabefelder des Templates erfassen, in dem Asset-Dokument.

Hinweis:

Wenn das MAM-Template für den Multiupload verwendet wird, kommt es bei Verwendung von Slots, Flexmodulen und Imperiablocks zu Fehlermeldungen. Bitte sehen Sie von deren Verwendung in Asset-Templates ab.

3.3.11 Flash-Mehrfachupload (Media-Asset-Management)

Die Bedienelemente für den Flash-Upload binden Sie im Template durch Processing-Instructions ein. Die Parameter der Processing-Instructions konfigurieren den Flash-Upload. Ein `<!--upload_start--><!--upload_end-->` ist nicht notwendig. Die Erfassung von Metainformationen zu den einzelnen Assets über das Template ist leider nicht möglich.

Weitere Hinweise zur Verwendung des Flash-Multiuploads finden Sie im Administrationshandbuch im Abschnitt **Mehrfachupload mit Flash** des Kapitels **Media-Asset-Management (MAM)**.

<?imperia swf_upload?>

Diese PI ist bindend, um den Flash-Upload nutzen zu können. Folgende Parameter sind verfügbar:

- `multiple`:

  Boolescher Wert (Default: 1). Setzen Sie diesen Parameter auf 1, um das Auswählen mehrerer Dateien im Dateiauswahlfenster zu aktivieren.

- `autoUpload`:

  Boolescher Wert (Defaults: 0). Setzen Sie diesen Parameter auf 1, damit der Upload-Vorgang automatisch startet, sobald der Benutzer das Dateiauswahlfenster schließt.

- `fileTypes`:

  Zeichenkette (Default: ''). Mit diesem Parameter schränken Sie die Auswahl hochladbarer Dateien für den Redakteur auf einen bestimmten Medientyp ein. Die Syntax ist wie folgt:

  ```
  fileTypes: Endung01;Endung02;EndungNN,Filtername
  ```

  Beispiel:

  ```
  fileTypes: *.png;*.jpeg;*.gif;*.jpg,Bilder
  ```

Dieser Filter ist dann beim Dateiauswahlfenster in dem Feld "Dateityp" eingestellt, so dass nur noch Dateien dieses Typs auswählbar sind.

- `fileSizeLimit`:

  Zeichenkette (Default: '1024') Geben Sie ein Dateigrößenlimit für den Upload an. Die Angabe erfolgt automatisch in Kilobyte, sofern nicht anders angegeben. Setzen Sie den Parameter auf 0, um unbegrenzt große Dateien hochladen zu können.

  Beispiele:

  123412, *4621B*, *7962kb*, *12MB*, *2GB*

- `fileUploadLimit`:

  Ganzzahl (Default: 0). Mit diesem Parameter geben Sie die Maximalanzahl erfolgreicher Uploads pro Durchgang an. Der Default, 0, ermöglicht beliebig viele erfolgreiche Uploads pro Durchgang.

- `fileQueueLimit`:

  Ganzzahl (Default: 0). Dieser Parameter definiert die Höchstzahl von Dateien, die sich in der Upload-Warteschlange befinden können.

- `id`:

  Zeichenkette (Default: Zufällige Zeichen). Dieser Parameter dient zur Verbindung mit einer Download-Warteschlange. Lesen Sie auch die Erklärung zum Paramter `bindToUpload` der Processing-Instruction <?imperia swf_upload_queue?> im folgenden Abschnitt.

<?imperia swf_upload_queue?>

Diese PI generiert im Upload-Template eine grafische Warteschlange für die Uploads der ausgewählten Dateien. Der Benutzer sieht dort den Fortschritt des Upload-Vorgangs und kann gezielt Einfluss darauf nehmen. Die Konfiguration der Warteschlange nehmen Sie mit folgenden Parametern vor:

- `bindToUpload`:

  Zeichenkette (Default: ''). Bindender Parameter. Geben Sie die ID des Flash-Uploads ein, mit dem Sie die Warteschlange verknüpfen wollen. Diese ID steht im Parameter id der Processing-Instruction `<?imperia swf_upload?>`. Ohne den Parameter ist die Darstellung einer Warteschlange nicht möglich.

- `showHeader`:

  Boolescher Wert (Default: 1). Mit diesem Paramter schalten Sie die Anzeige von Spaltenüberschriften in der Upload-Warteschlange ein bzw. ab.

- `order`:

  Zeichenkette (Default: 'name,status,remove,conflicts'). Geben Sie die Spaltenüberschriften der Upload-Warteschlange als kommaseparierte Zeichenkette in der Reihenfolge an, in der sie erscheinen sollen. Es dürfen in der Zeichenkette keine Leerzeichen enthalten sein! Folgende Spalten stehen zur Verfügung:

  - `name`:

    Name der hochzuladenden Datei. Anklicken, um einen neuen Namen einzugeben, unter dem Imperia die Datei speichern soll.

  - `status`:

    Zeigt einen Ladebalken für den jeweiligen Upload-Status an.

  - `start`:

    Blendet einen Button ein, mit dem sich der Upload der betreffenden Datei gezielt starten lässt.

  - `stop`:

    Blendet einen Button ein, mit dem sich ein laufender Upload-Vorgang abbrechen lässt.

- `remove`:

  Blendet einen Button ein, mit dem sich ein laufender Upload unterbrechen und die betreffende Datei aus der Warteschlange entfernen lässt.

- `conflicts`:

  Blendet eine Spalte ein, in der das Ergebnis der Konfliktprüfung angezeigt wird. Besteht eine gleichnamige Datei bereits, wird das in der Spalte vermerkt.

  Dieser Parameter ist für order notwendig! Fehlt er, funktioniert der Flash-Upload nicht.

Sollen mehrere Parameter für swf_upload_queue vergeben werden, müssen diese zwingend in einzelne Zeilen aufgeteilt werden, z.B. so:

```
<?imperia swf_upload_queue
  bindToUpload: multi
  order: name,conflicts
?>
```

3.3.12 Arrayblocks

Arrayblocks definieren einen oder mehrere Template-Bausteine als eine Werte-Matrix. Die Werte, die in diese Bausteine eingegeben werden, werden als Elemente des Arrayblocks gespeichert. Anders als Imperiablocks und Flexmodule erzeugt ein Arrayblock keinen neuen Namensraum. Aus diesem Grund dürfen die in einem Arrayblock verwendeten Meta-Variablennamen nicht für andere Meta-Variablen im Template verwendet werden.

Wird der Index für einen Arrayblock mit dem Parameter `INDEX` manuell gesetzt, muss er auch für alle anderen Arrayblocks im Template manuell gesetzt werden. Ein Vermischen von automatisch und manuell vergebenen Arrayblock-Indizes muss vermieden werden.

Wird kein Parameter zur Bestimmung der Länge eines Arrays (`LENGTH` oder `KEY`) angegeben, werden alle referenzierten Meta-Variablen geprüft, um die Anzahl der anzuzeigenden Instanzen eines Arrayblocks zu ermitteln. Es wird dann die Instanzen-Anzahl der Meta-Variablen mit dem längsten Array für alle anderen Arrayblocks übernommen. Unter bestimmten Voraussetzungen kann dies zu Problemen führen, so dass wir empfehlen, einen Parameter zum Festlegen der Instanzen-Anzahl, entweder `KEY` oder `LENGTH`, zu verwenden. Lesen Sie hierzu auch die Beschreibung dieser Parameter weiter unten.

Die Syntax für einen Arrayblock ist wie folgt:

```
<!--ARRAYBLOCK-->
  beliebiger HTML-Code
<!--/ARRAYBLOCK-->
```

Parameter werden im öffnenden Arrayblock-Tag übergeben und durch einen Doppelpunkt vom Schlüsselwort bzw. von anderen Parametern getrennt.

Beispiel:

```
<!--ARRAYBLOCK:INDEX=4:LENTGTH=3-->
  <input name="IMPERIA:array_text" type="text" size="30" value="" />
<!--/ARRAYBLOCK-->
```

Die hier übergebenen Parameter bestimmen den Index des Arrayblocks (INDEX=4) sowie die maximale Länge des Arrays (LENGTH=3), also wie viele Instanzen der Arrayblocks erzeugt werden.

Folgende Parameter stehen zur Verfügung:

INDEX=*N*

Dieser Parameter setzt den Index des Arrayblocks.

LENGTH=*N*

Dieser Parameter bestimmt die maximale Anzahl der Array-Instanzen.

KEY=*Meta-Variable*

Mit dem Parameter `KEY` bestimmen Sie die Instanzen-Anzahl für alle Arrayblocks im Template mittels einer explizit festgelegten Meta-Variablen.

Folgende "Kommentar-Elemente" stehen Ihnen in Arrayblöcken zur Verfügung:

ARRAY_INDEX

Diese Variable enthält den Index des aktuellen Arrayblocks in einem Template beginnend mit `0`, sofern Sie mit dem Parameter *INDEX* keinen festen Index vergeben haben.

ARRAY_POSITION

Diese Variable enthält die Position der Instanz innerhalb eines Arrayblocks beginnend mit `0`.

ARRAY_POSITIONN

Diese Variable enthält, ähnlich wie `ARRAY_POSITION`, die Position der Instanz, addiert jedoch den dezimalen Wert `N` zur Position hinzu.

XX-ARRAY-textfeld

Diese Variable enthält den Wert der Arrayblock-Meta-Variablen `textfeld` an der aktuellen Position.

Beispiel:

```
<!--ARRAYBLOCK-->
Text:
<input name="IMPERIA:text" type="text" size="30" value="" /><br />
URL:
<input name="IMPERIA:url" type="text" size="30" value="" /><br />
<!--/ARRAYBLOCK-->
```

Der Beispielcode erzeugt im EDIT-Modus einen Arrayblock mit zwei Eingabefeldern. Jedes davon erhält einen eigenen Array. Mit Hilfe von Buttons können beliebig viele Instanzen erzeugt werden, die jeweils oberhalb der Ursprungsinstanz plaziert werden.

3.3.12.1 Zugriff auf Array-Instanzen

Außerhalb eines Arrayblocks werden Daten in Arrayblocks mit Hilfe einer Klammersyntax referenziert. Hierbei enthält die Klammer die Position der gewünschten Array-Instanz im Arrayblock. Die Nummerierung der Array-Elemente beginnt mit `0`. Die entsprechende Syntax ist wie folgt:

<!--XX-*Meta-Variablenname*[*Instanzposition*]-->

Beispiel:

```
<!--ARRAYBLOCK:LENGTH=3-->
  <input name="IMPERIA:text" type="text" size="30" value="" />
<!--/ARRAYBLOCK-->

Referenz: <!--XX-text[2]--><br />
```

Der HTML-Kommentar wird ersetzt durch den Inhalt der dritten Instanz des Arrayblock-Meta-Variable `text`.

Soll innerhalb eines Arrayblocks auf Meta-Variablen zugegriffen werden, die ebenfalls im Arrayblock stehen, ist keine Klammersyntax erforderlich. Hier wird wie mit Hilfe der Variable `XX-ARRAY-`*Meta-Variablenname* auf eingegebenen Content zugegriffen. Beispiel:

```
<!--ARRAYBLOCK:LENGTH=3-->
  <input name="IMPERIA:text" type="text" size="30" value="" />
  <!--XX-ARRAY-text-->
<!--/ARRAYBLOCK-->
```

Der Abschnitt zwischen <!--ARRAYBLOCK--> und <!--/ARRAYBLOCK--> wird im Template-Prozessor durch das Modul „Array“ mit dem Inhalt der Arrayblock-Meta-Variablen `text` ersetzt.

3.3.13 One-Click-Edit

One-Click-Edit ist eine Imperia Browser-Erweiterung. Redakteure können damit beispielsweise beim Betrachten einer veröffentlichten Seite auf dem Live-System das betreffende Dokument direkt zum Bearbeiten aus dem Archiv importieren. Einzelheiten zur Handhabung von One-Click-Edit finden Sie im Abschnitt "**One-Click-Edit**" des Benutzerhandbuchs und zu dessen Konfiguration im ebenfalls "**One-Click-Edit**" benannten Abschnitt des Administrationshandbuchs.

Sie haben folgende Möglichkeiten, die für One-Click-Edit relevanten Informationen in den fertigen Dokumenten zu speichern:

```
<!--IMPERIA:LIVEEDIT-->
<!--IMPERIA:ONECLICKEDIT-->
<!--IMPERIA:ONECLICKEDIT_FILE-->
<!--IMPERIA:ONECLICKEDIT_DOCUMENT-->
```

Mit dem Kommentar `<!--IMPERIA:LIVEEDIT-->` veranlassen Sie, dass Imperia die relevanten Informationen in einem eigenen Namespace speichert. In den publizierten Dokumenten erscheint beispielsweise folgender Code:

```
<imp:live-info sysid="12345678-1234-abcd-efgh-1234567890ab"
node_id="/2/3/13" />
```

Hinweis:

Dieser spezielle Namespace wird von den meisten HTML-Validierungstools als ungültig eingestuft.

Der Kommentar `<!--IMPERIA:ONECLICKEDIT-->` erzeugt im veröffentlichten Dokument ein Meta-Tag mit den relevanten Informationen:

```
<meta name="X-Imperia-Live-Info"
content="12345678-1234-abcd-efgh-1234567890ab/2/3/13" />
```

Hinweis:

Diese Notation entspricht den XHTML-Spezifikationen.

Die Variante `<!--IMPERIA:ONECLICKEDIT_FILE-->` ist im Zusammenhang mit Copy-Seiten interessant. Hierbei enthält das Meta-Tag im fertigen Dokument zusätzlich Pfad und Dateinamen der entsprechenden Kopie:

```
<meta name="X-Imperia-Live-Info"
content="12345678-1234-abcd-efgh-1234567890ab/2/3/13-/path/to/document/filename.html" />
```

So ist es möglich, beispielsweise mit der OCE-Funktion "**Live-Löschen**" nur die jeweilige Kopie zu entfernen. Die übrigen Kopien sowie das Masterdokument (falls freigeschaltet) verbleiben dann auf dem Live-System. Wenn keine Copy-Seiten vorhanden sind, hat diese Variante keine besonderen Auswirkungen.

Der Kommentar <!--IMPERIA:ONECLICKEDIT_DOCUMENT--> ist synonym mit <!--IMPERIA:ONECLICKEDIT-->.

Hinweis:

Die letzten beiden Varianten des One-Click-Edit-Kommentars können Sie auch mit der Standard Variante <!--IMPERIA:ONECLICKEDIT--> *und der System-Konfigurationsvariable "*ONECLICKEDIT*" aus der Datei* site/config/system.conf *abbilden. Setzen Sie die Variable auf den Wert "file", um Copy-Seiten getrennt zu verarbeiten. Jeder andere Wert veranlasst die Verarbeitung auf Dokumentbasis. Für mehr Informationen zur Konfigurationsvariablen* ONECLICKEDIT *lesen Sie den Abschnitt "***Liste aller Variablen zur System-Konfiguration***" im Anhang des Administrationshandbuchs.*

3.3.14 Media-Asset-Management und Grafik-Upload

Media-Asset-Management und Grafik-Upload binden mit Hilfe einer speziellen Syntax Medienobjekte in ein Template ein. Abhängig davon, ob eine Grafik oder ein anderes Objekt im fertigen Dokument erscheinen soll, unterscheidet sich diese Syntax.

Um Images aus dem Media-Asset-Management oder über den Grafik-Upload in das Dokument laden zu können, wird das IMG-Tag verwendet. Andere Objekte, wie zum Beispiel Flash-Objekte, PDF-Dateien oder Audio-Dateien, können mit Hilfe des <OBJECT>- bzw. <EMBED>-Tags oder des <A>-Tags integriert werden. Lesen Sie hierzu Abschnitt 3.3.14.2 **Integration von Objekten** auf Seite 40.

Des Weiteren ist es möglich, über Processing Instructions Eigenschaften des Media-Asset-Management-Aufrufs zu bestimmen und Media-Asset-Management-Variablen (MAM-Variablen) auszulesen. Diese Variablen enthalten alle Informationen, die in den Details zu einem Image oder einem anderen Objekt in der Media-Asset-Management gespeichert sind. Eine Liste aller MAM-Variablen finden Sie in Abschnitt 3.3.14.10 **MAM-Variablen** auf Seite 47.

Hinweis:

Die Syntax für Processing-Instructions und MAM-Variablen ist aus Kompatibilitätsgründen immer mit mdb statt mam* zu notieren. So können Sie Templates aus älteren Imperia-Installationen weiterhin verwenden, ohne diese migrieren zu müssen.*

Die Bedienung des Media-Asset-Managements wird eingehend in den Kapiteln "**Media Asset Management (MAM)**" im Administrationshandbuch und "**Das Media-Asset-Management (MAM)**" im Userhandbuch behandelt, das Thema **Grafik-Upload** im Abschnitt "**Grafik-Upload**" im Userhandbuch.

3.3.14.1 Einbindung von Grafikdateien mit dem Media-Asset-Management

Die Einbindung von Grafiken in einem Dokument geschieht durch einen Aufruf des Media-Asset-Managements. Diesen setzen Sie über das SRC-Attribut des IMG-Tags um. Die resultierende Größe der Grafik im Dokument passen Sie über die Größenattribute WIDTH und HEIGHT des IMG-Tags an.

Hinweis:

Wenn Sie die Werte der Größenattribute in Anführungszeichen setzen (XHTML-konform), erzeugt das Media-Asset-Management automatisch eine entsprechend skalierte Grafikdatei. Bei Verkleinerungen wird damit gleichzeitig die Datenmenge reduziert. Wenn Sie keine Anführungszeichen setzen (nicht XHTML-konform, in HTML 4.0 Transitional erlaubt) skaliert der Browser lediglich die Darstellung der Grafik. Bei Verkleinerungen findet in diesem Fall keine Reduzierung der Datenmenge statt, da dieselbe Datei verwendet wird.

Die Syntax zur Einbindung des Media-Asset-Management-Aufrufs im Template ist wie folgt:

```
<img src="IMPERIA:Variablenname" alt="IMPERIA:alt" height="180" width="220" />
<input name="IMPERIA:alt" />
```

Abb. 3.3: Aufruf von MAM und Grafik-Upload

Im EDIT-Modus erscheint das Default-Bild, bis der Benutzer eine Grafik ausgewählt hat. Der Link **Assets** öffnet das Media-Asset-Management in einem neuen Fenster, der Link **Grafiker** öffnet den Grafik-Upload.

Die Syntax mit Processing Instructions ist zum Beispiel folgendermaßen:

```
<IMG SRC="IMPERIA:image" ALT="IMPERIA:alt" HEIGHT="180" WIDTH="300" />
  <?imperia mdb copy FILESIZE:filesize LASTMODIF_DATE:lastmod?>
  <?imperia mdb default "/imperia/md/defaults/icons/ico_smiley1.jpg"?>
</IMG>
<input name="IMPERIA:alt" /><br />
<input name="IMPERIA:filesize"> Größe in Bytes /<br />
<input name="IMPERIA:lastmod"> Letzte Modifikation /<br />
```

Dieses Beispiel enthält zwei Processing Instructions. Mit Hilfe der ersten werden MAM-Variablen ausgelesen und die Werte in Metavariablen gespeichert. Diese Werte werden in die Input-Felder eingetragen, wenn ein Benutzer ein Image auswählt. Die zweite Processing Instruction legt ein anderes Default-Bild fest.

Nach der Auswahl einer Grafik enthält das Input-Feld den Namen der Grafik-Datei, wie er im Media-Asset-Management gespeichert ist. Dieses Input-Feld ist optional und wird in diesem Beispiel dazu verwendet, dem Benutzer die Möglichkeit zu geben, einen Alternativtext für das ausgewählte Bild einzugeben. Die aus dem Media-Asset-Management ausgelesenen Informationen können in den Input-Feldern verändert werden. Diese Änderungen sind jedoch nur für das betreffende Dokument gültig und übertragen sich nicht auf das Asset im Media-Asset-Management.

3.3.14.2 Integration von Objekten

Zur Integration des Media-Asset-Managements und zur Auswahl von Objekten, die nicht mit einem <img>-Tag eingebunden werden können, müssen abhängig vom verwendeten Tag unterschiedliche Attribute verwendet werden.

Um zum Beispiel einen Link zu einem PDF-Dokument zu erzeugen und zusätzlich die Größe der Datei anzuzeigen, können Sie beispielsweise folgenden Aufruf verwenden:

```
#IF ("<!--XX-editmode-->")
  <a href="IMPERIA:MDB:download">
  <?imperia mdb copy OBJURL:pdfurl NAME:pdfname FILESIZE:pdfsize?>Datei auswählen</a>
  <input name="IMPERIA:pdfurl" type="hidden" />
  Dateigröße: <input name="IMPERIA:pdfsize" />
  Name der Datei: <input name="IMPERIA:pdfname" />
#ELSE
  <a href="IMPERIA:MDB:pdfurl"><!--XX-pdfname--></a>
#ENDIF
```

Im EDIT-Modus erzeugt dieser Code im Dokument einen Link, über den Sie das Media-Asset-Management öffnen. Unterhalb dieses Links erscheinen zwei Input-Felder, die nach dem Auswählen eines Objekts mit den entsprechenden Daten gefüllt werden.

Beachten Sie, dass Sie für jede ausgelesene MAM-Variable im Template eine Meta-Variable erzeugen müssen. Soll die Information aus dem Media-Asset-Management nicht lesbar im Dokument erscheinen, kann man, wie im Beispiel für die Meta-Variable `pdfurl`, das Eingabe-Element unsichtbar füllen (Attribut `type` auf den Wert `hidden` setzen).

In allen anderen Modi als dem SAVE-Modus wird ein Link eingefügt, der auf das entsprechende Objekt verweist und den Download startet.

Soll ein Objekt direkt im Dokument gestartet werden, beispielsweise ein Flash-Film, muss das <src>-Attribut des <embed>-Tags mit dem Pfad des Objekts gefüllt werden. Hierfür kann beispielsweise folgende Syntax verwendet werden:

```
#IF (<!--XX-editmode-->)
  <a href="IMPERIA:MDB:download">
  <?imperia mdb copy FILESIZE:size
   NAME:name?>Flash aussuchen</a><br>
  <input name="IMPERIA:size" />
  <input name="IMPERIA:name" />
#ELSE
  <object classid= [...] ID="<!--XX-name-->" width="550" height="400">
  <param name=movie VALUE="<!--XXOBJ-download-->">
  <param name=quality VALUE=high>
  <param name=bgcolor VALUE=#FFFFFF>
  <embed src="<!--XXOBJ-download-->"> [...] </embed>
  </object><p>
#ENDIF
```

Hinweis:

Ausgelassene, nicht relevante Code-Teile sind durch [...] gekennzeichnet.

Dieser Code erzeugt im EDIT-Modus einen Link, mit dessen Hilfe der Benutzer das Media-Asset-Management öffnet, um eine Datei auszuwählen. Zusätzlich werden für die Eingabe von Name und die Größe der ausgewählten Datei zwei Input-Felder angezeigt. In jedem anderen Modus wird das ausgewählte Objekt angezeigt, im Beispiel ein Flash-Film. Beachten Sie, dass Sie die Referenz auf diese Objekte durch einen besonderen Variablentyp, `XXOBJ`, herstellen müssen, damit Imperia die Pfade zu den entsprechenden Dateien auflösen kann. Lesen Sie mehr hierzu in Abschnitt 3.4.3 **Meta-Variablen referenzieren** auf Seite 54 und in Abschnitt 6.16 **XX-Variablen und XXOBJ-Variablen** auf Seite 145.

3.3.14.3 MAM-Aufruf anpassen

Folgende Eigenschaften des MAM-Aufrufs können Sie mit Hilfe von Processing Instructions angepassen:

- welche Links anzuzeigen sind
- welche Aktion zum Bearbeiten voreingestellt ist
- Texte der Links
- das Default-Bild, wenn noch kein Objekt ausgewählt ist
- nach welcher Methode Bilder verkleinert werden sollen

mdb switchbyname

Mit diesem Kommando wird über eine Meta-Variable gesteuert, welche Links im Media-Asset-Management-Aufruf im EDIT-Mode verfügbar sein sollen (**Assets** und/oder **Grafiker**). Lesen Sie hierzu auch Abschnitt 3.3.14.6.3 **Konfiguration mittels Processing Instructions** auf Seite 45.

mdb switchbyvalue

Mit diesem Kommando wird über einen Übergabe-Wert gesteuert, welche Links im Media-Asset-Management-Aufruf im EDIT-Mode verfügbar sein sollen (**Assets**, **Grafiker**, **Remove** und/oder **Actions** (Customize-Link auf dem Bild selbst). Lesen Sie hierzu auch Abschnitt 3.3.14.6.3 **Konfiguration mittels Processing Instructions** auf Seite 45.

mdb label

Mit diesem Kommando können Sie den Text des Links zum Öffnen des Media-Asset-Managements im Media-Asset-Management-Aufruf ändern.

Beispiel: `<?imperia mdb mdb_label Bilddatenbank?>`

Der Beispielcode ändert den Linktext im MAM-Aufruf von **Assets** in **Bilddatenbank**.

upload label

Mit diesem Kommando lässt sich der Text des Links zum Öffnen des Upload-Dialogs im Media-Asset-Management-Aufruf ändern.

Beispiel: `<?imperia mdb upload_label Dateisystem?>`

Der Beispielcode ändert den Linktext im MAM-Aufruf von **Grafiker** nach **Dateisystem** (siehe Screenshot).

Abb. 3.4: Media-Asset-Management-Aufruf mit geändertem Linktext

mdb actions_action

Mit diesem Kommando kann definiert werden, welche Aktion (z.B. Cropping) ausgeführt wird, wenn der Benutzer auf den Link „Anpassen“ der Bildeinfügung klickt.

```
<?imperia mdb actions_action (AssetActions_PLUGINNAME)?>
```

mdb default

Mit Hilfe dieses Kommandos wird der URL für das Default-Bild festgelegt, das im EDIT-Modus erscheint.

mdb quickupload

Wird diese Option gesetzt, wird der Grafiker-Upload durch einen unkomplizierten Schnell-Upload in das MAM ersetzt. Dabei hat der Benutzer keine Wahl, in welche MAM-Rubrik er hochladen möchte; die Rubriken-ID wird direkt im Aufruf festgelegt.

```
<?imperia mdb quickupload /3/23/651?>
```

mdb quickupload_mime_filter

Mit diesem Kommando kann definiert werden, welche Arten von Assets (z.B. Bild- oder Dateiformate) an der Stelle dieser Bildeinfügung hochladbar sind, wenn der Quickupload benutz wird.

```
<?imperia mdb quickupload_mime_filter [Regulärer Ausdruck, mit dem der Mime-Typ verglichen wird
```

mdb letterbox

Mit Hilfe dieses Kommandos wird das Verhalten beim Verkleinern eines Bildes geändert. Normalerweise wird, wenn ein Bild in einen MAM-Aufruf mit einem anderen Seitenverhältnis eingesetzt wird, ein Ausschnitt aus dem Bild hergestellt, der den Aufruf komplett ausfüllt. Gibt man die Processing Instruction „letterbox“ mit an, so wird das Bild nicht beschnitten, sondern das Bild wird eingepasst und der übrigbleibende Raum des Aufrufs wird mit einer Farbe gefüllt.

Als Parameter muss ein HTML-Farbwert ohne Rautenzeichen (z.B. „606060“) oder „trans“ angegeben werden; „trans“ bewirkt bei GIF- und PNG-Bildern, dass die Balken im Bild transparent ausgegeben werden. Da das JPEG-Format keine Transparenz kennt, werden die Balken bei JPEG-Bildern bei Angabe von „trans“ in schwarz wiedergegeben.

```
<?imperia mdb letterbox 606060?>
<?imperia mdb letterbox trans?>
```

3.3.14.4 MAM-Fenster anpassen

Folgende Eigenschaften des MAM-Fensters können mit Hilfe von Processing Instructions angepasst werden:

- Höhe und Breite
- die automatisch geöffnete Sektion (IMAGES, CONTENT, ...)
- die automatisch geöffnete MAM-Rubrik

Höhe und Breite anpassen

Die Höhe und Breite des MAM-Fensters können mit Hilfe von Processing Instructions (kurz PI) im MAM-Aufruf angepasst werden.

```
<?imperia mdb window_width 500?>
<?imperia mdb window_height 500?>
```

Diese beiden Processing Instructions setzen die Breite und Höhe des MAM-Fensters auf jeweils 500 Pixel.

Außerdem können Sie mit PIs bestimmen, welche Rubrik beim Aufruf der Media-Asset-Management beim Aufruf geöffnet werden soll. Die Bestimmung der Rubrik ist in Abschnitt 3.3.14.9 **Bestimmte MAM-Rubrik öffnen** auf Seite 47 beschrieben.

3.3.14.5 Weitere Processing Instructions

Mit Hilfe von Processing Instructions werden die Werte von MAM-Variablen ausgelesen und in Meta-Variablen gespeichert. Es können beliebig viel Processing Instructions in einem Media-Asset-Management-Aufruf verwendet werden. Auch die Reihenfolge der Processing Instructions ist beliebig.

Beispiel:

```
<?imperia MAM Kommando1 MAM-Variable1:Meta-Variable1 MAM-Variable2:Meta-Variable2?>
<?imperia MAM Kommando2 MAM-Variable3:Meta-Variable3 MAM-Variable4:Meta-Variable4?>
```

Hinweis:

Beachten Sie, dass hier zwar **MAM**-*Kommandos und* **MAM**-*Variablen beschrieben sind, Sie aber im Quelltext das Kürzel mdb verwenden müssen, da die Kommandos und Variablen aus Kompatibilitätsgründen so implementiert sind.*

Folgende Kommandos stehen als Bestandteile von Processing Instructions zur Verfügung:

mdb copy: Werte in Meta-Variablen kopieren

Dieses Kommando sorgt dafür, dass Werte aus dem Media-Asset-Management in eine Meta-Variable kopiert werden.

mdb mcopy: Mehrwertige Variablen in Meta-Variablen kopieren

Der Befehl mcopy ist eine Erweiterung des copy-Befehls, der auch mit mehrwertigen Variablen (Arrays) funktioniert. Während copy immer nur den ersten Wert überträgt, setzt mcopy alle Werte in das Imperia-Eingabefeld ein.

Beispiel:

```
<?imperia mdb mcopy name:name width:breite?>
```

mdb setsrc *MAM-Variable*: *Meta-Variable*: *Imagename*

Dieses Kommando setzt das SRC-Attribut der Grafik **Imagename** auf den Wert der Meta-Variablen **Meta-Variable**. Des Weiteren aktualisiert sich das SRC-Attribut jedes Mal, wenn ein Benutzer ein neues Objekt aus der Media-Asset-Management wählt. Zu diesem Zweck muss eine Verbindung zwischen den Meta-Variablen und der Media-Asset-Management-Variable (MAM-Variable) etabliert werden.

Zuweisungspaar *MAM-Variable*:*Meta-Variable*

Mit einem solchen Zuweisungspaar wird der Wert einer MAM-Variable einer Meta-Variablen zugewiesen und kann so im Dokument weiterverarbeitet werden. Zuweisungspaare bestehen aus dem Namen der MAM-Variable und dem Namen der Meta-Variablen, getrennt durch einen Doppelpunkt. Eine Processing Instruction kann mehrere Zuweisungspaare enthalten.

33.14.6 Links im MAM-Aufruf anpassen

Standardmäßig beinhaltet der MAM-Aufruf sowohl einen Link zum Starten des Media-Asset-Managements als auch einen Link zum Öffnen des Grafik-Uploads. Meist wird aber nur das Media-Asset-Management oder der Grafik-Upload alleine verwendet, so dass der nicht verwendete Link bei den Benutzern Verwirrung stiften kann.

Des Weiteren ist es manchmal sinnvoll, den Benutzern die Möglichkeit zu nehmen, aus ihrem Dateisystem über den Grafik-Upload Bilder in das Dokument zu laden, wenn sie ausschließlich Media-Asset-Management verwenden sollen.

Um zu steuern, welche Links im MAM-Aufruf angezeigt werden, gibt es folgende Möglichkeiten

- für das gesamte System über eine SYSTEM_CONF-Variable MD-DEFAULT-LINKS
- über die Meta-Variable `__imperia_MDB_linkswitch`
- aus dem Template heraus mit Hilfe der Processing Instructions `switchbyvalue` oder `switchbyname`

Hierbei muss beachtet werden, dass die Einstellung der Systemkonfigurations-Variablen von der Einstellung über die Meta-Variable `__imperia_MDB_linkswitch` überschrieben wird. Die Einstellung der Meta-Variablen `__imperia_MDB_linkswitch` wiederum wird von den Processing Instructions überschrieben.

33.14.6.1 Konfiguration mittels einer Systemkonfigurations-Variablen

Die Einstellung über die Systemkonfigurations-Variable wird von allen anderen Einstellungen überschrieben.

In der Datei `site/config/system.conf` wird die Variable `MD-DEFAULT-LINKS` eingetragen. Sie kann folgende Werte haben:

none

Keiner der beiden Links wird angezeigt. Der Rahmen um das Default-Bild wird ebenfalls nicht angezeigt und das Bild ist nicht editierbar.

mdb

Es wird nur der Link **MedienDB** angezeigt.

upload

Es wird nur der Link **Grafiker** angezeigt.

mdb, upload

Beide Links werden angezeigt. Dies ist die Default-Einstellung, wenn die Variable nicht gesetzt wird.

Die Variable wird nur ausgewertet, wenn sie gesetzt ist. Ist sie nicht gesetzt, ist das gleichbedeutend mit dem bisherigen Verhalten: es wird sowohl der Link **Assets** als auch der Link **Hochladen** angezeigt.

331462 Konfiguration mittels einer Imperia-Variablen

Bei dieser Variante wird die Imperia-Variable `__imperia_mdb_linkswitch`, die im Dokument auf den entsprechenden Wert gesetzt sein muss, ausgewertet. Das System erzeugt diese Variable nicht automatisch, sie muss entweder über eine Metadatei oder über einen Workflow-Schritt gesetzt werden.

Die Werte und deren Bedeutung sind identisch mit denen der SYSTEM_CONF-Variablen.

331463 Konfiguration mittels Processing Instructions

Mit Hilfe dieser Variante kann im Aufruf der Media-Asset-Management im Template bestimmt werden, welche Links angezeigt werden sollen. Hierzu gibt es zwei Alternativen:

- über einen in der Processing Instruction gesetzten Übergabe-Wert
- über den Wert einer Meta-Variablen, deren Wert an anderer Stelle entsprechend gesetzt wird

Anpassung durch einen Übergabe-Wert einer Processing Instruction

Die Anpassung durch einen Übergabe-Wert in der Processing Instruction geschieht mit folgender Syntax:

```
<?imperia mdb switchbyvalue Wert?>
```

Für den Platzhalter *Wert* wird *mdb* oder *upload* angegeben.

Anpassung durch den Wert einer Meta-Variablen

Die Anpassung durch einen Übergabe-Wert in der Processing Instruction geschieht mit folgender Syntax:

```
<?imperia mdb switchbyname Meta-Variablenname?>
```

Für den Platzhalter *Meta-Variablenname* wird der Name der Meta-Variablen angegeben. Der Wert dieser Meta-Variablen muss an anderer Stelle mit `mdb`, `upload`, `remove` oder `actions` oder mit Komma getrennten Kombinationen daraus (z.B. `mdb, upload`) gesetzt werden.

Achtung:

Die Processing Instructions `switchbyvalue` *und* `switchbyname` *dürfen nicht gleichzeitig in einem MAM-Aufruf verwendet werden.*

33.14.7 CGI-Skript bei Objekt-Übertragung ausführen

Bei der Übertragung eines Objekts vom Media-Asset-Management in das Dokument kann ein CGI-Skript ausgeführt werden. Das Default-Skript ist `site_md_resize.pl`.

Das Skript wird über eine Processing Instruction angegeben:

```
<?imperia mdb script Skriptname?>
```

Um diese Funktion nutzen zu können, sind folgende Dinge zu beachten:

- Das Skript sollte die Übergabe-Werte der Subroutine *evaluate_mdb_cgiform* auswerten. Einer dieser Übergabe-Werte enthält einen JavaScript-Aufruf als String. Dieser JavaScript-Aufruf befindet sich in einem versteckten Frame und wird ausgeführt, nachdem das Skript durchgelaufen ist.
- Anschließend sollte das Skript die gewünschten Aktionen ausführen. Es könnte beispielsweise das Bild manipulieren, indem es eine Kopie des ausgewählten Bildes in einer anderen Bildgröße erzeugt und speichert, den JavaScript-Aufruf ändert oder ergänzt.

- Da die HTML-Ausgabe, die das Skript erzeugt, in dem unsichtbaren versteckten Frame angezeigt wird, ist es wenig sinnvoll, eventuelle Fehlermeldungen als HTML auszugeben, da diese nicht lesbar wären. Solche Fehlermeldungen sollten per JavaScript als Message-Box dargestellt werden. Weiterhin kann es notwendig sein, die Fehlermeldung zu maskieren.
- Das Skript sollte beendet werden, nachdem es die folgenden beiden Zeilen Code ausgeführt hat:

```
my $page = Imperia::Display::MediaDB->new;
print $page->getHiddenFrame($jscript);
```

  wobei die Variable `$jscript` als String den ursprünglichen oder einen modifizierten JavaScript-Aufruf des Media-Asset-Managements enthält, der die Daten beinhaltet, die auf das ausgewählte MAM-Objekt verweisen.

Während man ein solches CGI-Skript entwickelt, bietet es sich an, die Konstante `DEBUG` im Skript `site_md.pl` auf einen wahren Wert zu setzen. Ein verstecktes MAM-Frame wird daraufhin sichtbar, so dass Debug-Nachrichten und eventuelle Internal Server Errors angezeigt werden können.

3.3.14.8 JavaScript-Funktion bei Objekt-Übertragung ausführen

Bei der Übertragung eines Objekts von der Media-Asset-Management in das Dokument kann eine JavaScript-Funktion ausgeführt werden. Die Standard-Funktion ist `__mdbApplyValues()` (siehe auch Subroutine `get_mam_javascript` in diesem Modul für die Definition der Standard-Funktion).

Um dieses Feature zu nutzen, gehen Sie wie folgt vor:

1. Erzeugen Sie die Datei *Funktions-Name* im Verzeichnis `/imperia/md/develop/javascript/`. Diese Datei wird automatisch zum Haupt-Frame des MAM-Fensters hinzugefügt. Bitte beachten Sie, dass ein Suffix `.js` an die erzeugte Datei angehängt werden muss.
2. Definieren Sie in dieser Datei eine Funktion gleichen Namens wie die Datei selbst. Diese Funktion wird von dem versteckten Frame aus aufgerufen, nachdem das Skript `site_md_resize.pl` oder ein angepasstes Skript (siehe Abschnitt 3.3.14.7 **CGI-Skript bei Objekt-Übertragung ausführen** auf Seite 45) ausgeführt wurde.

   Diese Funktion muss die Werte in die `value`-Attribute der Formular-Felder und in die `src`-Attribute der `img`-Tags eintragen.

   Es werden die Argumente *list* und *is_delector* automatisch an die Funktion übergeben:

   Das erste Argument *list* ist ein Array-Objekt, dessen Elemente wiederum Array-Objekte mit jeweils drei String-Elementen sind. Das Format der Sub-Arrays ist wie folgt:

```
[TYPE,NAME,VALUE]
```

   wobei *TYPE* `src` bei Bildern bzw. `value` bei anderen Objekten ist. *NAME* bestimmt den Namen des Formular-Feldes bzw. den Namen eines `img`-Tags. *VALUE* bestimmt den Wert, der an das durch *TYPE* bestimmte Attribut übergeben wird.

   Das zweite Argument *is_delector* ist ein boolescher Wert, der dann wahr ist, wenn die Funktion über den **Objekt löschen**-Button im Template aufgerufen wurde. Falls die angepasste Funktion weitere `src`- oder `value`-Attribute setzt, ist es möglicherweise nötig, zusätzliche Aktionen auszuführen. Die Standard-Funktion benötigt den `is_delector`-Wert nicht.

Beispiel: Angenommen, in einem Template ist folgender MAM-Aufruf implementiert:

```
<img name="IMPERIA:my_width" width="100">
  <?imperia mdb copy COPYRIGHT:copyright?>
  <?imperia mdb function mymamFunction?>
</img>
```

Dies wird zu folgendem Funktionsaufruf umgesetzt:

```
mymdbFunction([['src',
          'my_img',
          '/imperia/md/images/spiders/image_100x150.jpg'],
          ['value',
           'copyright',
           '2004, Cross Spider Co.']]);
```

Die Funktion `__mdbApplyValues` kann als Ausgangspunkt für eigene Funktionen verwendet werden. Diese Funktion ist innerhalb des Hauptframes der MAM-Fensters immer definiert, auch dann, wenn eine angepasste Funktion verwendet wird. Aus diesem Grund kann diese Funktion auch aus der angepassten Funktion heraus aufgerufen werden. Daraus folgt aber auch, dass der Name der Funktion ein reservierter Name ist. Hinzu kommt, dass auch `__imperia_node_id` eine reservierte Variable ist. Sie wird verwendet, um Fehler zu vermeiden, falls versehentlich auf Objekte in der MAM geklickt wird, nachdem das editierte Dokument im aufrufenden Fenster bereits wieder geschlossen wurde. Es sollte gewährleistet sein, dass angepasste Funktionen solche Fehlerquellen ebenfalls abfangen.

3.3.14.9 Bestimmte MAM-Rubrik öffnen

Um eine bestimmte Rubrik des Media-Asset-Managements über den Aufruf zu öffnen, verwenden Sie folgende Processing Instructions:

```
<?imperia mdb directory /Pfad/zum/Verzeichnis?>
<?imperia mdb directory numerische Verzeichnis-ID?>
```

mdb directory: Pfad zum Verzeichnis

Der Pfad zum Verzeichnis wird ausgehend vom Document-Root angegeben, jedoch kann der führende Pfadbestandteil `/imperia/md` auch weggelassen werden.

Sollte das angegebene Verzeichnis in der Media-Asset-Management nicht vorhanden sein, dann wird der letzte Teil des angegebenen Verzeichnis-Strings abgeschnitten und der daraus resultierende Pfad wird genommen.

Beispiel: <?imperia mdb directory /imperia/md/images/marketing?>

Beim Öffnen der MDB wird automatisch die Rubrik *marketing* geöffnet.

3.3.14.10 MAM-Variablen

Im Folgenden finden Sie eine Liste aller MAM-Variablen, die mit Hilfe von Processing Instructions aus dem Media-Asset-Management ausgelesen werden können. Neben dem Namen der Variablen erscheint in Klammern der verwendete Datentyp.

NAME (char(255))

Diese Variable enthält den Namen des Objekts, der beim Upload oder im Dialog **Objekteigenschaften ändern** definiert wird. Wird dem Objekt kein Name gegeben, dann enthält diese Variable den Dateinamen ohne Endung.

DATATYPE (char (255))

Diese Variable enthält den Datentyp, der anhand der Dateiendung bestimmt wird. Handelt es sich beispielsweise bei dem Objekt um ein JPEG-Bild, enthält die Variable den Wert `jpg`.

FILESIZE (int)

In dieser Variable wird die Größe des Objekts in Bytes gespeichert.

OBJURL (char (255))

Diese Variable enthält die Media-Asset-Management-URL des Objekts ausgehend vom DocumentRoot-Verzeichnis des Servers.

PREVFILE (char (255))

Dieser Parameter enthält die URL des Thumbnails. Im Falle eines Flash4-Films ist dies zum Beispiel folgende URL: `/imperia/md/defaults/icons/ico_flash4.gif`.

PREVSRC (char(255))

Diese Variable enthält die URL des Objekts. Sie kann nicht ausgelesen werden, sondern wird lediglich in Verbindung mit dem Schlüsselwort `setsrc` verwendet.

SELLPRICE (char(255))

Diese Variable enthält den Wert des Feldes `SELLPRICE` aus dem Dialog **Objekteigenschaften ändern**.

VIEWPRICE (char(255))

Diese Variable enthält den Wert des Feldes `VIEWPRICE` aus dem Dialog **Objekteigenschaften ändern**.

PLAYLENGTH (char(100))

Diese Variable enthält die Spiellänge eines Objektes, wenn es sich dabei um einen AVI-Film, ein Flash-Movie oder dergleichen handelt. Sie kann beim Upload des Objekts oder im Dialog **Objekteigenschaften ändern** definiert werden.

XRES (int)

Diese Variable enthält die horizontale Anzahl der Pixel eines Bildes. Sie wird im Feld **X-PIXEL** beim Upload des Objekts oder im Dialog **Objekteigenschaften ändern** definiert.

YRES (int)

Diese Variable enthält die vertikale Anzahl der Pixel eines Bildes. Sie wird im Feld **Y-PIXEL** beim Upload des Objekts oder im Dialog **Objekteigenschaften ändern** definiert.

DPI (int)

Diese Variable enthält die Auflösung eines Bildes. Sie wird im Feld **DPI** beim Upload des Objekts oder im **Dialog Objekteigenschaften** ändern definiert.

COPYRIGHT (char(255))

Diese Variable enthält den Wert des Feldes **COPYRIGHT** aus dem Dialog **Objekteigenschaften ändern**.

UPLOAD_DATE (date)

Diese Variable enthält das Datum, an dem das Objekt in die Media-Asset-Management übertragen wurde.

LASTMODIF_DATE (date)

Diese Variable enthält das Datum der letzten Änderung der Objekteigenschaften.

COMMENT1 (char(255))

Diese Variable enthält den Wert des Feldes **COMMENT1** aus dem Dialog **Objekteigenschaften ändern**.

COMMENT2 (char(255))

Diese Variable enthält den Wert des Feldes **COMMENT2** aus dem Dialog **Objekteigenschaften ändern**.

3.3.14.11 Beispiel: Thumbnail mit Link auf Originalbild

Ein Thumbnail mit einem Link auf das Originalbild kann mit folgendem Code erzeugt werden:

```
<input name="IMPERIA:myicon_url" type="hidden">
<a href="IMPERIA:MDB:mylink">
  <?imperia mam copy PREVFILE:myicon_url?>
  <?imperia mam setsrc PREVSRC:myicon_url:myicon_img?>
```

```
  <img name="IMPERIA:myicon_img" src="/imperia/md/defaults/icons/
  ico_smiley1.gif">
</a>
```

In diesem Beispiel wählt der Benutzer eine Grafik aus dem Media-Asset-Management. Im fertigen Dokument wird der Thumbnail zu diesem Objekt angezeigt. Dieser Thumbnail ist mit einem Link versehen, der das Originalbild öffnet.

Hierzu muss zunächst die URL des Thumbnails aus dem Media-Asset-Management ausgelesen und in einer Meta-Variablen gespeichert werden. Dies wird durch ein verstecktes Eingabefeld (erste Codezeile im Beispiel) und eine Processing Instruction (dritte Zeile im Codebeispiel) erreicht. Mit der Processing Instruction wird der Wert der MAM-Variable `PREVFILE` ausgelesen und in der Meta-Variablen `myicon_url` gespeichert.

Des Weiteren muss das `href`-Attribut des Links auf die URL des ausgewählten Originalbildes und das `src`-Attribut des `img`-Tags auf die URL des Thumbnails gesetzt werden. Dies geschieht durch die zweite Processing Instruction. Übersetzt bedeutet diese Processing Instruction in etwa folgendes:

Setze das `href`-Attribut des `a`-Tags auf die URL des ausgewählten Assets. Trage gleichzeitig die URL des Thumbnails dieses Assets in das `src`-Attribut des `img`-Tags mit dem Namen `myicon_img` ein.

Der Beispielcode kann auch dazu verwendet werden, um einen Download-Link zu erzeugen. Das `img`-Tag zeigt dann das Icon, das im Media-Asset-Management dem Objekttyp (Flash, PDF, MPEG etc.) zugeordnet ist.

3.3.15 Convert-Plug-Ins (postconvert)

Imperia stellt eine Reihe von Plug-Ins zur Verfügung, mit deren Hilfe Konversionen immer dann ausgeführt werden, wenn ein Template geparst wird. Dies ist beispielsweise dann der Fall, wenn aus dem Archiv heraus eine Vorschau auf ein Dokument erzeugt wird oder wenn ein Dokument editiert wird. Dies ist insbesondere bei der Verwendung von Multilanguage-Templates praktisch.

Zurzeit sind folgende Convert-Plug-Ins im Lieferumfang enthalten:

- **SafeUnicode** zum Umwandeln von UTF-8 Text nach „us-ascii“ oder „iso-8859-1“.
- **Recode** zum Umwandeln von Text von einem Codeset in ein anderes.
- **PHPExec** zum Ausführen von PHP-Code im Template.
- **Null** als Rumpf für eigene Plug-Ins.

3.3.15.1 SafeUnicode-Convert-Plug-In

Das Plug-In wandelt Sonderzeichen in UTF-8-encodierten Texten in HTML-Entities um. Der Text bleibt dabei im Zeichensatz UTF-8, aber wegen der Verwendung der Entities sind Fehler bei der Darstellung von Sonderzeichen ausgeschlossen. So erreichen Sie, dass selbst nicht unicode-fähige Software Texte im UTF-8 Zeichensatz immer korrekt verarbeitet. Mit einem optionalen Parameter können Sie festlegen, ob das Ergebnis der Konvertierung US-ASCII- oder ISO-8859-1-kompatibel sein soll. Außerdem besteht die Möglichkeit, die Verwendung numerischer Entities zu erzwingen. Die Einstellungen für den Konvertierungsvorgang nehmen Sie beim Aufruf des Plug-Ins im Template vor:

```
<!--postconvert:SafeUnicde(Parameter|Parameter=Wert|...)-->
```

Die Parameter können Sie wahlweise mit oder ohne einen booleschen Wert angeben. Gegenwärtig verarbeitet das Plug-In zwei Parameter:

numerical

Ist dieser Parameter gesetzt, verwendet das Plug-In ausschließlich numerische Entities für Sonderzeichen, also zum Beispiel *&* statt *&*. Diese Variante bietet geringfügig höhere Performance und bessere Kompatibilität, da einige Browser neuere benannte Entities wie beispielsweise € ignorieren.

latin1

Wenn Sie diesen Parameter übergeben, wandelt das Plug-In nur Zeichen um, die in der Zeichentabelle nach den ersten 255 Zeichen liegen. Den Bereich der Steuerzeichen (128-159) ignoriert das Plug-In. Verwenden Sie den Paramter nicht, wenn der bearbeitete Text kompatibel zu den Zeichensätzen CP-1252 (Windows-1252) oder ISO-8859-15 (Latin-9) sein soll.

Wenn Sie keine Parameter verwenden, arbeitet das Plug-In mit den Default-Einstellungen. Das heißt es konvertiert, Zeichen oberhalb der Position 127 in Entities (US-ASCII-kompatibel) und benutzt dazu, sofern vorhanden, benannte Entities.

Beispiel für einen Aufruf mit Werten:

```
<!--postconvert:SafeUnicde(latin1|numerical=1)-->
```

3.3.15.2 Recode-Convert-Plug-In

Dieses Plug-In konvertiert Text in einem Template von einem Charset in ein anderes Charset.

Das Plug-In binden Sie mit folgender Syntax ins Template ein:

```
<!--postconvert:Recode(Quell-Codeset|Ziel-Codeset)-->
```

Beispiel:

```
<!--postconvert:Recode(ISO-8859-1|UTF-8)-->
```

Mit diesen Einstellungen konvertiert das Plug-In Texte von ISO-8859-1 nach UTF-8.

3.3.15.3 PHPExec-Convert-Plug-In

Dieses Plug-In startet einen PHP-Interpreter und übergibt diesem den PHP-Code aus dem Imperia-Template.

Um das Plug-In zu aktivieren, verwenden Sie folgende Syntax:

```
<--postconvert:PHPExec(Meta-Variablen)-->
```

Wenn der Templateprozessor diesen Code in einem Template findet, startet er das PHPExec-Plug-In. Mit dem optionalen Parameter *Meta-Variablen* übergeben Sie dem Plug-In eine kommaseparierte Liste von Meta-Variablen, die dieses als normale CGI-Form-Variablen an den PHP-Interpreter weitergibt.

Beispiel:

```
<!--postconvert:PHPExec(template, directory, filename)-->
```

Die Meta-Variablen erscheinen dem PHP-Interpreter als Formular-Variablen. Enthält eine solche Meta-Variable mehrere Werte, werden auch mehrere Variablen-Instanzen an den Interpreter übergeben. Dem String `[]` wird der Variablenname hinzugefügt, wie es der PHP-Interpreter erwartet.

Bei dem Plug-In handelt es sich um einen **Post**-Prozessor. Alle Ersetzungen im Template sind also bei der Übergabe des Codes an den PHP-Interpreter bereits erfolgt. Dadurch können Sie innerhalb des PHP-Codes Imperia-Variablen verwenden.

Beispiel:

```
<?php
        $imperia_template = '<!--XX-template-->'
?>
```

Damit Sie das Plug-In einsetzen können, müssen Sie Typ und Pfad des PHP-Interpreters in der Systemkonfiguration von Imperia angeben. Dazu nutzen Sie die Variablen `PHP_CLI` und `PHP_PATH`. Mehr Einzelheiten zu diesen beiden Variablen entnehmen Sie bitte dem Abschnitt **Gruppe PHP** im Kapitel **Liste aller Variablen zur Systemkonfiguration** im Anhang des Administrationshandbuchs

3.3.15.4 Null-Convert-Plug-In

Dieses Plug-In dient als Rumpf für Ihre eigenen Plug-In-Entwicklungen. Es enthält die Funktion `convert()`, die Ihren Code aufnimmt.

Hinweis:

Eine weiterführende Beschreibung und Details finden sich in der Perldoc: /site/modules/core/Template/Convert.pm /site/modules/core/Dynamic/Convert/PHPExec.pm /site/modules/core/Dynamic/Convert/Recode.pm

3.3.16 Flexmodule

Flexmodule stehen in einer normalen und in einer kompakten Version zur Verfügung. Die normale Version beinhaltet alle Funktionen, während die kompakte Version nur Grundfunktionen zur Verfügung stellt, wodurch die Möglichkeiten für den Benutzer eingeschränkt werden.

Der Einbau eines Flexmoduls in ein Template ist denkbar einfach. Geben Sie folgenden Code ein, um ein Flexmodul im Template anzeigen zu lassen:

```
<!--INSERT_FLEXMODULE-->
<!--INSERT_FLEXMODULE_COMPACT-->
```

Beide Aufrufe erzeugen eine Flexsteuerung, über die Flexmodule ausgewählt und angezeigt werden können. Weitere Informationen finden Sie in Kapitel 4 **Flexmodule und Slots**.

3.3.17 Imperiablocks

Imperiablocks werden eingehend in Abschnitt 3.6 **Imperiablocks** auf Seite 77 behandelt. .

Mit Hilfe von Imperiablocks kann der Benutzer die Teile des Templates, die zwischen dem öffnenden und dem schließenden IMPERIABLOCK-Tag stehen, beliebig vervielfachen.

Auch hier ist der Einbau in ein Template denkbar einfach. Setzen Sie den öffnenden Tag vor den Code-Abschnitt des Templates, der duplizierbar sein soll, und den schließenden dahinter:

```
<!--IMPERIABLOCK-->
  Template-Code
<!--/IMPERIABLOCK-->
```

Zwischen diesen Tags kann beliebiger Template-Code stehen. Die einzige Ausnahme bilden Imperiablocks selbst, da diese nicht schachtelbar sind.

Für ausführliche Informationen zu Imperiablocks lesen Sie bitte Abschnitt 3.6 **Imperiablocks** auf Seite 77. Die Bedienung in einem Template finden Sie im Kapitel **Imperiablocks** im Userhandbuch

3.3.18 Codeinclude

Um den Quellcode von Templates schlank und pflegefreundlich zu halten, können Sie Codeabschnitte in externe Dateien auslagern. Diese lassen sich dann an bestimmten Stellen ins Template einfügen. Solche Codeincludes haben zusätzlich den Vorteil, dass Sie diese in mehreren Templates verwenden können. Imperia erwartet Codeincludes im Verzeichnis `/site/include`. Dieses Verzeichnis legt das Installationsprogramm von Imperia aber nicht automatisch an. Sie müssen es also auf dem Develop-System von Hand erstellen. Unterhalb dieses Verzeichnisses können Sie durch weitere Verzeichnisse Ihre Include-Dateien strukturieren. Die Syntax für die Einbindung externer Dateien mit Codebestandteilen ist wie folgt:

<!--CODEINCLUDE:*Verzeichnis*/*Dateiname*-->

Anstelle des Platzhalters *Dateiname* tragen Sie den Pfad zur Include-Datei ausgehend vom Verzeichnis `/site/include` und/oder den Dateinamen der einzufügenden Datei ein.

Beachten Sie, dass der Templateprozessor von Imperia mit obenstehender Syntax eingebundene Codeincludes vor Imperiablocks und Flexmodulen expandiert. Sie können also in diesen Elementen keine Codeincludes mit dieser Syntax einbinden. Um Codeincludes innerhalb von Flexmodulen und Imperiablocks zu nutzen, verwenden Sie folgende Syntax:

```
<!--CINCL:Verzeichnis/Dateiname-->
```

Hinweis:

Imperia stellt für die Verwaltung von Codeincludes in der Benutzerfläche ein Interface bereit. Lesen Sie hierzu den Abschnitt **Codeinclude-Dateien** *im Kapitel* **Struktur** *des Administrationshandbuchs.*

Einzufügende Dateien können HTML- oder Perl-Code enthalten. Dateien mit HTML-Code enden auf dem Suffix `.htms`, Dateien mit Perl-Code enden auf dem Suffix `.perl` bzw. `.pl`. Eingefügten Perl-Code führt der Templateprozessor aus. Hierfür stehen folgende spezielle Variablen zur Verfügung:

`$mode`	Liefert den Dokumenten-Modus.
`$metainfo`	Liefert ein Meta: Info-Objekt. Weitere Informationen hierzu finden Sie im Perldoc-Format in der Datei `/site/modules/core/Meta/Info.pm`.
`$userconf`	Liefert die Werte des angemeldeten Benutzers.
`$index`	Liefert den Wert des Parameters `INDEX`, wenn dieser gesetzt wurde, bzw. `undef`, wenn der Parameter nicht gesetzt wurde (siehe unten).
`$suffix`	Liefert den Anhang, der von Imperia an eine Meta-Variable angehängt wurde, beispielsweise bei Flexmodulen und Imperiablocks. `$suffix` enthält einen leeren String, wenn kein Suffix angehängt wurde.
`$new`	Enthält das Ergebnis des Perl-Skripts, beispielsweise den resultierenden HTML-Code.
`@parameters`	Enthält eine Liste mit den Parametern, die Sie im Template beim Aufruf des Codeincludes übergeben haben (siehe auch Abschnitt 3.3.18.1 **Code-Include-Parameter** `PARAMETERS` auf Seite 52).

Tabelle 3.1. Perl-Variablen für Code-Includes

Innerhalb der Codeinclude-Syntax können Parameter angegeben werden, die untereinander durch Doppelpunkte getrennt werden. Es stehen folgende Parameter zur Verfügung:

3.3.18.1 Code-Include-Parameter PARAMETERS

Dieser Parameter enthält eine Liste von Parametern, die durch Schrägstriche (/) voneinander getrennt werden. Parameter aus dieser Liste können in der eingefügten Datei ausgelesen und verwendet werden. Übergebene Strings werden mit Hilfe des Prozent-Zeichens maskiert (siehe RFC1630).

Beispiel:

Mit folgender Zeile wird eine Datei in ein Dokument eingebunden und es werden zwei Parameter übergeben:

```
<!--CODEINCLUDE:navi/top.htms:PARAMETERS=eins/zwei-->
```

In der eingebundenen Datei kann auf die übergebenen Parameter wie folgt zugegriffen werden:

```
<!--CI_PARAM1-->
<!--CI_PARAM2-->
```

3.3.18.2 Code-Include-Parameter INDEX

Dieser Parameter bestimmt den Index eines Code-Includes. Die Funktion verhält sich analog zu den Indizes bei Flexmodulen. Dieser Index wird automatisch an den Namen aller Meta-Variablen angehängt.

Der Index kann wie folgt referenziert werden:

```
<!--CI_INDEX-->
```

3.3.18.3 Perl-Codeincludes und das Pragma Strict

Natürlich können Sie auch in einem Codeinclude das Pragma **strict** verwenden, um die Einhaltung der damit verbundenen Konventionen zu erzwingen. Damit Sie dies nicht in jeder einzelnen Include einbinden müssen, können Sie die Konfigurationsvariable `USE_STRICT` einsezten. Setzten Sie diese in der Datei `site/config/system.conf` auf *1*, um das Pragma global für alle Codeincludes einzubinden.

3.3.19 Standardfarben für die Änderungsverfolgung festlegen

Die Änderung der Standardfarben der Änderungsverfolgung nehmen Sie im Template mit Hilfe von Meta-Variablen vor. Die Namen der Meta-Variablen ergeben sich aus den Namen der CSS-Formate.

Hinweis:

Weitere Informationen zur Änderungsverfolgung finden Sie im Kapitel **Änderungsverfolgung** *im Userhandbuch und in Kapitel* **Änderungsverfolgung** *im Systemhandbuch.*

Enthält ein Dokument eine Meta-Variable mit dem Namen `imperiaMinusSpan` und dem Wert `color: #347F33; background: #235453`, werden diese Werte zur Darstellung von gelöschtem Text verwendet. Für CSS-Formate, die Sie nicht mit Hilfe von Meta-Variablen überschrieben haben, gilt weiterhin das Standard-Stylesheet der Änderungsverfolgung.

Beispiel:

```
<input name="IMPERIA:imperiaPlusSpan" type="hidden" value="color:
#DEADDE; background: red" />
```

Achtung:

Die verschachtelten CSS-Formate (zum Beispiel `imperiaPlusSpan td`*) in der Datei* `diff.css` *können Sie nicht mit Hilfe von Meta-Variablen überschreiben.*

3.4 Funktionen und Konstanten in Templates

Im Folgenden finden Sie Erläuterungen zu den in Imperia verfügbaren Steuerungselementen und Variablen für Templates.

3.4.1 Elemente ausblenden

Bestimmte Elemente eines Templates, insbesondere beschreibender Hinweistext im Template, sind nur für die Anzeige während der Bearbeitung eines Dokuments bestimmt. In der Vorschau oder der endgültigen Version des Dokuments hingegen sollen diese Elemente nicht erscheinen. Dies erreichen Sie, indem Sie diese Elemente abhängig vom Dokumenten-Modus ausblenden lassen.

In Imperia existieren vier verschiedene Dokumenten-Modi, die Sie mit Hilfe einer IF-Abfrage überprüfen können (siehe auch Abschnitt 3.2 **Template-Grundlagen** auf Seite 28 und Abschnitt 3.4.5 **Modus abfragen und überprüfen** auf Seite 55).

Beispiel:

```
Artikeltext eingeben:<br />
  <textarea name="IMPERIA:textfeld2" rows="15" cols="40">
 </textarea><br />
```

Bei diesem Beispiel erscheint die Zeile "`Artikeltext eingeben:`" auch im fertigen Dokument. Da es sich um eine Anweisung für den Benutzer handelt, der das Dokument bearbeitet, ist das aber nicht erwünscht.

Verhindern können Sie dies, indem Sie den Erläuterungstext für die Bearbeitung mit Hilfe einer #IF-Abfrage nur im entsprechenden Modus anzeigen lassen:

```
#IF ("<!--XX-editmode-->")
  Artikeltext eingeben:<br />
#ENDIF
  <textarea name="IMPERIA:textfeld2" rows="15" cols="40">
  </textarea><br />
```

So erscheint die fragliche Zeile nur, wenn sich das Dokument im EDIT-Modus befindet. Um andere Bedingungen zu bilden, stehen alle in #IF-Abfragen erlaubten Operatoren zur Verfügung (siehe Abschnitt 3.4.15 **IF-Abfragen** auf Seite 67).

Dabei können Sie wahlweise die Elemente in bedingten Anweisungen nach den Modi gruppieren, in denen sie erscheinen sollen, oder für jedes jeweils eine gesonderte #IF-Abfrage erstellen.

3.4.2 Mit Meta-Variablen arbeiten

Wie am Anfang von Abschnitt 3.3 **Syntax-Referenz** auf Seite 29 beschrieben, speichert Imperia den Inhalt eines mit `IMPERIA:feldname` benannten Formularfelds im Template in einer entsprechend benannten Meta-Variablen. Sind mehrere Formularfelder in einem Template identisch benannt, speichert Imperia die eingegebenen Werte dieser Felder als Elemente eines Arrays in einer einzelnen entsprechend benannten Meta-Variablen. Wenn das nicht das erwünschte Verhalten ist, müssen Sie also die Eindeutigkeit der Formularfeldnamen sicherstellen.

3.4.3 Meta-Variablen referenzieren

Innerhalb eines Templates können Sie auf die Werte, die in benannten Meta-Variablen gespeichert wurden, zugreifen, um diese Werte an anderer Stelle anzeigen zu lassen oder sie in #IF-Abfragen zu prüfen. Dazu dienen XX-Variablen. Diese XX-Variablen werden von Imperia automatisch durch den Inhalt der referenzierten Meta-Variablen ersetzt (siehe auch Abschnitt 6.16 **XX-Variablen und XXOBJ-Variablen** auf Seite 145).

Existiert eine referenzierte Meta-Variable nicht, wird die Referenz ohne Ersetzung aus dem fertigen Dokument entfernt.

Um mit Hilfe einer XX-Variablen auf eine Meta-Variable zuzugreifen, wird innerhalb eines HTML-Kommentars zunächst der Typ der Variablen, also XX, anschließend ein Minus-Zeichen gefolgt vom Namen der zu referenzierenden Meta-Variablen eingetragen.

Beispiel:

`<!--XX-title-->`
: Diese Referenz liefert den Titel des Dokuments.

`<!--XX-directory-->`
: Diese Referenz liefert das Verzeichnis, in dem Imperia das fertige Dokument abspeichert.

`<!--XX-text1-->`
: Diese Referenz liefert den Inhalt der Meta-Variablen `text1`, so dass dieser Inhalt an Stelle der Referenz im fertigen Dokument erscheint.

<!--XXOBJ-myimage-->
: Diese Referenz liefert den Dateipfad eines Objekts aus der Media-Asset-Management, bzw. eines Objekts, das Sie mit dem Grafiker-Upload in das Dokument eingebunden haben.

Eine Besonderheit stellt hierbei die Referenzierung von Dateien dar, die Sie in die Media-Asset-Management oder per Grafiker-Upload hochgeladen haben. Diese von Imperia verwalteten Objekte referenzieren Sie mit dem eigenen Variablentyp `XXOBJ`. So erkennt Imperia, dass es sich um eine Referenz auf eine Datei handelt und kann an der entsprechenden Stelle den Pfad zu dieser Datei einsetzen (siehe auch Kapitel 6 **Variablen**).

Neben XX-Variablen können auch CC-Variable zur Referenzierung verwendet werden. Bei einer Referenz durch eine CC-Variablen wird der Inhalt der referenzierten Meta-Variablen URL-maskiert. Beispielsweise werden Leerzeichen zu `%20` (siehe Kapitel 6 **Variablen**).

Außerdem können Sie für die Ausgabe einer Meta-Variablen gezielt einen Escaping-Modus erzwingen, auch wenn dieser von dem der Eingabe abweichen sollte. Mit folgender Syntax erzwingen Sie einen Escaping-Modus für die Ausgabe einer Meta-Variablen:

```
<!--XX-MODUS:Metafeldname-->
```

Um also beispielsweise die Interpretation von zuvor als Text eingegebenen HTML-Tags nachträglich zu erzwingen, notieren Sie diese Anweisung:

```
<!--XX-HTML:myMetafield-->
```

Für Informationen zu Escaping-Modi bei Eingabefeldern und den verfügbaren Modi lesen Sie Abschnitt 3.3.2 **HTML-Escaping-Modi** auf Seite 30 und Abschnitt 6.21 **Escaping Modes** auf Seite 149.

3.4.4 Meta-Variablen prüfen

In manchen Fällen ist es sinnvoll zu prüfen, ob eine Meta-Variable leer ist, so dass beispielsweise in Abhängigkeit davon bestimmte Aktionen ausgelöst werden können. Um zu prüfen, ob eine Meta-Variable gefüllt ist, benutzen Sie folgende Syntax:

<!--XXDEFINED-*Meta-Variable*-->

oder die Kurzform:

<!--XXDEF-*Meta-Variable*-->

Ist die überprüfte Meta-Variable nicht leer, liefern beide Formen `1` zurück, ansonsten `0`. Lesen Sie hierzu auch Abschnitt 6.17 **XXDEFINED-Variablen** auf Seite 146.

3.4.5 Modus abfragen und überprüfen

Ein Dokument kann sich in verschiedenen Modi befinden. Über Variablen kann dieser Modus abgefragt und überprüft werden. Dies ist insbesondere dann wichtig, wenn die Ausgabe in Abhängigkeit vom Dokumenten-Modus erfolgen soll, beispielsweise dann, wenn erläuternde Hilfetexte in der Vorschau ausgeblendet werden sollen.

Um den Dokumenten-Modus im Template abzufragen, ist folgender Code nötig:

```
<!--XX-mode-->
```

Diese Variable liefert den aktuellen Dokumenten-Modus. Befindet sich das Dokument im `EDIT`-Modus, enthält die Variable den Wert `EDIT`. Befindet sich das Dokument in einer Vorschau, enthält die Variable den Wert `PREVIEW`.

Um den Dokumenten-Modus zu überprüfen, werden verschiedene Variablen verwendet. Befindet sich das Template in dem der Variablen entsprechenden Modus, enthält sie den Wert `1` (true), ansonsten enthält sie den Wert `0` (false).

Einzige Ausnahme bildet `<!--XX-savemode-->`. Diese Variable enthält den Wert `1`, wenn sich das Dokument im SAVE-, PREVIEW- oder im REPARSE-Modus befindet.

Um die einzelnen Dokumenten-Modi zu überprüfen, wird folgende Syntax verwendet:

Variable	<!--mode-->
<!--XX-editmode-->	EDIT
<!--XX-savemode-->	SAVE, REPARSE, PREVIEW
<!--XX-reparsemode-->	REPARSE
<!--XX-previewmode-->	PREVIEW

Tabelle 3.2. Syntax zur Prüfung der Dokumenten-Modi

3.4.6 Rollenzugehörigkeit prüfen

Sie können feststellen, ob der Benutzer, der gerade ein Dokument bearbeitet, einer bestimmten Rolle angehört. Das Ergebnis dieser Überprüfung können Sie beispielsweise als Bedingung für die Anzeige bestimmter Bestandteile eines Templates verwenden. So erreichen Sie, dass nur Mitglieder einer bestimmten Rolle diese Elemente sehen, wenn sie das Dokument bearbeiten. Um die Zugehörigkeit eines Benutzers zu einer Rolle zu prüfen nutzen Sie folgende Syntax:

```
<!--MEMBER_OF:[RollenID | Rollenname]-->
```

Sie können wahlweise den Namen oder die ID der fraglichen Rolle für die Überprüfung notieren.

3.4.7 Daten eines Benutzers auslesen

Diese Funktion erlaubt es, Daten eines Benutzers aus der Benutzer-Verwaltung auszulesen. Dadurch können Sie zum Beispiel den Namen oder die Email-Adresse des Autors eines Artikels automatisch in das Dokument einfügen.

Um im Template auf Benutzerdaten zuzugreifen, wird folgende Syntax verwendet:

```
<!--USER_CONF:Feld>
```

Anstelle des Platzhalters Feld setzen Sie einen der folgenden Feldnamen ein:

Name	Inhalt
homepage	Startseite
country	Land
telnumber	Telefonnummer
language	Sprache
cellular	Mobiltelefon-Nummer
name	Name
fname	Vorname
city	Stadt
faxnumber	Faxnummer
comment	Kommentar
zip	PLZ
login	Anmeldekennung/Login
email	Email-Adresse
street	Straße

Tabelle 3.3. Felder der Benutzerverwaltung

Achtung:

Die alte Syntax mit Bindestrich (USER-CONF) wird nicht mehr unterstützt!

3.4.8 Parameter der System-Konfiguration auslesen

Diese Funktion erlaubt es, Konfigurationsdaten des Systems auszulesen. Es stehen alle Parameter zur Verfügung, die in der Datei `/site/config/system.conf` definiert werden.

Um im Template auf Parameter der System-Konfiguration zuzugreifen, wird folgende Syntax verwendet:

```
<!--SYSTEM_CONF:Parameter-->
```

Anstelle des Platzhalters *Parameter* muss der Name der entsprechenden system.conf-Variable eingesetzt werden. Um zum Beispiel die eingestellte Standard-Sprache zu erhalten, geben Sie folgenden Code ein:

```
<!--SYSTEM_CONF:LANGUAGE-->
```

Beachten Sie die genaue Schreibweise der System-Parameter (Groß- und Kleinschreibung, Unterstriche oder Bindestriche).

Eine Liste aller Konfigurationsparameter finden Sie in Kapitel **Konfiguration von Imperia** im Administrationshandbuch.

3.4.9 Dokument-ID auslesen

Die Dokument-ID ist Bestandteil der Node-ID, die wiederum jedes Dokument innerhalb von Imperia eindeutig macht. Eine Node-ID besteht aus zwei Teilen:

- der Rubrik-ID, zum Beispiel `/12/24/35/`
- der Dokument-ID, zum Beispiel `130336`

Die vollständige Node-ID aus diesen beiden Teilen ist `/12/24/35/130336`. Sowohl die Dokument-ID als auch jeder einzelne Abschnitt der Rubrik-ID sind garantiert eindeutig, da Imperia diese aus einem gemeinsamen, automatisch inkrementierten Counter generiert.

Verwenden Sie folgende Syntax, um die Dokument-ID des aktuellen Dokuments im Template auszulesen:

```
<!--DOC_ID:Format-->
```

Im Gegensatz zu früheren Imperia-Versionen können Sie nun komplexere Parameter angeben. Parameter sind optional und im `printf`-Format zu notieren. Ein leerer String oder der String `0` gelten nicht als Format-Angabe. Hier findet eine Ersetzung durch `%s` statt.

Hinweis:

Das Format `0` aus früheren Imperia-Versionen können Sie weiterhin verwenden. Es liefert die Dokument-ID, also den letzten Teil der Node-ID. Dies wird nun durch `%s` erreicht.

Wenn Sie beispielsweise einen Verzeichniszähler benötigen, wie Sie ihn in Metadateien mit Hilfe der Funktion `<!--count-->` generieren, verwenden Sie folgende Syntax:

```
<!--DOC_ID:%05d-->
```

Numerische Dokument-IDs mit weniger als fünf Stellen füllen Sie so nach links mit Nullen auf.

Ein weiteres Beispiel:

```
<!--DOC_ID:Die Dokumenten-ID dieses Dokuments: %d-->
```

3.4.10 Berechnungen im Template

Im Template stehen die vier Grundrechenarten sowie der Gleichheitsoperator zur Verfügung, um zwei Werte zu manipulieren. Der Inhalt von Metavariablen kann folgendermaßen in eine Rechenvariable überführt werden:

```
<!--TCOUNTER:tcountervar={Metafeldname}-->
<!--TCOUNTER:uid={__imperia_uid}-->
```

Die Syntax für Berechnungen im Template ist wie folgt:

<!--TCOUNTER:*Name*=*Wert*-->
<!--TCOUNTER:*Name*+*Wert*-->
<!--TCOUNTER:*Name*-*Wert*-->
<!--TCOUNTER:*Name*/*Wert*-->
<!--TCOUNTER:*Name**Wert*-->

In der ersten Zeile wird der Zuweisungsoperator = verwendet, um eine neue Rechen-Variable zu erzeugen, die zum Rechnen verwendet werden kann.

Anstelle des Platzhalters *Name* muss der Name einer Rechen-Variablen angegeben werden.

Anstelle des Platzhalters *Wert* kann entweder ein numerischer Wert oder der Name einer weiteren Rechen-Variablen angegeben werden.

Variablen, die nicht existieren, werden automatisch erzeugt und erhalten den Wert `0`. Bei einer Division durch `0` liefert die Berechnung einen leeren String.

Beispiel für einfache Berechnungen:

```
<!--TCOUNTER:var1=4-->
<!--TCOUNTER:var2=2-->
Ergebnis:<br />
<!--TCOUNTER:var1+var2-->
```

Berechnung mit Hilfe einer Konstanten:

```
<!--TCOUNTER:var1=4-->
<!--TCOUNTER:var1+dirlevel:0-->
```

Das Ergebnis der Berechnung wird im Dokument angezeigt.

3.4.11 Rubrik-Eigenschaften auslesen

Über das Template können Sie verschiedene Rubrik-Eigenschaften der Rubrik des jeweiligen Dokuments auslesen und in das Dokument eintragen.

Rubrik-Eigenschaften lesen Sie mit folgender Syntax aus:

<!--SECTION:*Schlüsselwort*:*Rubriklevel*-->

Anstelle des Platzhalters `Schlüsselwort` stehen folgende Schlüsselwörter zur Verfügung:

Schlüsselwort	Beschreibung
NAME	Liefert den Namen der Rubrik.
DESCR	Liefert die Beschreibung der Rubrik, die in der Rubriken-Verwaltung eingegeben wurde.
DIRECTORY	Liefert das Verzeichnis, das in der Rubriken-Verwaltung angegeben wurde.
TEMPLATE	Liefert den Namen des Templates, das in der Rubrik verwendet wird.
FILENAME	Liefert den Dateinamen, der in der Rubrik verwendet wird.
META_INFO_XY	Liefert den Wert der Metavariablen XY, die in den Rubrikeinstellungen gespeichert ist.

Tabelle 3.4. Liste der Schlüsselwörter

Wir empfehlen, nur die Schlüsselwörter `DESCR` und `NAME` zu verwenden, da alle anderen möglicherweise nicht den erwarteten Wert enthalten. Der initiale Wert aus der Rubrik kann sich an anderer Stelle im Workflow ändern.

Dazu ein Beispiel: Die Hauptrubrik `movies` enthält eine Unterrubrik `action`. Darin befindet sich die Unterrubrik `martial arts`. Um nun den Namen der Unterrubrik `action` zu erhalten, müssen Sie folgenden Code eingeben:

```
<!--SECTION:NAME:2-->
```

Wollen Sie nur den untersten Rubriklevel abfragen, brauchen Sie den Rubriklevel nicht anzugeben. Folgender Code liefert den Namen der untersten Rubrik:

```
<!--SECTION:NAME-->
```

Verwalten Sie z.B. den Namen des Autors zu jeder Rubrik in den zusätzlichen Metainformationen der Rubrik, können Sie diese Information mit folgendem Ausdruck auswerten:

```
<!--SECTION:META_INFO_autor-->
```

Das Präfix *META_INFO_* ist hierbei optional und dient der besseren Unterscheidung von anderen Metafeldern. Sollten Sie allerdings in den Rubrikeinstellungen Metafeldnamen verwenden, die auch in den Meta-Informationen eines Dokuments vorkommen, ist das Präfix bindend. Lesen Sie hierzu auch Abschnitt 6.7.1 **Zugriff auf Metainformationen von Rubriken** auf Seite 141.

3.4.12 Pfad-Bestandteile auslesen

Im Zusammenhang mit `<!--XX-directory-->` gibt es noch einige Besonderheiten. Diese Variable enthält, wie oben erwähnt, das Verzeichnis, in dem das Dokument abgelegt wird.

Es besteht auch die Möglichkeit, nur auf einzelne Teile dieses Pfades zuzugreifen. Hierzu wird die Variable <!-- dirlevel:*xxx*--> verwendet, wobei `xxx` für die Position des gewünschten Verzeichnisses im Pfad steht, beginnend mit dem obersten Verzeichnis. Das oberste Verzeichnis erhält die Nummer 1.

Angenommen, ein Dokument liegt im Verzeichnis `/sport/juni/16`. Dann würde folgender Code

```
Dieser Artikel ist vom <!--dirlevel:3-->. <!--dirlevel:2-->.
```

im fertigen Dokument zu

```
Dieser Artikel ist vom 16. Juni.
```

3.4.13 Mehrsprachige Inhalte eingeben

Wenn Sie mehrsprachige Inhalte mit dem Imperia-Multilanguage-Feature publizieren, können Sie die Inhalte für die einzelnen Sprachversionen getrennt erfassen, indem Sie die betreffenden Eingabefelder in die folgenden Anweisungen einschließen:

```
<!--multilang_start-->

[Template-Code für mehrsprachige Eingabe]

<!--multilang_end-->
```

Wenn Sie in einem Template Inhaltsbereiche für die mehrsprachige Eingabe definiert haben, blendet Imperia im Edit-Modus des Dokuments für jede verfügbare Sprachversion am Seitenkopf ein Icon mit der Landesflagge ein. Klicken Sie auf eines dieser Icons, um den Inhalt für die betreffende Sprachversion einzugeben.

Wichtig:

Beachten Sie, dass Sie bei Flexmodulen feste Indizes vergeben müssen. Lesen Sie hierzu auch Abschnitt 4.3.6 **INDEX=***xxx auf Seite 106.*

Eingabefelder für mehrsprachige Inhalte müssen Sie mit einem entsprechenden Suffix versehen, um deren Eindeutigkeit im Template zu gewährleisten. Hängen Sie hierzu das Suffix *_lingua* an den Variablennamen an:

```
<input name="IMPERIA:myfield_lingua" type="text" />
```

Mit dem Kommentar *<!--XX-lingua-->*können Sie die gegenwärtige Sprache abfragen bzw. ausgeben:

```
#IF ("<!--XX-lingua-->" EQ "DE")

[do something]

#ENDIF
```

In beiden Fällen nimmt der Templateprozessor eine Ersetzung durch das Kürzel der gegenwärtigen Sprache des jeweiligen Abschnitts vor.

Hinweis:

Bei Eingabefeldern in Flexmodulen ist dies nicht notwendig, da diese bereits durch die automatisch vergebenen und angehängten Indizes eindeutig sind (siehe Abschnitt 4.6 **Variablen in Flexmodulen** *auf Seite 110 und Abschnitt 4.7* **Inhalt außerhalb eines Flexmoduls verwenden** *auf Seite 111).*

3.4.14 Uhrzeit und Datum einfügen

Mit Imperia lassen sich Datum und Uhrzeit in verschiedenen Formaten in Dokumenten einbinden. Dabei haben Sie die Wahl zwischen automatisch vom System errechneten oder manuell vom Benutzer einzugebenden Datums- und Zeitangaben.

3.4.14.1 Datumseingaben mit dem Kalender-Tool

Mit dem Kalender-Tool generieren Sie mit wenigen Mausklicks Datumseingaben in Ihren Dokumenten. Die Einbindung ins Template nehmen Sie über eine Processing-Instruction vor. Weitere Einstellungen des Kalender-Tools legen Sie über Parameter für die Processing-Instruction fest. Die Syntax lautet folgendermaßen:

```
<?imperia calendar
  parameter: wert
  parameter: wert
?>
```

Der Platzhalter `parameter: wert` steht hierbei für die Einstellungen des Kalenders. Diese notieren Sie immer als durch einen Doppelpunkt getrenntes Name-Wert-Paar, wobei für einige Parameter auch mehrere Werte möglich sind. Die einzelnen Parameter müssen jeweils in einer eigenen Zeile stehen. Folgende Parameter stehen zur Verfügung:

Parameter popup

Setzen Sie diesen Parameter, damit der Kalender sich in einem Popup-Fenster öffnet. Wenn der Parameter fehlt, erscheint der Kalender direkt im Bearbeiten-Schritt des Dokuments. Die Syntax lautet wie folgt:

```
popup: ELEMENT EVENT [SKIN]
```

Folgende Werte sind für diesen Parameter verfügbar:

ELEMENT

Dieser Wert gibt das Element an, über das der Benutzer Kalender-Tool aufruft. Der Wert *default* fügt im Bearbeitungsmodus automatisch ein kleines Kalender-Icon ein  . Wenn Sie stattdessen ein anderes HTML-Element verwenden möchten, beispielsweise ein anderes Icon, müssen Sie es mit einem ID-Attribut versehen und diese ID als Wert für `ELEMENT` notieren. Dazu ein Beispiel:

```
<img src="/img/icon/my_calendar_icon.png" id="c_Icon" width="64" height="64" />

<?imperia calendar
  popup: c_Icon click
  [weitere Parameter]
?>
```

In diesem Beispiel öffnet ein Klick auf die Grafik `my_calendar_icon.png` das Kalender-Tool. Die Verbindung zwischen Grafik und Processing-Instruction kommt dabei durch die ID "**c_Icon**" zustande.

Während der Templateprozessor das Standard-Icon zum Aufruf des Kalender-Tools automatisch im Template ergänzt, müssen Sie ein individuelles Element, z. B. ein eigenes Icon wie im obigen Beispiel selbst ins Template einfügen

EVENT

Geben Sie ein JavaScript-Event an, auf das hin sich das Kalender-Tool öffnen soll, beispielsweise *click*. Das betreffende Event müssen Sie komplett in Kleinbuchstaben notieren.

SKIN

Mit diesem optionalen Parameter bestimmen Sie das Aussehen des Kalender-Tools. Sie können wahlweise eine absolute Pfadangabe ausgehend von der Document-Root Ihres Systems notieren oder den Namen eines der folgenden Standard-Skins:

`aqua`, *blue*, *blue2*, *brown*, *default*, *green*, *system*, *tas*, *win2k-1*, *win2k-2*, *cwin2k-cold-1*, *win2k-cold-2*. Bitte beachten Sie hierbei die Groß- und Kleinschreibung.

Diese Kalender-Skins bestehen jeweils aus einer Sammlung von Grafik-Dateien und einer CSS-Datei. Diese liegen in einem eigenen Verzeichnis unter `htdocs/imperia/skin/widgets/calendar`.

Parameter connectInputId

Mit diesem Parameter definieren Sie, in welchem Element des Templates das Kalender-Tool die Datumsangabe speichern soll. Außerdem legen Sie das Format der erzeugten Datumsangabe fest. Das Element zur Speicherung muss ein Imperia-Eingabefeld sein. Damit das Kalender-Tool auf dieses Eingabefeld zugreifen kann, müssen Sie es mit einem ID-Attribut versehen.

Syntax:

```
connectInputId: Element-ID "Format"
```

Wichtig:

Wenn Sie das Kalender-Tool in einem Imperiablock, Arrayblock, Slot- oder Flexmodul einsetzen möchten, müssen Sie die ID des Eingabefelds um Indizes und Instanz-IDs des Imperiablocks, Arrayblocks, Slot- oder Flexmoduls ergänzen, da der Templateprozessor dieses Attribut nicht automatisch anpasst. In einem Flexmodul sieht das beispielsweise folgendermaßen aus:

```
...

<?imperia calendar
  popup: default click
  connectInputId: datumsfeld_<!--FLEX_INDEX-->_<!--FLEX_ID-->
  [weitere Parameter]
?>

<input name="IMPERIA:mydate" id="datumsfeld_<!--FLEX_INDEX-->_<!--FLEX_ID-->" type="

...
```

Den Wert für die Festlegung des Datumsformats notieren Sie als in Anführungszeichen eingeschlossenen String. Dieser String besteht aus einer oder mehreren Formatierungsangaben und beliebigem Text. Wenn Sie keine Angaben zum Format des Datumseintrags machen, erscheint dieser als Unix-Zeitstempel. Die verfügbaren Formatierunsangaben sind in der folgenden Tabelle aufgeführt:

Formatierungsangabe	Beschreibung
%a	abgekürzter Wochentagsname
%A	vollständiger Wochentagsname
%b	abgekürzter Monatsname
%B	vollständiger Monatsname
%C	Nummer des Jahrhunderts
%d	zweistelliger Tag des Monats (00 - 31)
%e	Tag des Monats (0 - 31)
%H	Stunde (00 - 23)
%I	Stunde (01 - 12)
%j	Tag des Jahres (000 .. 366)
%k	Stunde (0 - 23)
%l	Stunde (1 - 12)
%m	Monat (01 - 12)
%M	Minute (00 - 59)
%n	Zeilenumbruch
%p	``PM'' oder ``AM''
%P	``pm'' oder ``am''
%S	Sekunde (00 - 59)
%s	Unix-Zeitstempel
%t	Tabulatorzeichen
%U, %W, %V	Wochennummer
%u	Nummer des Wochentags (1 - 7, 1 = MON)
%w	Nummer des Wochentags (0 - 6, 0 = SUN)
%y	Jahreszahl ohne Jahrhundert (00 - 99)
%Y	vollständige Jahreszahl (z. B. 1979)
%%	Prozentzeichen ('%')

Tabelle 3.5. Datums-Formatangaben

Dazu ein Beispiel:

```
...
connectInputId: datumsfeld "Ausgewähltes Datum:%n %A, der %d.%m.%Y, %H:%M:%S%n.
Zu diesem Zeitpunkt liegt der 01.01.1970 %s Sekunden zurück."
...
```

Nach Auswahl eines Datums steht dann beispielsweise folgender Text in dem betreffenden Metafeld:

ausgewähltes Datum: Samstag, der 20.10.2007, 23:42:05. Zu diesem Zeitpunkt liegt der 01.01.1970 1192916525 Sekunden zurück.

Hinweis:

In obigem Beispiel ist die Formatierungsanweisung aus Darstellungsgründen umbrochen. Im Template muss sie in einer Zeile stehen.

Wenn Sie die Textbestandteile der Formatierungsanweisung später mehrsprachig ausgeben lassen wollen, schließen Sie diese in doppelte eckige Klammern ein. Die Übersetzungen müssen Sie in Form von po-Dateien in Imperia einbinden.

Parameter showWeekNumbers

Legen Sie fest, ob das Kalender-Tool die Nummer der Kalenderwoche anzeigt oder nicht.

Syntax:

```
showWeekNumbers: true | false
```

Wenn Sie diesen Parameter weglassen, erscheinen keine Wochennummern (Default).

Parameter showTime

Mit diesem Parameter blenden Sie die Anzeige der Uhrzeit im Kalender-Tool ein oder aus. Nur bei eingeblendeter Uhrzeit kann der Benutzer auch die Zeitangabe des Datumseintrags verstellen. Ohne diesen Parameter ist die Uhrzeit ausgeblendet.

Syntax:

```
showTime: true | false
```

Parameter setFirstDayOfWeek

Nutzen Sie diesen Parameter, um in der Anzeige des Kalender-Tools einen anderen Tag als Montag als ersten Wochentag zu definieren. Die Angabe erfolgt numerisch, wobei der erste Tag, Montag, den Wert 1 hat.

Syntax:

```
setFirstDayOfWeek: 4
```

Bei dieser Einstellung erscheint Donnerstag im Kalender-Tool als erster Wochentag.

Parameter showAdjacentMonths

Mit diesem Parameter aktivieren Sie die Anzeige von Tagen der angrenzenden Monate im Kalender-Tool. Die Default-Einstellung ist *false* (keine Anzeige der Tage aus angrenzenden Monaten).

Syntax:

```
showAdjacentMonths: true | false
```

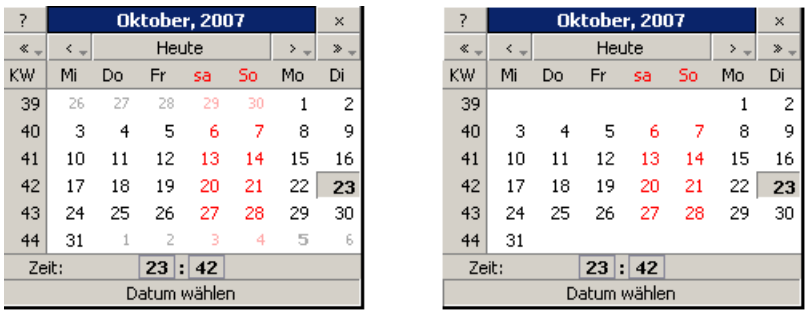

Abb. 3.5: Tage aus angrenzenden Monaten im Kalender-Tool ein- und ausgeblendet

Parameter startDate

Verwenden Sie diesen Parameter, um das Kalender-Tool beim Aufruf mit einem Datumseintrag vorzubelegen. Die Angabe erfolgt als Unix-Zeitstempel. Ein Online-Tool zur Umrechnung ins Unix-Zeitstempel-Format inklusive Zeitzonen-Versatz finden Sie beispielsweise unter der URL `http://www.theblog.ca/?p=148`.

Beispiel:

```
startDate: 1193175725
```

Der Unix-Zeitstempel aus diesem Beispiel entspricht dem Datum 23.10.2007, 23:42:05 Uhr MEZ.

Parameter startYear

Geben Sie eine Jahreszahl an, mit der die Jahres-Skala des Kalendertools beginnen soll. Weiter zurückliegende Jahre lassen sich dann vom Benutzer nicht einstellen.

Parameter endtYear

Geben Sie eine Jahreszahl an, mit der die Jahres-Skala des Kalendertools enden soll. Weiter in der Zukunft liegende Jahre lassen sich dann vom Benutzer nicht einstellen.

34.142 Datum einfügen

Das aktuelle Datum kann in verschiedenen Formaten berechnet und angezeigt werden. Um das Datum in das Template einzufügen, wird folgende Variable verwendet:

```
<!--Xdate:Format-->
```

Für den Platzhalter *Format* muss einer der folgenden Parameter eingefügt werden:

Formatname	angezeigtes Format
default	15.08.2001
full	15.08.2001
inverse	2001-08-15
afiles	2001-08-15 22:13
iso	20010815
normal	15.08.2001
american	15.08.2001
imperia	15.08.2001 22:03
finddate	2001.08.15 22:03

Tabelle 3.6. Datums-Parameter

Ein Datum kann auch in einem `input`-Feld als Default-Eintrag eingetragen werden:

```
<input name="IMPERIA:datum" type="text" size="8" value="<!--Xdate:default-->" />
```

34.143 Datums-Teile einfügen

Um auf einzelne Datumselemente, also nur auf den Tag, den Monat oder das Jahr zuzugreifen, wird folgende Syntax verwendet:

```
<!--DATE:Format-->
```

Für den Platzhalter *Format* muss einer der folgenden Parameter eingefügt werden:

Parameter	Beschreibung
day	Tag (numerisch)
month	Monat (Klartext)
mon	Monat (numerisch)
year	Jahr (numerisch)

Tabelle 3.7. Datums-Parameter

3.4.144 Uhrzeit einfügen

Um die Uhrzeit einzubinden, wird folgende Syntax verwendet:

```
<!--Xtime:Format-->
```

Für den Platzhalter *Format* muss einer der folgenden Parameter eingefügt werden:

Parameter	Beschreibung
default	22:14
normal	22:14:18
full	22:14:18
compact	22:14
hour	22
minute	14
second	18

Tabelle 3.8. Uhrzeit-Parameter

3.4.145 Uhrzeit und Datum lokalisiert

Um das aktuelle Datum und die aktuelle Zeit über die Systemfunktion `strftime` zu formatieren, wird folgende Syntax verwendet:

```
<!--Xstrftime:Locale:Format-->
```

Der Parameter `Locale` steht hier für die betriebssystemabhängige LOCALE-Variable. Lesen Sie hierzu in der Dokumentation Ihres Betriebssystems nach. (z.B. man LOCALE)

Beispiel:

```
<!--Xstrftime:de_DE: Heute ist %A, der %d.%B %Y.-->
```

Mit dieser Syntax erscheint im Dokument

```
Heute ist Donnerstag, der 16. August 2001.
```

Alle eventuellen HTML-Zeichen (kleiner als, größer als, Ampersand, doppelte und einfache Anführungszeichen) im String, der von `strftime` geliefert wird, werden korrekt escaped.

3.4.146 Lokalisiertes Datum mit Offset

Um ein Datum in der Zukunft oder der Vergangenheit zu erhalten, können Offsets angegeben werden. Die erforderliche Syntax ist wie folgt:

```
<!--Xstrftimeoff:Offset:Locale:Format-->
```

Der Parameter *Locale* wird durch die systemabhängige LOCALE-Variable ersetzt. Der Parameter *Format* wird entsprechend des gewünschten Datumsformats aufgebaut.

Offsets bestehen aus einer beliebigen Anzahl von Parametern der Form `Operator Wert Einheit`. Der Operator ist optional. Wird kein Operator angegeben, dann wird der Wert zum aktuellen Datum addiert. Folgende Operatoren stehen zur Verfügung:

Operator	Beschreibung
+	Addiert den folgenden Wert zum aktuellen Datum bzw. zur aktuellen Zeit.
-	Subtrahiert den folgenden Wert vom aktuellen Datum bzw. der aktuellen Zeit.

Tabelle 3.9. Offset-Operatoren

Folgende Werte stehen zur Verfügung:

Parameter	Beschreibung
s	Sekunden
m	Minuten (immer 60 Sekunden)
h	Stunden (immer 3600 Sekunden)
D	Tage (immer 86400 Sekunden)
W	Wochen (immer 604800 Sekunden, z.B. 7 Tage)
M	Monate (immer 2592000 Sekunden, z.B. 30 Tage)
Y	Jahre (immer 378432000 Sekunden, z.B. 365 Tage)

Tabelle 3.10. Offset-Parameter

Schaltsekunden und Schalttage werden ignoriert, genauso der Unterschied in der Anzahl der Tage eines Monats.

Beispiele:

Wert	Beschreibung
`+1W-1D`	Heute plus 1 Woche minus 1 Tag
`600`	jetzt plus 600 Sekunden
`+1M`	Heute plus 1 Monat

Tabelle 3.11. Lokalisierungsbeispiel

Der betreffende Algorithmus ist darauf optimiert, erwartete Ergebnisse zu berechnen. Wenn Sie am 13. eines Monats ein Offset von drei Monaten einstellen, ist das Ergebnis ebenfalls der 13. des entsprechenden Monats. Ein nicht existierendes Zieldatum, wie beispielsweise der 31. Februar, wird durch den ersten existierenden Tag vor dem Wunschdatum ersetzt.

3.4.15 IF-Abfragen

Mit IF-Abfragen überprüfen Sie in einem Template Bedingungen. Je nach Ergebnis der Überprüfung führt Imperia von Ihnen festgelegte Anweisungen aus. Zusätzlich besteht die Möglichkeit, auch Alternativbedingungen und das Nichtzutreffen einer Bedingung zu prüfen. Die Schachtelung von IF-Abfragen ist ebenfalls möglich (siehe auch Abschnitt 3.4.15.8 **Verschachtelte IF-Abfragen** auf Seite 71). Das Ergebnis ist entweder `wahr` (auch `true` oder `1`) oder `falsch` (auch `false` oder `0`).

Eine bedingte Anweisung beginnt mit dem Schlüsselwort IF, gefolgt von der Bedingung, die in runden Klammern steht und einer oder mehrerer Anweisungen, die beim Zutreffen der Bedingung auszuführen sind. Den Abschluss der Anweisung bildet das Schlüsselwort ENDIF. Optional können Sie einen Zweig mit einem alternativen Anweisungsblock einfügen. Diesen leiten Sie durch das Schlüsselwort ELSE bzw. ELSIF für Alternativbedingungen ein. Ausführliche Erläuterungen der einzelnen Schlüsselwörter finden Sie in den folgenden Abschnitten. Die Syntax für eine IF-Abfrage sieht wie folgt aus:

```
#IF (Bedingung)
  Code

#ELSIF (Bedingung)
  Code

#ELSE
```

```
  Code

#ENDIF
```

Die Schlüsselwörter für die bedingte Anweisung müssen Sie durch ein vorangestelltes Rautezeichen '#' kenntlich machen. Innerhalb einer IF-Abfrage stehen Ihnen Operatoren wie AND, OR und EQ zur Verfügung, um mehrere Teilbedingungen logisch zu verknüpfen. Eine vollständige Liste aller erlaubten Operatoren finden Sie im Abschnitt 3.4.15.3 **Operatoren** auf Seite 69. Operatoren können sowohl direkt hinter dem Schlüsselwort IF, als auch zwischen Klammerpaaren stehen.

Der Operator AND wird bei der Überprüfung der Bedingungen vor dem Operator OR ausgewertet. Wird der Operator NOT verwendet, kehrt sich der Effekt der Operatoren AND und OR um.

3.4.15.1 Stringvergleiche in IF-Abfragen

Beachten Sie beim Vergleich zweier Strings in IF-Abfragen folgende Hinweise:

1. Referenzierte Meta-Variablen sollten Sie mit dem Modus TEXT abfragen. In diesem Modus ist eingegebener HTML-Code maskiert (siehe Abschnitt 6.21 **Escaping Modes** auf Seite 149). Des Weiteren verhindert er Komplikationen, falls der Inhalt der Meta-Variablen Anführungszeichen enthält.
2. In der Syntax der IF-Abfrage müssen alle Variablen und Strings in Anführungszeichen gesetzt werden. Beispiel:

```
#IF ("<!--XX-TEXT:text1-->" EQ "das ist der Text")
#IF ("<!--XX-TEXT:text1-->" EQ "<!--XX-TEXT:text2-->")
```

   Bei XXDEFINED-Variablen (XXDEF) sind die Anführungszeichen nicht bindend.

3.4.15.2 Schlüsselwörter in IF-Abfragen

Mögliche Schlüsselwörter in IF-Abfragen sind:

3.4.15.2.1 IF (*Bedingung*)

Das Schlüsselwort *IF* leitet die Bedingte Anweisung ein. Die zu überprüfende Bedingung steht in Klammern. Darauf folgen die Anweisungen, die Imperia bei Zutreffen der Bedingung, bzw. bei einem wahren Ergebnis der Prüfung ausführen soll.

```
#IF (Bedingung)
```

Wenn Sie das Nichtzutreffen einer Bedingung prüfen wollen, verwenden Sie folgende Syntax:

```
#IF NOT (Bedingung)
```

Das Ergebnis dieser Prüfung ist wahr, wenn die Bedingung **nicht** zutrifft.

Hinweis:

Im Gegensatz zu früheren Imperia-Versionen müssen Sie Meta-Variablen innerhalb von Klammern nun in Anführungszeichen setzen.

3.4.15.2.2 THEN

Benutzen Sie dieses Schlüsselwort, wenn eine IF-Abfrage in nur einer Zeile steht. Beispiel:

```
#IF (Bedingung) #THEN (Anweisung)
```

Achtung:

Das Schlüsselwort THEN sollten Sie nicht mehr verwenden, da es zu Mehrdeutigkeiten kommen kann. Aus Kompatibilitätsgründen wird THEN jedoch noch unterstützt.

3.4.15.2.3 ELSE

Nach dem Schlüsselwort *ELSE* notieren Sie die Anweisungen, die Imperia ausführen soll, wenn die Überprüfung der Bedingung einen falschen Wert zurückliefert (siehe auch Abschnitt 3.4.15.5 **IF-Abfragen mit ELSE und ELSIF** auf Seite 71).

3.4.15.2.4 ELSIF (*Bedingung*)

Mit dem Schlüsselwort `ELSIF` leiten Sie einen Zweig der bedingten Anweisung ein, den Imperia ausführt, wenn vorangegangene Bedingungen sich als unwahr erwiesen haben und eine alternative Bedingung wahr ist (siehe auch Abschnitt 3.4.15.5 **IF-Abfragen mit ELSE und ELSIF** auf Seite 71).

3.4.15.2.5 ENDIF

Beendet eine IF-Abfrage.

3.4.15.3 Operatoren

Es stehen sowohl boolesche Operatoren als auch vergleichende Operatoren zur Verfügung.

3.4.15.3.1 Boolesche Operatoren

Operator	Beschreibung
AND	Stellt eine Und-Verknüpfung dar. Damit eine Bedingung mit einer Und-Verknüpfung zutrifft, müssen alle Unterbedingungen zutreffen.
OR	Stellt eine Oder-Verknüpfung dar. Damit eine Bedingung mit einer Oder-Verknüpfung zutrifft, muss mindestens eine Unterbedingung zutreffen.
XOR	Stellt eine Entweder-Oder-Verknüpfung dar. Damit eine Bedingung mit einer Entweder-Oder-Verknüpfung zutrifft, darf nur genau eine Unterbedingung zutreffen.
NOT	Stellt eine Nicht-Verknüpfung dar. Damit eine Bedingung mit einer Nicht-Verknüpfung zutrifft, muss die hinter dem Operator stehende Bedingung falsch sein.

Tabelle 3.12. Boolesche Operatoren

3.4.15.3.2 Vergleichende Operatoren

Operator	Beschreibung
EQ	Dieser Operator prüft, ob zwei Werte gleich sind. Der Code "foo" EQ "foo" ist wahr, "foo" EQ "bar" ist falsch.
NOT EQ (alias NE)	Dieser Operator prüft, ob zwei Werte nicht gleich sind. Der Code "foo" NOT EQ "bar" ist wahr, "foo" NOT EQ "foo" ist falsch.
CEQ	Dieser Operator prüft, ob zwei Strings, unabhängig von Groß- und Kleinschreibung, gleich sind. Der Code "foo" CEQ "FoO" ist wahr, aber "foo" EQ "FoO" ist falsch.
NOT CEQ (alias NCE)	Dieser Operator prüft, ob zwei Strings, unabhängig von der Groß- und Kleinschreibung, nicht gleich sind. Der Code "foo" NOT CEQ "bar" ist wahr, aber "foo" NOT CEQ "FoO" ist falsch.
REQ	Dieser Operator vergleicht einen Wert mit einem regulären Ausdruck. Näheres hierzu erfahren Sie in Abschnitt 3.4.16 **Reguläre Ausdrücke** auf Seite 73.
LT (oder <)	Dieser Operator prüft, ob ein Wert kleiner als ein zweiter Wert ist. Es wird ein alphanumerischer Vergleich vorgenommen, d.h. beim Vergleich zweier Strings wird der String als größer definiert, dessen Buchstaben im Alphabet später erscheinen.
LE (oder = <)	Dieser Operator prüft, ob ein Wert kleiner oder gleich einem zweiten Wert ist.
GT (oder >)	Dieser Operator prüft, ob ein Wert größer als ein zweiter Wert ist.
GE (oder >=)	Dieser Operator prüft, ob ein Wert größer oder gleich einem zweiten Wert ist.

Tabelle 3.13. Vergleichende Operatoren

3.4.15.4 Einfache IF-Abfrage

Eine einfache IF-Abfrage überprüft eine Bedingung. Trifft diese zu (bzw. ergibt die Prüfung einen wahren Wert), führt Imperia eine oder mehrere zuvor definierte Anweisungen aus. Andernfalls passiert nichts. Dazu ein Beispiel:

```
❶#IF ("<!--XX-showpic-->" EQ "yes")
❷  <img src="/images/grafik1.gif" width="40" height="20" border="0" />
❸#ENDIF
```

Hinweis:

Anstelle von XXDEF ist auch XXDEFINED möglich.

❶ In dieser Zeile beginnt die IF-Abfrage. Zunächst erfolgt die Definition der zu prüfenden Bedingung, in diesem Fall ob die Meta-Variable `showpic` den String "yes" enthält. Liefert diese Prüfung einen wahren Wert zurück, trifft die Bedingung zu und Imperia führt die folgende Anweisung aus.

❷ An dieser Stelle notieren Sie eine oder mehrere Anweisungen, die bei einem wahren Ergebnis der Bedingungsprüfung auszuführen sind. In diesem Beispiel ist das die Anzeige eines Bildes.

❸ Diese Zeile beendet die IF-Abfrage.

3.4.15.5 IF-Abfragen mit ELSE und ELSIF

Bei einer IF-Abfrage mit ELSE-Zweig können Sie auch für den Fall Anweisungen definieren, dass die abgefragte Bedingung unwahr ist. Dazu ein Beispiel:

```
#IF ("<!--XX-editmode-->")
  <img src="/images/dummies/topnavi_dummy.gif" border="0" />
#ELSE
  <!--CODEINCLUDE:navigation/topnavi.perl-->
#ENDIF
```

Die Bedingung ist wahr, wenn sich das Dokument im EDIT-Modus befindet. In diesem Fall erscheint ein Bild. Befindet sich das Dokument hingegen in einem anderen Modus, fügt Imperia stattdessen an dieser Stelle ein Codeinclude ein.

Wollen Sie mehr als einen Fall überprüfen und jeweils unterschiedliche Anweisungen ausführen lassen, bietet sich die Definition von Alternativbedingungen mit *ELSIF* an.

```
#IF ("<!--dirlevel:2-->" EQ "fussball")
  <img src="/images/icons/fussball.gif" border="0" />
#ELSIF ("<!--dirlevel:2-->" EQ "eishockey")
  <img src="/images/icons/eishockey.gif" border="0" />
#ELSIF ("<!--dirlevel:2-->" EQ "handball")
  <img src="/images/icons/handball.gif" border="0" />
#ELSE
  <img src="/images/icons/sport_default.gif" border="0" />
#ENDIF
```

In diesem Beispiel finden nacheinander mehrere Prüfungen auf die zweite Verzeichnisebene des Dokuments statt. Lautet diese "**fussball**", erscheint an dieser Stelle die Grafik `fussball.gif`. Trifft dies nicht zu, überprüft Imperia die nächste Bedingung und so fort. Der ELSE-Zweig stellt den (optionalen) Default-Fall dieser bedingten Anweisung dar. Diesen führt Imperia aus, wenn zuvor alle Bedingungsprüfungen ein unwahres Ergebnis zurückgeliefert haben.

3.4.15.6 Und-Verknüpfung

Eine Und-Verknüpfung besteht aus mindestens zwei Teilbedingungen und liefert nur dann ein wahres Ergebnis zurück, wenn alle Teilbedingungen erfüllt sind. Die Syntax einer Und-Verknüpfung ist wie folgt:

```
#IF ((Bedingung1) AND (Bedingung2))
  Anweisung1
  Anweisung2
#ENDIF
```

3.4.15.7 Oder-Verknüpfung

Eine Oder-Verknüpfung besteht ebenfalls aus mindestens zwei Teilbedingungen und liefert ein wahres Ergebnis, wenn mindestens eine Bedingung oder aber alle Bedingungen erfüllt sind. Die Syntax einer Oder-Bedingung ist folgendermaßen:

```
#IF ((Bedingung1) OR (Bedingung2))
  Anweisung1
  Anweisung2
#ENDIF
```

3.4.15.8 Verschachtelte IF-Abfragen

Verschachtelte IF-Abfragen überprüfen Bedingungen in Abhängigkeit von anderen Bedingungen.

Sie können bedingte Anweisungen beliebig tief verschachteln. Beachten Sie jedoch, dass mit zunehmender Schachtelungstiefe die Abarbeitungsgeschwindigkeit abnimmt. Damit Sie in tief verschachtelten IF-Abfragen leichter die Übersicht behalten und um die Pflege zu erleichtern, können Sie innerhalb des Konstrukts erläuternde HTML-Kommentare notieren.

Das folgende Beispiel zeigt die Syntax einer verschachtelten IF-Abfrage:

```
❶#IF ("<!--dirlevel1-->" EQ "sport")
  ❷#IF ("<!--dirlevel:2-->" EQ "fussball")
    <img src="/images/icons/fussball.gif" border="0" />
  #ELSIF ("<!--dirlevel:2-->" EQ "eishockey")
    <img src="/images/icons/eishockey.gif" border="0" />
  #ELSIF ("<!--dirlevel:2-->" EQ "handball")
    <img src="/images/icons/handball.gif" border="0" />
  #ELSE
    <img src="/images/icons/sport_default.gif" border="0" />
  ❸#ENDIF
❹#ELSIF ("<!--dirlevel1-->" EQ "kultur")
  ❺#IF ("<!--dirlevel:2-->" EQ "theater")
    <img src="/images/icons/theater.gif" border="0" />
  #ELSIF ("<!--dirlevel:2-->" EQ "ausstellung")
    <img src="/images/icons/museum.gif" border="0" />
  #ELSIF ("<!--dirlevel:2-->" EQ "konzert")
    <img src="/images/icons/musik.gif" border="0" />
  #ELSE
    <img src="/images/icons/kultur_default.gif" border="0" />
  ❻#ENDIF
❼#ENDIF
```

In obenstehendem Beispiel findet eine Überprüfung der obersten Verzeichnisebene des Dokuments statt. Abhängig vom Ergebnis dieser Abfrage prüft Imperia jeweils eine Reihe unterschiedlicher Alternativbedingungen für die zweite Verzeichnisebene. Das Ergebnis dieser zweiten Prüfung löst dann letztendlich die Anzeige einer bestimmten Grafik aus.

❶ Der Beginnn der bedingten Anweisung. In dieser Zeile steht die erste Bedingung, die Imperia für die erste Verzeichnisebene des Dokuments prüfen soll.
❷ Liefert die Prüfung der ersten Bedingung für die erste Verzeichnisebene einen wahren Wert zurück, führt Imperia den nun folgenden Anweisungsblock aus. Dabei handelt es sich um die Überprüfung einiger Bedingungen für die zweite Verzeichnisebene des Dokuments wie sie auch in Abschnitt 3.4.15.5 **IF-Abfragen mit ELSE und ELSIF** auf Seite 71 stattfindet.
❸ Ende des Anweisungsblocks, den Imperia bei Zutreffen der ersten Bedingung ausführt.
❹ Diese Zeile enthält die Definition der zweiten Bedingung für die erste Schachtelungsebene der bedingten Anweisung.
❺ In dieser Zeile beginnt der Anweisungsblock für das Zutreffen der zweiten Bedingung. Es handelt sich erneut um die Überprüfung einer Reihe von Bedingungen für die zweite Verzeichnisebene des Dokuments.
❻ In dieser Zeile endet die bedingte Anweisung für die zweite Schachtelungsebene.
❼ Dieses ENDIF markiert das Ende der gesamten bedingten Anweisung.

3.4.15.9 Kommentar in #IF-Abfragen

Um die Übersichtlichkeit von tief verschachtelten IF-Abfragen zu verbessern, besteht die Möglichkeit, Zeilen oder Bedingungen mit HTML-Kommentaren zu versehen. Die Syntax für einen Kommentar ist wie folgt:

```
<!--Hier steht ein Kommentar-->
```

3.4.16 Reguläre Ausdrücke

Als Reguläre Ausdrücke bezeichnet man eine Reihe von Symbolen und syntaktischen Elementen, mit denen Sie innerhalb von Texten bestimmte Muster finden können. Sie lassen sich für alle Arten von Textmanipulation verwenden. In Funktionen wie **Suchen** bzw. **Suchen und Ersetzen** sorgen sie beispielsweise für eine Verfeinerung der Suchkriterien. Bei der Untersuchung eines Texts oder eines Datenstroms dienen Sie zur Prüfung auf bestimmte Bedingungen.

Die Möglichkeiten von Regulären Ausdrücken lassen sich im Rahmen dieser Dokumentation leider nicht vollständig abhandeln. Wenn Sie mehr über dieses Thema erfahren möchten, empfehlen wir das Standardwerk *Jeffrey E. Friedl.: Mastering Regular Expressions: Powerful Techniques for Perl and Other Tools* aus dem O'Reilly-Verlag.

In den folgenden Abschnitten finden Sie ein paar einfache Beispiele für die Verwendung von Regulären Ausdrücken in bedingten Anweisungen in Imperia.

3.4.16.1 Beispiele für Reguläre Ausdrücke

Folgender Code prüft, ob die Meta-Variable `defaultimage` auf dem String `default.jpg` endet:

```
#IF ("<!--XX-defaultimage-->" REQ "default\.jpg$")
```

Folgende Syntax prüft, ob die Meta-Variable `textfeld` wenigstens 10 Zeichen enthält:

```
#IF ("<!--XX-textfeld-->" REQ ".{10}")
```

Folgender Code überprüft, ob die Meta-Variable `keywords` das Wort *Computer* enthält:

Hinweis:

Mit den anderen Operatoren könnten Sie eine solche Abfrage nicht verwirklichen.

```
#IF ("<!--XX-keywords-->" REQ "Computer")
```

Die folgende Syntax prüft, ob die Datenausgabe in einer Meta-Variablen aus dem Monat Dezember stammt:

```
#IF ("<!--XX-datum-->" REQ "^\d{1,2}\.12\.\d{2,4}")
```

3.5 Dynamische Module

Mit dynamischen Modulen können Sie bestimmte Teile des Templates durch HTML-Code oder durch den Inhalt externer Dateien ersetzen. Es ist sogar möglich, nur ein einziges Template zu verwenden, das erst durch dynamische Ersetzungen entsprechend modifiziert wird.

Die Ersetzungen, die durch dynamische Module vorgenommen werden sollen, konfigurieren Sie in der Datei `dynamic.conf`, die Sie im Verzeichnis `site/config` finden. Zum Einstieg ein kleines Beispiel für Einträge in dieser Datei:

```
IF [<!--dirlevel:1-->] EQUALS [agenda] OR [news]
OR [politik] OR [tnt]
REPLACE [<!--standard-desc-->] BY[ProSieben Online - agenda]
REPLACE [<!--standard-key-->] BY [ProSieben Online, tv, nachrichten,
aktuell, news,
hintergrund, reportage, fussball, sport, klatsch, wetter, real video]
REPLACE [<!--main-navigation-->] BY [FILE:n_ak_ne.txt]
REPLACE [<!--sub-navigation-->] BY [FILE:nsakne.txt.txt]
FI
```

Die folgenden Abschnitte erklären die Syntax der `dynamic.conf`. Einige Beispiele für die Funktionen und Möglichkeiten finden Sie in den Abschnitten ab Abschnitt 3.5.2 **Einfache Ersetzung** auf Seite 74 .

3.5.1 Syntax-Referenz für die dynamic.conf

Die in der `dynamic.conf` verwendete Syntax ist denkbar einfach und kommt mit einem Mindestmaß an Schlüsselwörtern aus. Grundsätzlich ist eine gültige Anweisung folgendermaßen aufgebaut:

```
IF [wert1] EQUALS [wert2] OR [wert3]
REPLACE [Zeichenkette] BY [Zeichenkette] TIMES [Anzahl der Ersetzungen]
REPLACE...
REPLACE...
FI
```

Die Schlüsselwörter im Einzelnen:

IF

Das Schlüsselwort `IF` leitet den Vergleich ein. Werte, die verglichen werden sollen, werden in eckige Klammern eingefasst.

EQUALS

Das Schlüsselwort `EQUALS` ist der Vergleichsoperator. Der im ersten Klammernpaar stehende Wert bzw. der Inhalt einer dort eingetragenen Meta-Variablen muss gleich dem Wert des zweiten Klammernpaares bzw. dem Inhalt der dort eingetragenen Meta-Variablen sein.

OR

Dieses Schlüsselwort bewirkt eine Oder-Verknüpfung. Hiermit können mehrere Werte in den Vergleich einbezogen werden. Es wird dann geprüft, ob mindestens einer der angegebenen Werte die Bedingung erfüllt. Eine Und-Verknüpfung kann nur durch zwei aufeinanderfolgende IF-Abfragen simuliert werden.

REPLACE

Dieses Schlüsselwort leitet die Ersetzung ein. Es wird gefolgt von der Zeichenkette in eckigen Klammern, die ersetzt werden soll. Nach dieser Zeichenkette muss das Schlüsselwort `BY` folgen (siehe unten).

BY

Hinter diesem Schlüsselwort steht die Zeichenkette in eckigen Klammern, mit der die durch `REPLACE` definierte Zeichenkette ersetzt wird.

TIMES

Dieses Schlüsselwort wird in einer `REPLACE`-Anweisung verwendet. Es bewirkt, dass nur eine bestimmte Anzahl an Ersetzungen durchgeführt wird.

FI

Dieses Schlüsselwort beendet eine IF-Abfrage.

3.5.2 Einfache Ersetzung

In den folgenden Abschnitten wird ein einfaches Beispiel aufgebaut, um die Mechanismen und Funktionen dynamischer Templates zu verdeutlichen.

Als Voraussetzung benötigen wir:

- Ein Template, das an irgendeiner Stelle das Wort `Sparte` enthält. Dieses Wort soll später dynamisch ersetzt werden.
- Vier Rubriken: Sport, Politik, Kunst und Kultur.
- Eine Metadatei pro Rubrik. Jede Metadatei muss auf das zur jeweiligen Rubrik gehörende Verzeichnis verweisen.

Zum Anpassen der `dynamic.conf` wird diese im internen Editor oder in einem beliebigen externen Editor geöffnet.

Hinweis:

Die originale `dynamic.conf` *beinhaltet Beispielcode. Kommentieren Sie diesen durch Rauten (#) aus oder sichern Sie die vorhandene* `dynamic.conf` *unter einem anderen Namen.*

Geben Sie folgenden Code ein:

```
IF [<!--dirlevel:1-->] EQUALS [sport]
  REPLACE [Sparte] BY [Kategorie: <b>Sport</b>]
FI
IF [<!--dirlevel:1-->] EQUALS [politik]
  REPLACE [Sparte] BY [Kategorie: <b>Politik</b>]
FI
IF [<!--dirlevel:1-->] EQUALS [kultur]
  REPLACE [Sparte] BY [Kategorie: <b>Kultur</b>]
FI
IF [<!--dirlevel:1-->] EQUALS [kunst]
  REPLACE [Sparte] BY [Kategorie: <b>Kunst</b>]
FI
```

Die erste Zeile vergleicht den Inhalt der ersten Meta-Variablen `<!--dirlevel:1-->` mit dem String `sport`. Sie erinnern sich: diese Meta-Variable enthält den ersten Teil des Pfades der Datei (siehe auch Abschnitt 3.4.12 **Pfad-Bestandteile auslesen** auf Seite 59). Stimmt nun der Inhalt der Meta-Variablen mit dem String überein, trifft die Bedingung zu und die nächste Zeile, die REPLACE-Anweisung, wird bearbeitet.

Die REPLACE-Anweisung sucht im Template den Inhalt des ersten eckigen Klammerpaares und ersetzt ihn durch den Inhalt des zweiten eckigen Klammerpaares. In unserem Beispiel wird also der String `Sparte` durch den String `Kategorie: <b>Sport</b>` ersetzt. Wie Sie sehen, kann auch HTML-Code verwendet werden, der im Template richtig angezeigt wird.

Die letzte Zeile enthält das Schlüsselwort `FI` mit dem die IF-Abfrage beendet wird.

Die anderen drei IF-Abfragen unseres Beispiels funktionieren nach dem gleichen Prinzip: trifft eine bestimmte Bedingung zu, wird ein definierter String durch einen anderen ersetzt.

3.5.3 Ersetzung durch eine Oder-Verknüpfung

Eine Oder-Verknüpfung überprüft, ob mindestens eine der angegebenen Bedingungen erfüllt wird. Ist dies der Fall, werden die angegebenen Anweisungen ausgeführt.

Im folgenden Beispiel wird die Hintergrundfarbe der Seite abhängig von einem Verzeichnis ersetzt:

```
IF [<!--dirlevel:1-->] EQUALS [sport] OR [politik]
  REPLACE [#ffffff] BY [#009933]
FI
IF [<!--dirlevel:1-->] EQUALS [kunst] OR [kultur]
  REPLACE [#ffffff] BY [#cc3300]
FI
```

Beide Oder-Abfragen funktionieren auf die gleiche Art und Weise. Es wird verglichen, ob der Wert der Meta-Variablen `<!--dirlevel:1-->` mit dem String `sport` oder dem String `politik` übereinstimmt. Trifft die Bedingung zu, wird die Ersetzung vorgenommen.

3.5.3.1 Und-Verknüpfung

Es gibt in der Syntax der `dynamic.conf` keine direkte Und-Verknüpfung. Eine Und-Verknüpfung kann jedoch durch zwei verschachtelte IF-Abfragen simuliert werden:

```
 IF [<!--dirlevel:1-->] EQUALS [sport]
  IF [<!--dirlevel:2-->] EQUALS [tennis]
    REPLACE [Sportart] BY [Tennis]
  FI
FI
```

3.5.4 Anzahl der Ersetzungen definieren

Falls der zu ersetzende String mehrfach in einem Template vorkommt, können Sie auch bestimmen, wie oft die Ersetzung vorgenommen werden soll. Dies geschieht über das Schlüsselwort TIMES. Die Syntax hierfür ist wie folgt:

```
IF [Ausdruck] EQUALS [String]
  REPLACE [Suchstring] BY [Ersetzungsstring] TIMES [Anzahl Ersetzungen]
FI
```

Beispiel:

```
IF [<!--XX-ersetzen-->] EQUALS [1]
  REPLACE [Sparte] BY [Kategorie: <b>Sport</b>] TIMES [5]
FI
```

In diesem Beispiel wird geprüft, ob die Meta-Variable `<!--XX-ersetzen-->` den Wert `1` hat. Ist dies der Fall, wird im Template fünfmal der String `Sparte` durch den String `Kategorie: <b>Sport</b>` ersetzt.

3.5.5 Ersetzung durch eine externe Datei

Imperia bietet die Möglichkeit, in einem Template einen beliebigen String durch den Inhalt einer externen Datei zu ersetzen. Diese Funktion bietet sich an, wenn die Ersetzung mehrere Zeilen enthält. Dateien, die eingefügt werden sollen, unterliegen keinerlei Namenskonventionen, müssen aber im Verzeichnis `/site/dynamic` liegen und im ASCII-Format vorliegen.

Die Syntax für eine Ersetzung durch den Inhalt einer externen Datei ist wie folgt:

```
IF [Ausdruck] EQUALS [String]
  REPLACE [Suchstring] BY [FILE:Dateiname]
```

3.5.6 Teile der dynamic.conf aus externen Dateien laden

Bei intensiver Nutzung der Möglichkeiten der `dynamic.conf` kann diese erheblich anwachsen. Aus diesem Grund besteht die Möglichkeit, externe Dateien einzuschließen und somit die Größe der `dynamic.conf` zu verringern.

Nachgeladene Dateien werden am Ende der `dynamic.conf` angehängt. Es gibt keine Namenskonventionen für die Benennung dieser Dateien, sie müssen lediglich im Verzeichnis `/site/dynamic` oder in Unterverzeichnissen dieses Verzeichnisses abgelegt werden.

Ein Aufruf einer externen Datei sieht wie folgt aus:

```
IF [Ausdruck] EQUALS [String] INCLUDE [Dateiname]
```

Beispiel:

```
IF [<!--XX-template-->] EQUALS [news] INCLUDE [basic.conf]
```

In diesem Beispiel wird überprüft, ob das verwendete Template den Namen `news` hat. Ist dies der Fall, wird die Datei `/site/dynamic/basic.conf` an das Ende der `dynamic.conf` angehängt.

3.5.6.1 Externe Datei aus einem Unterverzeichnis laden

Liegt die externe Datei in einem Unterverzeichnis des Verzeichnisses `/site/dynamic`, sieht der Aufruf wie folgt aus:

```
IF [Ausdruck] EQUALS [String] INCLUDE [Pfad/Dateiname]
```

Beispiel:

```
IF [<!--XX-wochentag-->] EQUALS [montag] INCLUDE
[wochentag/montag.conf]
```

Dieser Code überprüft, ob die Meta-Variable `<!--XX-wochentag-->` den String `montag` enthält. Ist dies der Fall, wird der Inhalt der Datei `/site/dynamic/wochentag/montag.conf` an das Ende der `dynamic.conf` angehängt.

3.6 Imperiablocks

Imperiablocks bieten dem Benutzer die Möglichkeit, bestimmte Abschnitte des Template-Codes beliebig oft zu duplizieren. Sie können aus beliebigen Code-Fragmenten bestehen. Das Duplikat eines Imperiablocks heißt in der Imperia-Terminologie **Instanz**. Befindet sich ein Dokument im EDIT-Modus, können Sie Instanzen beliebig oft erzeugen oder löschen.

Informationen zur Bedienung von Imperiablocks während des Editierens finden Sie im Kapitel **Imperiablocks** des Benutzerhandbuchs.

Einen Imperiablock definieren Sie im Template durch folgende Schlüsselwörter:

```
<!--IMPERIABLOCK-->
<!--/IMPERIABLOCK-->
```

Zwischen den Schlüsselwörtern steht der zu duplizierende Code. Imperiablocks können beliebig oft in einem Template vorkommen. Es ist jedoch nicht möglich, Imperiablocks zu schachteln oder durch Ersetzungen in der `dynamic.conf` zu erstellen. Auch innerhalb von Arrayblocks (siehe Abschnitt 3.3.12 **Arrayblocks** auf Seite 36) können Sie keine Imperiablocks einsetzen. Imperiablock-Code muss immer im jeweiligen Template vorhanden sein und darf 6000 Zeichen pro Kopie bzw. 150 Textfelder nicht überschreiten.

Ein Imperiablock ist bei der Bearbeitung des Dokuments durch einen grauen Balken gekennzeichnet, der am rechten Rand die Bedienungselemente enthält (siehe Screenshot).

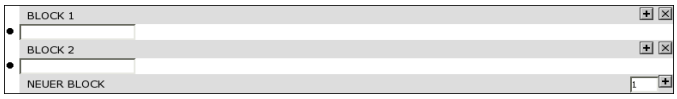

Abb. 3.6: Imperiablock mit zwei Instanzen im Edit-Modus

Der folgende Beispiel-Code erzeugt einen Imperiablock zur Erstellung einer ungeordneten Liste:

```
<ul>
<!--IMPERIABLOCK-->
  <li>
    <textarea name="IMPERIA" rows="5" cols="40"></textarea>
  </li>
<!--/IMPERIABLOCK-->
</ul>
```

Grundsätzlich fügt Imperia jede neu erzeugte Instanz eines Imperiablocks oberhalb des Originals ein. Um eine neue Instanz an anderer Position einzufügen, tragen Sie die gewünschte Position für das neue Element in das Eingabefeld neben dem Button zum Hinzufügen eines neuen Blocks ein.

3.6.1 Imperiablock-Variablen

Folgende Variablen können Sie verwenden, um Meta-Variablen innerhalb eines Imperiablocks zu referenzieren:

<!--BLOCK_INDEX-->

Diese Variable enthält den numerischen Index eines Imperiablocks im Template beginnend mit `0`.

<!--BLOCK_ID-->

Diese Variable enthält die numerische ID der Instanz eines Imperiablocks.

Des Weiteren stehen Variablen zur Verfügung, die innerhalb einer Instanz den Inhalt einer bestimmten Meta-Variablen liefern. Mit diesen Variablen kann Inhalt, der in einer Meta-Variablen gespeichert wurde, an anderer Stelle innerhalb der Instanz erneut verwendet werden.

<!--XX-IBLOCK-*Meta-Variable*-->

Diese Variable enthält den Wert der benannten Meta-Variablen `metavar` innerhalb der Instanz.

<!--XX-IBLOCK-*Nummer*-->

Diese Variable enthält den Wert einer unbenannten Meta-Variablen innerhalb einer Instanz. Anstelle des Platzhalters *Nummer* müssen Sie die Position der gewünschten Meta-Variablen in der Instanz angeben. Beispielsweise würde die Variable <!--XX-IBLOCK-2--> den Wert der dritten unbenannten Meta-Variablen in der Instanz enthalten.

3.6.2 Inhalt außerhalb eines Imperiablocks referenzieren

Damit Inhalt eines Imperiablocks an anderer Stelle im Dokument referenzierbar ist, ergänzt Imperia die Namen von darin einthaltenen Meta-Variablen automatisch. Hierbei wird zwischen benannten und anonymen Meta-Variablen unterschieden. Die Benennung von Meta-Variablen geschieht wie folgt:

Anonyme Meta-Variablen erhalten automatisch einen Namen, der sich folgendermaßen zusammensetzt:

- der String `imperiablock`
- Block-Index
- Block-ID
- die Nummer der anonymen Meta-Variablen

Bei benannten Meta-Variablen findet folgende Ergänzung statt:

- Block-Index
- Block-ID

Über den Namen, den Block-Index und die Block-ID ist jede Meta-Variable eindeutig referenzierbar. Nutzen Sie hierzu folgende Syntax:

`<!--XX-`*Meta-Variable_Block-Index_Block-ID*`-->`

Dieser Ausdruck greift auf den Inhalt einer Meta-Variablen in einer bestimmten Instanz in einem bestimmten Imperiablock zu. Der Imperiablock ist eindeutig durch den Block-Index und die Block-ID identifiziert.

Beispiel:

Angenommen ein Template enthält zwei Imperiablocks, mit jeweils einem Inputfeld. Das Inputfeld des ersten Imperiablocks hat den Name `input`, das Inputfeld des zweiten Imperiablocks ist nicht benannt, also anonym.

Der erste Imperiablock erhält den Block-Index `0`. Wenn Sie die erste Instanz des ersten Blocks erzeugen, ergänzt das System den Namen der aus dem Inputfeld resultierenden Meta-Variablen automatisch mit dem Block-Index und der Block-ID: `input_0_0`. Bei jeder weiteren Instanz dieses Imperiablocks erhöht sich die Block-ID jeweils um 1 und der Name der zugehörigen Meta-Variablen ändert sich: `input_0_1`, `input_0_2` usw..

Der zweite Imperiablock erhält als Block-Index die `1`. Wenn Sie die erste Instanz dieses Imperiablocks erzeugen, bildet das System den Namen der aus dem anonymen Inputfeld resultierenden Meta-Variablen aus dem String `imperiablock`, dem Block-Index, der Block-ID und einer fortlaufenden Nummerierung. Die fortlaufende Nummerierung fängt den Fall ab, dass mehrere anonyme Felder in einem Imperiablock vorhanden sind. Die resultierenden Meta-Variablennamen lauten dann wie folgt: `imperiablock_1_0_0`, `imperiablock_1_0_1` etc..

Erzeugen Sie eine weitere Instanz des zweiten Imperiablocks, erhöht sich automatisch die Block-ID um 1 und die fortlaufende Nummerierung der anonymen Meta-Variablen für diese Instanz beginnt wieder bei 0. Resultat: `imperiablock_1_1_0`, `imperiablock_1_1_1`, `imperiablock_1_1_2` etc..

3.6.3 Imperiablocks in Flexmodulen

Imperiablocks können Sie auch in Flexmodulen mit der oben beschriebenen Syntax verwenden. Bitte beachten Sie folgende Dinge bei der Verwendung von Imperiablocks:

- Der Zugriff auf Variablen ist nur über folgende Syntax möglich:

  `<!--XX-FLEX-IBLOCK-`*Meta-Variable*`-->`

- Imperia ergänzt die Namen von Meta-Variablen durch Erweiterungen des Flexmoduls und des Imperiablocks. Dies gilt sowohl für benannte als auch für anonyme Meta-Variablen. Zuerst erfolgen die Ergänzungen des Flexmoduls und danach die des Imperiablocks.

Beispiel für Meta-Variablennamen aus benannten Feldern:

1. ursprünglicher Name: `text1`
2. mit der Namenserweiterung des Flexmoduls: `text1_2_5`
3. mit der Namenserweiterung des Imperiablocks: `text1_2_5_0_1`

Beispiel für Meta-Variablennamen aus anonymen Feldern:

1. **kein** ursprünglicher Name
2. mit der Namenserweiterung des Flexmoduls: `imperiaflex_2_5_0`
3. mit der Namenserweiterung des Imperiablocks: `imperiaflex_2_5_0_0_1`

Hinweis:

Es ist zu beachten, dass anonyme Meta-Variablen in einem Imperiablock innerhalb eines Flexmoduls als Basisnamen `imperiaflex` *und nicht* `imperiablock` *erhalten.*

3.7 Das Metatool

Mit Hilfe des Metatools können Sie aus bereits erstellten Dokumenten, den Quelldokumenten, den Inhalt bestimmter Meta-Variablen an definierte Zielpositionen im Zieldokument einfügen. Dadurch können Sie manuell Leitseiten erstellen. Ändert sich der Inhalt im Quelldokument, wird das Zieldokument jedoch nicht automatisch aktualisiert. Dafür stehen SiteActives zur Verfügung.

Im Metatool können mehrere Zielpositionen zu sogenannten Feature-Sets zusammengefasst werden. Weiterhin können mit Hilfe des Metatools Linklisten erzeugt werden.

Feature-Sets dienen dazu, mehrere Zielpositionen des Zieldokuments zusammenzufassen. Jede dieser Zielpositionen wird im Metatool mit dem gleichen Namen und einer fortlaufenden Nummer bezeichnet. Feature-Sets sind meist Gruppen von Zielpositionen, die thematisch zusammengehören.

Die Oberfläche des Metatools wird über verschiedene Parameter gestaltet. Die Bedienung und der Aufbau des Metatools wird im Kapitel **Metatool** des Userhandbuchs beschrieben.

3.7.1 Aufruf des Metatools 2 im Template

Der Aufruf des Metatools in Imperia 8 hat sich gegenüber der Vorversion grundlegend geändert. Das alte Metatool wurde z.B.: mit dem folgendem Ausdruck aufgerufen:

```
<a href="javascript: var win=open('/cgi-bin/site_metatool.pl? \
MYURL=<!--XX-directory-->/<!--XX-filename-->: \
SELFILTER=directory,title:ANZMAINS=2:SELSYNC_abc=title: \
SELSYNC_xyz=directory:DISPLAY_LISTELEMS=0:'','
'toolbar=no,width=700,height=500,directories=no,status=yes, \
 scrollbars=yes,resize=yes,resizable,menubar=no,location=no, \
 copyhistory=no')" >
    <img src="/imperia/images/metatool/mtt.gif" width="250" height="30" >
</a>
```

Beim aktuellen Metatool verwenden Sie für die Konfiguration Processing Instructions statt Aufrufparametern. In der Syntax für das aktuelle Metatool sieht der Aufruf aus dem obigen Beispiel folgendermaßen aus:

```
<? imperia metatool start ?>
    <? filter metafield directory ?>
    <? filter metafield title ?>
    <? link feature element_count 2 ?>
    <? link feature sync to="abc" from="title" ?>
    <? link feature sync to="xyz" from="directory" ?>
    <? link linklist element_count 0 ?>
<? imperia metatool end ?>
```

Das neue Metatool binden Sie mit der Anweisung

```
<? imperia metatool start ?>
<? imperia metatool end ?>
```

in ein Template ein. Zwischen diesen beiden Imperia-Tags konfigurieren Sie das Metatool mit Processing Instructions, die in den folgenden Abschnitten im Detail erklärt werden. Diese übernehmen die Aufgaben der Attribute aus dem Aufruf des alten Metatools und sind in folgende Gruppen eingeteilt:

- setup
- gui
- prefilter
- filter
- link

In der Regel können Sie die Processing Instructions nur genau einmal verwenden. Ausnahmen sind explizit bei der jeweiligen Anweisung vermerkt. Wenn Sie Anweisungen mehrfach verwenden, die nicht dafür vorgesehen sind, ist das Verhalten nicht vorhersehbar.

Erkennt das System eine Anweisung beim Parsen nicht, so wird diese Anweisung als Kommentar in das Template geschrieben. z.B.:

```
<!-- unmatched processing instructions
                        <? setup foo 110 ?>
     unmatched processing instructions -->
```

Auf diesem Weg können Sie Tippfehler im Template schnell finden, indem Sie sich den Quellcode der Seite im Editmode anschauen.

Für die Einbindung des Metatools ist außerdem der Aufruf eines Postconvert-Plug-Ins im Template notwendig. Das Plug-In rufen Sie mit folgender Anweisung auf:

```
<!--postconvert:Mtt2()-->
```

Positionieren Sie diese Anweisung im Template zwischen `<!--formend-->` und `</body>`.

3.7.2 Setup-Anweisungen

Mit den Setup-Anweisungen steuern Sie das allgemeine Verhalten des Metatools.

- <? setup version *Zahl* ?>

  Parameter : *Zahl* kann den Wert 1 oder 2 annehmen. Ist der Parameter nicht gesetzt, wird per Default der Wert 2 angenommen.

  Beschreibung : Über die Versionsanweisung stellen Sie ein, ob Sie das Metatool 1 oder das Metatool 2 verwenden.

  

  Achtung:

  Beachten Sie, dass im Metatool 1 nicht alle Funktionen zur Verfügung stehen.

  Kompatibilität : Kann erst in der neuen Templatesyntax verwendet werden.

- <? setup id *Zahl* ?>

  Parameter : *Zahl* kann ein Wert größer als 0 sein.

  Beschreibung : Über die Id können Sie intern zu jeder verwendeten Metatool-Instanz die ausgewählten Werte speichern lassen. Dies ist bei der Verwendung mehrerer Metatool-Instanzen innerhalb eines Templates vorteilhaft.

  Wird dieser Parameter nicht gesetzt, ist eine Unterscheidung zwischen den Instanzen nicht möglich. Das entspricht dem Verhalten des Metatool 1.

  Kompatibilität : Ist erst in Version 2 mit der neuen Templatesyntax implementiert.

- <? setup myurl *String* ?>

  Parameter : *String* kann den Wert <!--XX-directory-->/<!--XX-filename--> aufnehmen.

  Beschreibung : Über diese Anweisung übergeben Sie dem Metatool die URL des aktuell bearbeiteten Dokumentes. Dadurch wird es bei der Anzeige herausgefiltert, so dass keine Referenzen auf das Dokument selbst entstehen.

  Diese Anweisung ist optional. Fehlt diese Anweisung, wird automatisch der Wert <!--XX-directory-->/<!--XX-filename--> aus den entsprechenden Metafeldern des Dokuments verwendet.

  Kompatibilität : Ersetzt den Parameter MYURL=*String* aus Version 1.

- <? setup debug ?>

  Parameter : ohne Parameter

  Beschreibung : Mit diesem Schalter aktivieren Sie die Anzeige zusätzlicher Debuginformationen im Metatool.

  Kompatibilität : Erst in Version 2 implementiert.

- <? setup store_filter_values ?>

  Parameter : ohne Parameter

Beschreibung : Mit dieser Anweisung legen Sie fest, ob einmal eingestellte Filterwerte je Dokument gespeichert werden sollen. Vom Redakteur eingestellte Filterwerte werden gespeichert und beim nächsten Aufruf des Metatools aus diesem Dokument wieder angezeigt. Ohne diese Anweisung werden die vom Redakteur verwendeten Suchbegriffe nicht gespeichert.

Kompatibilität : Ersetzt den Parameter STOREFILTER=1 aus Version 1.

- <? setup pic_sync ?>

  Parameter : ohne Parameter

  Beschreibung : Wenn Sie Bilder über das Metatool in das aktuelle Dokument übernehmen wollen, können Sie mit dieser Anweisung beispielsweise Media-Asset-Managementaufrufe synchronisieren.

  Kompatibilität : Ersetzt den Parameter PICSYNC=1 aus Version 1.

- <? setup charset *String* ?>

  Parameter : Mit *String* kann ein anderes Charset übergeben werden.

  Beschreibung : Über diese Anweisung lässt sich das Standard-Imperia-Charset-Verhalten umgehen.

  

  Achtung:

  Diese Anweisung sollte mit Vorsicht verwendet werden.

  Kompatibilität : Ersetzt den Parameter CHARSET=*String* aus Version 1.

3.7.3 GUI-Anweisungen

Durch die GUI-Anweisungen beeinflussen Sie das Aussehen der Benutzeroberfläche im Metatool.

- <? gui docs_per_page *Zahl* ?>

  Parameter : *Zahl* kann ein ganzzahliger Wert größer 0 sein.

  Beschreibung : Die Anzahl im Metatool-Dialogfenster angezeigter Dokumente pro Seite legen Sie mit dieser Anweisung fest. Wird diese Anweisung nicht gesetzt, werden 20 Treffer pro Seite angezeigt.

  Kompatibilität : Ersetzt den Parameter DISPLAY_PERPAGE=*Zahl* aus Version 1.

- <? gui hide_title ?>

  Parameter : ohne Parameter

  Beschreibung : Mit diesem Schalter steuern Sie, ob der Titel des Dokuments im Metatool verborgen wird. Per Default wird der Titel angezeigt.

  Kompatibilität : Ersetzt den Parameter HIDE_TITLE=1 aus Version 1.

- <? gui label key="*Metafeld*" value="*String*" ?>

  Parameter : *key* enthält den Namen eines Metafeldes, *value* enthält den freien Text, der angezeigt werden soll.

  Beschreibung : Über die Label-Anweisung können Sie für Metafelder, die als Filter dienen, einen anderen Text anzeigen lassen. Im Feld `key` tragen Sie den Namen des Metafeldes ein, *value* enthält den anzuzeigenden Ersatztext.

  

  Hinweis:

  Diese Anweisung darf mehrfach vorkommen.

  Kompatibilität : Ist erst in Version 2 implementiert.

- <? gui use_as_title *Metafeld* ?>

  Parameter : *Metafeld* Name des Metafeldes, das title ersetzen soll.

Beschreibung : der Inhalt des mit *Metafeld* bezeichneten Metafeldes wird anstelle des Dokumententitels in der Auswahlliste angezeigt.

Kompatibilität : Ersetzt den Parameter DISPLAY_TITLE=*Metafeld* aus Version 1.

- <? gui show_additionaly *Metafeld* ?>

  Parameter : *Metafeld* Name des anzuzeigenden Metafeldes.

  Beschreibung : Mit dieser Anweisung veranlassen Sie die Anzeige zusätzlicher Metafelder im Metatool.

  

  Hinweis:

  Diese Anweisung darf mehrfach vorkommen.

  Kompatibilität : Ersetzt den Parameter DISPLAY_CONTENT1=*Metafeld* aus Version 1.

- <? gui sort_by *Metafeld* ?>

  Parameter : *Metafeld* Name des Metafeldes, das als Sortierkriterium dienen soll.

  Beschreibung : Mit dieser Anweisung verändern Sie die Sortierreihenfolge im Metatool. Per Default wird nach dem Modifikationsdatum (__imperia_modified) sortiert.

  Kompatibilität : Ersetzt den Parameter SORTBY=*Metafeld* aus Version 1.

- <? gui use_as_image *Metafeld* ?>

  Parameter : *Metafeld* Name des Metafeldes, das die URL der anzuzeigenden Grafik enthält.

  Beschreibung : Mit use_as_image können Sie ein Bild pro Dokument im Metatool anzeigen lassen.

  Kompatibilität : Ersetzt den Parameter DISPLAY_IMAGE=*Metafeld* aus Version 1.

- <? gui hide_filter_headlines ?>

  Parameter : ohne Parameter

  Beschreibung : Dieser Schalter steuert die Anzeige von Überschriften oberhalb von Vorfiltern und Benutzerfiltern.

  Kompatibilität : Ist erst in Version 2 implementiert.

- <? gui show_cattree_nodeid ?>

  Parameter : ohne Parameter

  Beschreibung : Dieser Schalter aktiviert die Anzeige von Node-IDs in der Cattree-Auswahl.

  Kompatibilität : Ist erst in Version 2 implementiert.

- <? gui window_height *Zahl* ?>

  Parameter : numerische Angabe, Höhe des Popup-Fensters in Pixeln

  Beschreibung : Mit dieser Anweisung legen Sie die Höhe des Popup-Fensters fest.

  Kompatibilität : Ist erst in Version 2 implementiert.

- <? gui window_width *Zahl* ?>

  Parameter : numerische Angabe, Breite des Popup-Fensters in Pixeln

  Beschreibung : Mit dieser Anweisung legen Sie die Breite des Popup-Fensters fest.

  Kompatibilität : Ist erst in Version 2 implementiert.

3.7.4 Prefilter-Anweisungen

Über die Prefilter-Anweisungen kann der Template-Entwickler steuern, aus welchem Bereich der Redakteur Dokumente auswählen kann. Diese Einstellung kann vom Redakteur nicht umgangen werden.

Wichtig:

Alle Vorfilter und Filterfelder werden logisch mit **UND** *verknüpft.*

- <? prefilter special category *String* ?>

  Parameter : *String* kann eine Imperia Node-ID, eine Rubriken-ID oder einen eindeutigen Rubrikennamen enthalten.

  Beschreibung : Mit diesem Vorfilter schränken Sie die angezeigten Dokumente auf einen Teil des Rubrikenbaumes ein.

  Kompatibilität : Ersetzt den Parameter CATEGORY=*String* aus Version 1.

- <? prefilter special docstatus *String* ?>

  Parameter : *String* kann die Buchstaben W, L, P enthalten.

  Beschreibung : Mit dem Docstatus-Vorfilter kann man die angezeigten Dokumente über ihren Status filtern.

  W : Es werden Dokumente angezeigt, die im Workflow sind.

  L : Es werden Dokumente angezeigt, die auf dem Live-System sind.

  P : Es werden Dokumente angezeigt, die aktuell in der Freischaltliste sind.

  Ist dieser Vorfilter nicht gesetzt, stehen alle Dokumente einschließlich des Archivs zur Auswahl.

  Kompatibilität : Ersetzt den Parameter DOCSTATUS=*String* aus Version 1.

- <? prefilter metafield key="*Metafeld*" value="*String*" ?>

  Parameter : *String* enthält den Text, der im *Metafeld* gefunden werden soll.

  Beschreibung : Über diesem Vorfilter können Sie die Dokumente auf den Inhalt eines beliebigen Metafeldes filtern.

  

  Hinweis:

  Diese Anweisung darf mehrfach vorkommen.

  Kompatibilität : Ersetzt die Zuweisung directory=*String* aus Version 1.

  Im Gegensatz zum alten Metatool wird beim aktuellen Metatool das Feld „directory“ nicht mehr speziell behandelt. Im alten Metatool war der Suchbegriff vorne verankert (`^suchwort`). Dies ist im aktuellen Metatool nicht mehr der Fall.

3.7.5 Filter-Anweisungen

Mit den Filter-Anweisungen kann der Redakteur die angezeigten Dokumente einschränken.

Wichtig:

Alle Vorfilter und Filterfelder werden logisch mit **UND** *verknüpft.*

- <? filter metafield *Metafeld* ?>

  Parameter : *Metafeld* Name des Metafeldes, das nach dem eingegebenen Suchbegriff durchsucht werden soll.

  Beschreibung : Durch diese Anweisung wird ein Eingabefeld für einen Suchbegriff angezeigt. Nach diesem Suchbegriff wird das über den Parameter definierte Metafeld durchsucht.

  

  Hinweis:

  Diese Anweisung darf mehrfach vorkommen.

  Kompatibilität : Ersetzt den Parameter SELFILTER=*Metafeld* aus Version 1.

- <? filter special author ?>

  Parameter : Ohne Parameter.

  Beschreibung : Hiermit aktivieren Sie die Anzeige einer Auswahlbox, die alle Benutzer enthält, auf die der angemeldete Redakteur Zugriff hat. Aus dieser Liste kann der Redakteur genau einen Benutzer auswählen.

  Die Anzeige enthält danach ausschließlich vom ausgewählten Benutzer angelegte Dokumente.

  Kompatibilität : Ist erst in Version 2 implementiert.

- <? filter special createdate ?>

  Parameter : Ohne Parameter.

  Beschreibung : Mit dieser Processing Instruction blenden Sie eine Auswahlbox mit den Werten „vor", „nach", „am" ein Eingabefeld für ein Datum im Format [YYYY-MM-DD] sowie eine Schaltfläche „..." zur Anzeige eines Kalenders ein.

  Nach der Auswahl eines Wertes aus der Auswahlbox und der Eingabe eines Datums werden nur noch Dokumente angezeigt, die dem eingestellten Datum entsprechen.

  Kompatibilität : Ist erst in Version 2 implementiert.

- <? filter special cattree ?>

  Parameter : Ohne Parameter.

  Beschreibung : Über diese Anweisung können Sie eine Auswahlbox mit dem Rubrikenbaum anzeigen lassen. Es kann genau ein Teilbaum ausgewählt werden.

  Nach der Auswahl eines Teilbaumes werden nur noch Dokumente innerhalb des Teilbaumes angezeigt.

  Die Anzeige des Rubrikenbaumes entspricht den Leserechten des aktuellen Benutzers im Rubrikenbaum. Ausnahme ist der Root-Eintrag, wenn dieser durch „Category" vorgegeben wurde. Dieser Eintrag muss immer in der Auswahlbox angezeigt werden, da es sonst nicht zu einer Einschränkung auf die Rubrik kommt.

  Kompatibilität : Ist erst in Version 2 implementiert.

- <? filter special fulltextsearch ?>

  Parameter : Ohne Parameter.

  Beschreibung : Es wird ein Eingabefeld für eine freie Suche über alle Metafelder angezeigt.

  Nach der Eingabe des Suchbegriffs werden nur noch Dokumente angezeigt, die diesen in einem beliebigen Metafeld enthalten.

  Kompatibilität : Ist erst in Version 2 implementiert.

3.7.6 Link-Anweisungen

Mit Link Anweisungen steuern Sie, wie und wohin Informationen von den ausgewählten Dokumenten in das aktuelle Dokument übertragen werden.

- <? link linklist element_count *Zahl* ?>

  Parameter : *Zahl* kann ein ganzzahliger Wert größer als 0 sein.

  Beschreibung : Diese Anweisung setzt die Anzahl der Link-Optionen im Metatool. Der Defaultwert ist 0.

  Kompatibilität : Ersetzt den Parameter DISPLAY_LISTELEMS=*Zahl* aus Version 1.

- <? link feature element_count *Zahl* ?>

  Parameter : *Zahl* kann ein ganzzahliger Wert größer als 0 sein.

  Beschreibung : Mit diese Anweisung legen Sie die Anzahl der Feature-Optionen im Metatool fest.

  Kompatibilität : Ersetzt den Parameter ANZMAINS=*Zahl* aus Version 1.

- <? link feature sync from="*Metafeld_from*" to="*Metafeld_to*" ?>

  Parameter : Mit *from* bezeichnen Sie den Namen des Quell-Metafeldes im ausgewählten Dokument, mit *to* den des Ziel-Metafeldes im aktuellen Dokument.

  Beschreibung : Durch Anweisung steuern Sie, welche Metainformationen von den ausgewählten Dokumenten in das aktuelle Dokument kopiert werden sollen.

  

  Hinweis:

  Diese Anweisung darf mehrfach vorkommen.

  Kompatibilität : Ersetzt den Parameter SELSYNC_*Metafeld_to*=*Metafeld_from* aus Version 1.

- <? link spfeature element_count *Zahl* ?>

  Parameter : *Zahl* kann ein ganzzahliger Wert größer als 0 sein.

  Beschreibung : Hiermit geben Sie die Anzahl der Special-Feature-Optionen im Metatool-Dialogfenster an. Special-Features (spfeature) sind Features (feature) bei denen der Name geändert werden kann.

  

  Hinweis:

  Diese Anweisung darf mehrfach vorkommen. Bei jedem Aufruf wird ein interner Zähler hochgezählt, so dass mehrere Special-Feature Einträge erstellt werden können.

  Wenn Sie mehrere Special-Feature Einträge erstellen wollen, müssen Sie diese blockweise anordnen.

```
<? link spfeature element_count 2 ?>
<? link spfeature name HUH ?>
<? link spfeature sync from="title" to="Btext" ?>

<? link spfeature element_count 4 ?>
<? link spfeature name UHU ?>
<? link spfeature sync from="title" to="Btext" ?>
```

Beispiel 3.1. Mehrere Special-Feature Einträge, blockweise angeordnet

  Kompatibilität : Ersetzt den Parameter FEATURE1_ANZMAINS=*Zahl* aus Version 1.

- <? link spfeature name *String* ?>

  Parameter : *String* enthält den Namen des Special-Features.

  Beschreibung : Mit dieser Anweisung setzen Sie den Namen des Special-Features.

  

  Hinweis:

  Diese Anweisung darf mehrfach vorkommen. s. o.

  Kompatibilität : Ersetzt den Parameter FEATURE1_name=*String* aus Version 1.

- <? link spfeature sync from="*Metafeld*" to="*Metafeld*" ?>

  Parameter : Mit *from* bezeichnen Sie das Quell-Metafeld im ausgewählten Dokument, mit *to* das Ziel-Metafeld im aktuellen Dokument.

  Beschreibung : Mit dieser Anweisung legen Sie fest, welche Metainformationen aus dem ausgewählten Dokument an welcher Stelle in das aktuelle Dokument kopiert werden.

  

  Hinweis:

  Diese Anweisung darf mehrfach vorkommen. s. o.

Kompatibilität : Ersetzt den Parameter FEATURE1_sync_*Metafeld_to=Metafeld_from* aus Version 1.

- <? link imperiablock *String* ?>

  Parameter : *String* enthält den Index des zu füllenden Blocks.

  Beschreibung : Mit dieser Anweisung verändern Sie den Zugriff auf Metafelder im aktuellen Dokument. Der Metafeldname setzt sich aus metafeld_*String*_nummer zusammen.

  Diese Anweisung benötigen Sie bei der Verwendung von Imperiablöcken oder Flexmodulen (siehe Beispiele).

  Kompatibilität : Ersetzt den Parameter IMPERIABLOCK=*String* aus Version 1.

- <? link imperiasuffix *String* ?>

  Parameter : *String* enthält ein Suffix, das an den Metafeldnamen angehängt wird.

  Beschreibung : Mit dieser Anweisung können Sie den Zugriff auf Metafelder im aktuellen Dokument verändern. Der Metafeldname ergibt sich aus metafeld_nummer_*String*.

  Diese Anweisung brauchen Sie, um das Metatool in einem Flexmodul aufzurufen (siehe Beispiele).

  Kompatibilität : Ersetzt den Parameter IMPERIASUFFIX=*String* aus Version 1.

3.8 Der Templateprozessor

Bei jeder Bearbeitung eines Dokuments im Edit-Schritt wird der Templateprozessor aufgerufen. Er hat die Aufgabe, alle im Template eingeführten Funktionen auszuwerten. Dazu muss er für jedes Template prüfen, welche der möglichen Module tatsächlich darin vorkommen.

Die einzelnen Module werden in einer festen Reihenfolge, der so genannten Template-Chain abgearbeitet. Diese Reihenfolge können Sie über die Systemkonfigurationsvariable „TEMPLATE_CHAIN“ in der Datei `system.conf` beeinflussen.

Warnung:

Ist diese Variable gesetzt, so werden **alle** *Dokumente nur noch mit den darin aufgeführten Modulen ausgewertet. Auf diese Weise vorgenommene Änderungen sind also global für Ihr System gültig.*

Wenn Sie die Template-Chain nicht für Ihr ganzes System ändern möchten, können Sie mit der Metavariablen `__imperia_template_chain` für jede Rubrik bzw. jedes Dokument steuern, welche Template-Chain jeweils gilt.

Wenn Sie die Variable nicht setzen, wird folgender Default verwendet:

```
DynamicConf,CleanActive,CodeInclude,Flex,ImperiaBlocks,Array,CleanTemplate,
Control,Localization,Expansion,FormFill,MediaDatabase,BlackMan,LiveEdit,
JavaScript,DynamicConf,Generator,Convert
```

Hinweis:

Der obenstehende Text wurde für die Druckdarstellung umbrochen.

3.8.1 Module deaktivieren

Wenn Sie genau wissen, dass Sie bestimmte Module in Ihrem Projekt nicht verwenden, können Sie den Templateprozessor zur Steigerung der Performance Ihres Systems so umkonfigurieren, dass nur noch die benötigten Module ausgeführt werden.

Warnung:

Die Deaktivierung unbedingt erforderlicher Module kann zu unerwarteten Ergebnissen bei der Verarbeitung von Templates führen. Lesen Sie Abschnitt 3.8.3 **Die Module des Templateprozessors** *auf Seite 89 für Informationen über die einzelnen Module. Deaktivieren Sie nur Module, bei denen Sie sicher sind, dass sie nicht benötigt werden.*

Wenn Sie beispielsweise in keinem Dokument Imperia-Blöcke, Array-Blöcke oder eine *DynamicConf* benötigen und das Bearbeiten auf dem Ziel-System per *LiveEdit* und nicht per *BlackMan* vornehmen möchten, entfernen Sie die entsprechenden Module aus der Liste. Tragen Sie dazu die Variable `TEMPLATE_CHAIN` wie im folgenden Beispiel gezeigt in der Datei `system.conf` ein:

```
"TEMPLATE_CHAIN" = "CleanActive,CodeInclude,Flex,CleanTemplate,Control,Localization,
  Expansion,FormFill,MediaDatabase,LiveEdit,JavaScript,Generator,Convert"
```

Hinweis:

Der oben stehende Text wurde für die Druckdarstellung umgebrochen.

3.8.2 Änderung der Modulreihenfolge

Die vorgegebene Reihenfolge, in der die einzelnen Module des Templateprozessors ausgeführt werden, macht die Verwendung bestimmter Konstrukte unmöglich.

Abhilfe können Sie in solchen Fällen durch eine Änderung der Template-Chain schaffen. Dazu ein Beispiel:

Sie wollen zwei alternative Codeincludes verwenden, die abhängig von dem Verzeichnis geladen werden, in dem sich das betreffende Dokument befindet: die Dokumente befinden sich in den Verzeichnissen `mitteilungen/news` und `mitteilungen/archiv`. Ihre Codeincludes haben Sie den Verzeichnissen entsprechend benannt:

```
mitteilungen_news.htms
mitteilungen_archiv.htms
```

Im Template können Sie den Dateinamen des Codeincludes mit folgendem Konstrukt zusammenstellen:

```
<!--CODEINCLUDE:<!--dirlevel:1-->_<!--dirlevel:2-->.htms-->
```

Mit der Default-Template-Chain funktioniert dieses Konstrukt leider nicht. Anstatt des Inhaltes des Codeincludes erscheint im Dokument lediglich ein Kommentar, beispielsweise:

```
<!--CODEINCLUDE:mitteilungen_news.htms-->
```

Der Grund hierfür ist, dass die Expansion der Imperia-Variablen nach der Ausführung der Codeincludes stattfindet. Wenn die Include-Datei in das Template geladen wird, ist der Inhalt der Variablen `dirlevel:1` und `dirlevel:2` also noch nicht bekannt, so dass der Dateiname nicht gebildet werden kann.

Die Default-Reihenfolge für die Ausführung der Module des Templateprozessors sieht folgendermaßen aus:

```
DynamicConf,CleanActive,CodeInclude,Flex,ImperiaBlocks,Array,CleanTemplate,
Control,Localization,Expansion,FormFill,MediaDatabase,BlackMan,LiveEdit,
JavaScript,DynamicConf,Generator,Convert
```

Hinweis:

Der oben stehende Text wurde für die Druckdarstellung umgebrochen.

Das Modul *CodeInclude* ist hierbei für die Ausführung der Include-Dateien zuständig, und die Auswertung der Imperia-Variablen wird in dem Modul *Expansion* durchgeführt. Der Templateprozessor arbeitet die einzelnen Module von links nach rechts in der Reihenfolge ab, in der Sie in der Template-Chain aufgeführt sind.

Damit das oben stehende Konstrukt zur Bildung des Codeinclude-Dateinamens funktioniert, müssen Sie die Reihenfolge der Module also ändern bzw. ergänzen.

Warnung:

Änderungen in der Reihenfolge der Ausführung unbedingt notwendiger Module können zu unerwarteten Ergebnissen bei der Verarbeitung von Templates führen. Lesen Sie Abschnitt 3.8.3 **Die Module des Templateprozessors** *auf Seite 89 für Informationen über die einzelnen Module.*

Neben dem Codeinclude selbst können auch etwaige Imperia- oder Array-Blöcke sowie Flexmodule auf Imperia Variablen zugreifen. Wenn Sie das Modul *Expansion* also in der Reihenfolge vor *CodeInclude* "verschieben", werden Sie wahrscheinlich Fehler bei der weiteren Verarbeitung des Templates erhalten.

Fügen Sie stattdessen einen weiteren Aufruf von *Expansion* in die Template-Chain ein, so dass diese wie folgt aussieht:

```
"TEMPLATE_CHAIN" = "DynamicConf,CleanActive,Expansion,CodeInclude,Flex,
ImperiaBlocks,Array,CleanTemplate,Control,Localization,Expansion,FormFill,
MediaDatabase,BlackMan,LiveEdit,JavaScript,DynamicConf,Generator,Convert"
```

Hinweis:

Der oben stehende Text wurde für die Druckdarstellung umgebrochen.

Hinweis:

Die mehrmalige Ausführung von Modulen des Templateprozessors erzeugt auch mehr Last auf dem System. Diese zusätzliche Last könne Sie vermindern, wenn Sie die entsprechenden Änderungen auf die Dokumente beschränken, bei denen dies notwendig ist.

3.8.3 Die Module des Templateprozessors

Im Folgenden finden Sie eine Aufzählung der Module, die der Templateprozessor nutzt, mit einer kurzen Erklärung.

Array

In diesem Modul findet die Verarbeitung von Array-Blöcken statt (siehe dazu auch Abschnitt 3.3.12 **Arrayblocks** auf Seite 36).

CodeInclude

In diesem Modul erfolgt die Auswertung von Codeincludes, Slots und Flexmodulen.

Control

Das Modul wertet im Template Code enthaltene Kontrollstrukturen wie *#IF*, *#ELSE* und *#ENDIF* aus (siehe dazu auch Abschnitt 3.4.15 **IF-Abfragen** auf Seite 67).

Convert

Dieses Modul steuert die Konvertierung von Textdaten im Template (siehe dazu auch Abschnitt 3.3.15 **Convert-Plug-Ins (postconvert)** auf Seite 49).

Dynamic

Dieses Modul nimmt die Ersetzungen vor, die Sie in der Datei `dynamic.conf` festgelegt haben (siehe dazu auch Abschnitt 3.5 **Dynamische Module** auf Seite 73).

Expansion

Die Expandierung von Variablen erfolgt in diesem Modul (siehe dazu auch Abschnitt 3.4 **Funktionen und Konstanten in Templates** auf Seite 53).

Flex

Das Modul verarbeitet im Template enthaltene Flexmodule (siehe dazu auch Abschnitt 3.3.16 **Flexmodule** auf Seite 51und Kapitel 4 **Flexmodule und Slots**).

FormFill

Dieses Modul dient zur Verarbeitung von Formularen und Eingabefeldern.

ImperiaBlocks

Dieses Modul verarbeitet die Imperia-Blöcke im Template (siehe dazu auch Abschnitt 3.6 **Imperiablocks** auf Seite 77).

JavaScript

Dieses Modul stellt eine JavaScript-Bibliothek für den Edit-Modus bereit.

Localization

Dieses Modul steuert die Lokalisierung von Metadaten und Template Code.

MediaDatabase

Dieses Modul ruft die Mediendatenbank auf und stellt eine minimale Kompatibilitätsschicht zur Media-Asset-Management von Imperia 5 bereit (siehe dazu auch Abschnitt 3.3.14 **Media-Asset-Management und Grafik-Upload** auf Seite 39).

ClearTemplate

Initialisierung des Templates.

ClearActive

Initialisierung des Templates.

JWord

Dieses Modul konvertiert die speziellen Anweisungen des Java-Wordmoduls zu Object- und Embed-Tags.

AXWord

Modul für die Darstellung des ActiveX-Wordmoduls.

EWE

Dieses Modul expandiert das EWE-Wordmodul im Bearbeiten-Schritt.

Hinweis:

In jedem der oben genannten Schritte, expandiert der Templateprozessor außerdem mehrsprachige Inhalte.

3.9 Templates für die Imperia-Volltextsuche

Für die Eingabe von Suchanfragen und die Ausgabe der Suchergebnisse verwendet die Imperia-Volltextsuche Templates. Diese werden unterhalb der DOCUMENT-ROOT Ihres Zielsystems abgelegt. Sie können Templates für die einfache und die verfeinerte Eingabe von Suchbegriffen sowie für die Darstellung der Suchergebnisse definieren. Zusätzlich haben Sie die Möglichkeit, jeweils mehrere Alternativen anzugeben, was einen schnellen Wechsel zwischen unterschiedlichen Layouts erlaubt. Die Auswahl der Varianten geschieht über einen Parameter, den Sie dem Suchskript beim Aufruf mitgeben. Lesen Sie hierzu auch Abschnitt 3.9.2 **Übergabeparameter und Variablen in Such-Templates** auf Seite 92.

Im Verzeichnis `site/fts/templates` finden Sie Beispieldateien für eine einfache und eine verfeinerte Suchmaske und für eine Ergebnisseite. Die Volltextsuche nutzt diese Templates automatisch, wenn Sie das Suchskript wie folgt aufrufen:

http://ihrserver.net/cgi-bin/fts_search.pl

Mit diesem Aufruf nutzen Sie die einfache Suchmaske ohne zusätzliche Optionen. Eine verfeinerte Suchmaske mit zusätzlichen Suchoptionen bekommen Sie, wenn Sie folgende URL aufrufen:

http://ihrserver.net/cgi-bin/fts_search.pl?ADV=1

Diese Beispieltemplates sind gleichzeitig die Default-Suchmasken, die zum Einsatz kommen, wenn Sie die Suche ohne eigene Konfiguration nutzen. Lesen Sie hierzu auch die Schnellstartanleitung im Kapitel **Die Imperia Volltextsuche** im Administrationshandbuch. Mehr zur Entwicklung eigener Suchtemplates lesen Sie im folgenden Abschnitt.

Geben Sie nun einen Suchbegriff ein und schicken Sie die Anfrage ab. Die Imperia Volltextsuche bearbeitet diese dann und liefert das Suchergebnis als Trefferliste zurück. Das Aussehen dieser Trefferliste steuern Sie wiederum über ein Template. Auch hierfür gibt es ein Beispieltemplate, das automatisch ausgewählt wird, wenn Sie die Default-Einstellungen verwenden. Informationen über die Erstellung eigener Ergebnistemplates finden Sie im Abschnitt 3.9.3 **Ergebnis-Templates** auf Seite 95.

3.9.1 Such-Templates

Das Such-Template enthält mindestens ein HTML - Eingabefeld zur Eingabe des Suchbegriffs sowie einen Button zum Abschicken der Anfrage. Diese Bedienelemente können Sie mit einem Formular in ein Dokument einbinden. Tragen Sie im action-Attribut des Formulars die URL des Suchskripts ein:

```
<form action="/cgi-bin/site_search.pl" method="post">
```

Etwaige Aufrufparameter hängen Sie als Query-String an den Aufruf des Suchskripts an. Diese Variante empfiehlt sich, wenn Sie ein Eingabefeld für Suchanfragen als Standard-Bedienelement auf allen Seiten Ihres Projekts anbieten wollen. Mehr Flexibilität, zum Beispiel bezüglich der Auswahl alternativer Layout-Varianten oder der Verfeinerung der Suchanfragen haben Sie, wenn Sie die Suchmaske als eigenes Template bereitstellen und dieses nicht direkt verlinken, sondern über die Volltextsuche parsen lassen.

Einige Parameter, beispielsweise die URL des Suchskripts oder die Layoutvariante für das Ergebnistemplate, können Sie dann als Variable im Suchtemplate referenzieren. Der Aufruf des Suchtemplates geschieht dann über das Suchskript im Verzeichnis `cgi-bin`. Das Skript lädt dann die eigentliche Suchmaske und füllt die Variablen im Template mit den entsprechenden Werten.

Ein Beispiel für einen solchen Einsatz von Template-Variablen ist das action-Attribut des Suchformulars:

```
<form action="[!--FORM_ACTION--]" method="post">
```

Das Suchskript füllt diese Variable mit dem entsprechenden Wert, in diesem Fall ein Verweis auf sich selbst, so dass der Anwender eine HTML-Seite bekommt, deren Quellcode dem hartcodierten Beispiel am Anfang dieses Abschnitts entspricht.

Ein Beispiel für ein einfaches Suchtemplate:

```
<html>
  <head>
    <title>Imperia-Volltextsuche</title>
  </head>
  <body>
    <form action="/cgi-bin/[!--FORM_ACTION--]" method="post">
      Bitte geben Sie einen Suchbegriff ein:<br />
      <input name="SEARCH"><br />
      <input type="submit">
    </form>
  </body>
</html>
```

Eine Erklärung der verfügbaren Template-Variablen und Übergabeparameter finden Sie im nächsten Abschnitt.

3.9.2 Übergabeparameter und Variablen in Such-Templates

Neben dem Suchbegriff können Sie weitere Parameter über Eingabefelder im Suchformular steuern. Diese werden dann beim Abschicken des Formulars an das Suchskript geschickt. Alternativ können Sie diese Parameter auch in Verweisen auf das Suchskript als Query-String an die URL anhängen. Folgende Template-Variablen bzw. Aufrufparameter stehen zur Verfügung:

SEARCH

Der Parameter mit dem Suchbegriff hat den Namen `"SEARCH"`.

Beispiel:

```
<input type="text" name="SEARCH" value="">
```

GROUP

Mit diesem Parameter übergeben Sie Gruppen, über die eine Suchanfrage gefiltert werden soll. Im Value-Wert muss mindestens einer der deklarierten Gruppennamen stehen. Die Gruppen für die Filterung definieren Sie in der Konfiguration der Indexerstellung. Lesen Sie hierzu den Abschnitt **Die Konfiguration der Indexerstellung (index.conf)** im Kapitel **Die Imperia Volltextsuche** im Administrationshandbuch.

Die Filterung über den Parameter *GROUP* deaktivieren Sie, indem Sie *0* als Wert übergeben, oder wenn Sie den Parameter weglassen.

Für die Auswahl der Gruppen im Suchformular können Sie Select-, Checkbox- oder Radio-Felder verwenden. Mit Multiselect-Feldern sind auch Mehrfachauswahlen möglich.

Beispiel:

```
<input type="checkbox" name="GROUP" value="DVD">
<input type="checkbox" name="GROUP" value="CD">
```

Wenn Sie eine Selectbox für die Auswahl der Gruppen verwenden, können Sie diese im Suchtemplate automatisch mit Optionen für vorhandene Gruppen befüllen lassen. Voraussetzung dafür ist, dass Sie beim name-Attribut des Select-Felds doppelte Anführungszeichen setzen und die automatisch einzutragenden Gruppen in der Datei `site/config/fts.conf` als `templateDynamicGroups` definieren. Lesen Sie hierzu den Abschnitt **Konfiguration der Suche (fts.conf)** im Administrationshandbuch.

DATE

Einschränkung der Suche auf die letzten xx Tage.

Beispiel:

```
DATE = 0 : alle Dokumente von heute
DATE = 7 : alle Dokumente der letzten 7 Tage
```

Die Auswahl von verschiedenen Zeitbereichen kann beliebig über Checkboxen oder Select-Felder erfolgen.

```
<select name="DATE">
  <option value="0">heute</option>
  <option value="7">letzten 7 Tage</option>
</select>
```

HTML_NR

Mit diesem Parameter bestimmen Sie, welche Templatevariante verwendet werden soll. Als Wert übergeben Sie die jeweilige Zahl, die Sie in der Datei `site/config/fts.conf` bei den Template-Steuerungsvariablen definiert haben.In Templates setzen Sie diesen Parameter am besten mit versteckten Formularfeldern.

Beispiel:

```
<input type="hidden" name="HTML_NR" value="3">
```

ADV

Setzen Sie diesen Parameter auf 1, um das Template für die verfeinerte Suchanfrage aufzurufen.

Beispiel:

```
<a href="[!--FORM_ACTION--]?ADV=1">Erweiterte Suche</a>
```

DATE_START

Sie können Sie den DATE-Filter auch auf Datumsbereiche eingrenzen. Mit diesem Parameter legen Sie dabei das Startdatum fest. Relevant ist das letzte Änderungsdatum eines Dokumentes.

Beispiel:

```
<input type="text" name="START_DATE" value="">
```

DATE_END

Mit diesem Parameter setzen Sie das Enddatum für den Datumsbereich beim Filtern der Dokumente nach dem Änderungsdatum.

Beispiel:

```
<input type="text" name="DATE_END" value="">
```

WCOUNT_MIN / WCOUNT

Nutzen Sie diesen Parameter, wenn Sie in der Trefferliste nur Dokumente mit einer Mindestanzahl von Wörtern anzeigen lassen wollen. Der Parameter *WCOUNT* ist synonym mit *WCOUNT_MIN*.

Beispiel:

```
<input type="text" name="WCOUNT_MIN" value="">
<input type="text" name="WCOUNT" value="">
```

WCOUNT_MAX

Dieser Parameter ermöglicht es Ihnen, Dokumente, die eine Höchstanzahl von Wörtern überschreiten, aus der Trefferliste auszufiltern.

Beispiel:

```
<input type="text" name="WCOUNT_MAX" value="">
```

LANG

Geben Sie eine Sprachvariante an, um bei der Auswertung des Suchbegriffs sprachspezifische Ersetzungen zu aktivieren. Diese sprachspezifischen Ersetzungen definieren Sie in der Suchkonfiguration.

Beispiel:

```
<input type="hidden" name="LANG" value="DE">
```

INPUT_LOCALE

Legen Sie einen Zeichensatz fest, in dem Eingaben in das Suchformular erfolgen. Diese Methode hilft Ihnen bei der Behebung von Problemen mit eingegebenen sprachspezifischen Sonderzeichen.

Beispiel:

```
<input type="hidden" name="INPUT_LOCALE" value="ISO-8859-1">
```

OUTPUT_LOCALE

Legen Sie einen Zeichensatz fest, in dem die Ausgabe des Suchergebnisses erfolgen soll. Diese Methode hilft Ihnen bei der Behebung von Problemen mit sprachspezifischen Sonderzeichen.

Beispiel:

```
<input type="hidden" name="OUTPUT_LOCALE" value="ISO-8859-1">
```

SORT

Mit diesem Parameter bestimmen Sie die Sortierreihenfolge der Trefferliste. Folgende Werte sind möglich:

Parameter	Erklärung
arelevance	Aufsteigende Sortierung nach Relevanz.
drelevance	Absteigende Sortierung nach Relevanz. (Default)
amodified	Aufsteigende Sortierung nach Änderungsdatum.
dmodified	Absteigende Sortierung nach Änderungsdatum.
asize	Aufsteigende Sortierung nach Dateigröße.
dsize	Absteigende Sortierung nach Dateigröße.
awords	Aufsteigende Sortierung nach Wortanzahl.
dwords	Absteigende Sortierung nach Wortanzahl.

Tabelle 3.14. Werte für den Parameter SORT

Hinweis:

Grundsätzlich findet immer zunächst eine absteigende Sortierung der Treffer nach Relevanz statt. Etwaige Sortieroptionen vom Benutzer oder aus der Konfiguration beeinflussen diese vorsortierte Liste.

TYPE

Nutzen Sie diesen Parameter, um die Suche auf bestimmte Dokumenttypen einzuschränken. Die folgende Tabelle zeigt die möglichen Werte für den Übergabeparameter. Mehrfachnennungen sind möglich.

Parameter	Erklärung
HTML	Nur HTML-Dateien durchsuchen.
TXT	Nur Textdateien durchsuchen.
XPDF	Nur PDF-Dateien durchsuchen.
DOC	Nur Microsoft Word-Dateien durchsuchen.
XLS	Nur Microsoft Excel-Dateien durchsuchen.
PPT	Nur Microsoft Powerpoint-Dateien durchsuchen.

Tabelle 3.15. Werte für den Parameter TYPE

3.9.3 Ergebnis-Templates

Neben den Bedienelementen für weitere Suchanfragen enthält das Ergebnistemplate die HTML-Ausgabe des Suchergebnisses. Diese Ausgabe können Sie durch eine Reihe von speziellen Template-Befehlen steuern.

Die Befehle werden durch HTML-Kommentarzeichen <!-- --> oder eckige Klammern [!-- --] eingeschlossen. Beide Schreibweisen haben die gleiche Bedeutung.

Bei WYSIWYG-Editoren ist es jedoch von Vorteil, sichtbare und unsichtbare Befehle eingeben zu können. Wenn Sie eckige Klammern verwenden, bleiben die Template-Befehle in der Vorschau des Editors sichtbar, während HTML-Kommentare ausgeblendet werden. Dafür verschiebt ein WYSIWYG-Editor letztere nicht, z.B. zwischen Tabellenreihen im HTML-Code. Welche Klammerungsart Sie verwenden, spielt für die Suche keine Rolle.

Beispiel für ein einfaches Ergebnis-Template (Template-Befehle zur besseren Unterscheidung in eckigen Klammern):

```
<html>
  <head>
    <title> Ergebnis der Imperia-Volltextsuche</title>
  </head>
  <body>

  <!-- Abschnitt für die erneute Suche -->
```

```
    <form action="[!--FORM_ACTION--]" method="post">
      Suche nach :<br />
      <input name="SEARCH" value="[!--SEARCH_REQUESTED--]"><br />
      <input type="submit">
    </form>


  <!-- Abschnitt für die Resultate -->

  [!--SEARCH_WORD_RESULT--]
  "[!--SEARCH_REQUESTED--]" in [!--RESULT_TOTAL--] Dokumenten gefunden. <br />
  [!--/SEARCH_WORD_RESULT--]


  [!--SEARCH_WORD_NOTFOUND--]
  Das Wort "[!--SEARCH_REQUESTED--]" wurde nicht gefunden. <br />
  [!--/SEARCH_WORD_NOTFOUND--]


  [<!--SEARCH_PAGE_RESULT--]
  [!--RESULT_NR--]. <a href="[!--PAGE_URL--]">Filename: [!--PAGE_URL--]</a><br />
  [!--/SEARCH_PAGE_RESULT--]
</body>
</html>
```

In den folgenden Abschnitten finden Sie Erläuterungen zu den in diesem Beispiel verwendeten Template-Befehlen.

3.9.3.1 Suchbegriffsbefehle

SEARCH_REQUESTED

Dies ist das Feld zur Aufnahme des Suchbegriffs, den der Benutzer bei seiner Suchanfrage eingegeben hatte. Dies ist sinnvoll, um bei einem Input-Feld den Inhalt vorzudefinieren.

Beispiel:

```
<input type="text" name="SEARCH" value="[!---SEARCH_REQUESTED--]">
```

SEARCH_WORD_RESULT

Mit diesem Befehl definieren Sie einen Block, indem Sie weitere Befehle zur Ausgabe von Suchtreffern notieren können.

Syntax:

```
[!--SEARCH_WORD_RESULT--]
  [weitere Termplate-Befehle]
[!--/SEARCH_WORD_RESULT--]
```

Folgende Befehle stehen Ihnen zur Verfügung:

`WORD_SEARCH_REQUEST_FOUND` - Diese Anweisung generiert in der Ergebnisseite für jedes im Suchbegriff enthaltene Wort einen Verweis auf eine Detailsuche mit dem jeweiligen Wort als Suchbegriff.

Beispiel:

Angenommen der Suchbegriff lautet "Imperia Volltextsuche". In der Ergebnisseite erscheinen dann folgende Links:

```
<a href="fts_search.pl?search=Imperia">Imperia</a><br />
<a href="fts_search.pl?search=Volltextsuche">Volltextsuche</a>
```

WORD_FOUND - Dieser Befehl gibt den Suchbegriff aus.

WORD_COUNT_FOUND - Gibt die Anzahl der Dokumente aus, in denen der Suchbegriff gefunden wurde.

META_KEY - Dies ist der Name des Metafeldes, in dem der Suchbegriff gefunden wurde.

META_DETAILS...../META_DETAILS - Innerhalb dieses Blocks notierte Anweisungen werden nur auf Meta-Informationen angewendet. Folgende Anweisungen stehen Ihnen hierfür zur Verfügung:

META_WORD_COUNT_FOUND - Verhält sich wie WORD_COUNT_FOUND, gilt aber für Meta-Informationen.

META_WORD_FOUND - Verhält sich wie WORD_FOUND, gilt aber für Meta-Informationen.

Die Ergebnisse werden jeweils für jedes Wort aus dem Suchbegriff angezeigt.

Hinweis:

Das Suffix _FOUND können Sie optional auch weglassen. Das ändert das Verhalten des jeweiligen Template-Befehls dahingehend, dass nun auch Treffer angezeigt werden, die sonst wegen etwaigen Filtern, z.B. auf ein bestimmtes Erstellungsdatum, Dateigrößen oder Gruppenzugehörigkeiten, ausgeblendet werden.

SEARCH_WORD_NOTFOUND

Hierbei handelt es sich um ein Template-Befehl zur Ausgabe von Teilen eines Suchbegriffs oder ganzen Suchbegriffen bei fehlgeschlagenen Anfragen.

Syntax:

```
[!--SEARCH_WORD_NOTFOUND--]
  [weitere Template-Befehle]
[!--/SEARCH_WORD_NOTFOUND--]
```

In diesem Block notieren Sie folgenden Befehl:

NOTFOUND_WORD = Nicht gefundenes Wort

3.9.3.2 Stopwörter in Ergebnis-Templates anzeigen lassen

Anfragen, die nur Stopwörter enthalten, liefern keinerlei Treffer. Enthält eine Anfrage neben Stopwörtern weitere Suchwörter, liefert die Volltextsuche alle Dokumente, die diese Wörter enthalten. Das schließt auch Dokumente ein, in denen nur die anderen Suchwörter vorkommen, nicht aber die Stopwörter. Um das Suchergebnis in so einem Fall für den Benutzer transparenter zu machen, lassen sich in der Anfrage vorkommende Stopwörter gesondert in einem eigenen Bereich des Ergebnis-Templates anzeigen.

SEARCH_WORD_BLACKLISTED

Mit diesem Befehl schließen Sie den Block zur Behandlung von Stopwörtern ein. In dem Block notieren Sie weitere Template-Befehle und Markup.

Syntax:

```
[!--SEARCH_WORD_BLACKLISTED--]
  [weitere Template-Befehle]
[!--/SEARCH_WORD_BLACKLISTED--]
```

BLACKLISTED_WORD

Dieser Befehl ist ein Platzhalter. An seiner Stelle erscheinen bei der Darstellung des Suchergebnisses etwaige Stopwörter aus der Suchanfrage.

Ein Block zur Anzeige von Stopwörtern im Ergebnis-Template könnte beispielsweise wie folgt aussehen:

```
[!--SEARCH_WORD_BLACKLISTED--]
  Im Suchbegriff enthaltene Stopwörter:<br />
  <b>"[!--BLACKLISTED_WORD--]"</b></br>
[!--/SEARCH_WORD_BLACKLISTED--]
```

3.9.3.3 Befehle zur Fehlerbehandlung

Für Fehler, die bei der Verarbeitung von Suchanfragen auftreten, können Sie im Ergebnis-Template die formatierte Ausgabe von Fehlermeldungen definieren.

ERROR

In diesem Block formatieren Sie die Ausgabe einer Fehlermeldung. Die Meldung selbst referenzieren Sie mit dem Befehl `[!--ERROR_MESSAGE--]`.

Beispiel:

```
[!--ERROR--]
<tr>
  <td colspan="3">
    <p>Folgende Fehler sind aufgetreten;</p>
    <b>[!--ERROR_MESSAGE--]</b>
  </td>
</tr>
[!--/ERROR--]
```

Warnungen und nicht kritische Fehlermeldungen werden in diesem Block ausgegeben.

FATAL_ERROR

Bei kritischen Fehlern, die zum Abbruch der Suche geführt haben, wird der Inhalt dieses Blocks ausgegeben.

Beispiel:

```
[!--FATAL_ERROR--]

<tr>
  <td colspan="3">
    <p>FATAL Error occurred. For more information check the error log.</p>
  </td>
</tr>

[!--/FATAL_ERROR--]
```

QUERY_ERROR

Mit diesem Befehl steuern Sie, wo und wie Fehler bei der Eingabe des Suchbegriffs angezeigt werden.

Beispiel:

```
<tr>
  <td colspan="3">
    <p><!--QUERY_ERROR--></p>
```

```
  </td>
</tr>
```

3.9.3.4 Befehle zur Behandlung gefundener Dokumente

Mit den folgenden Befehlen steuern Sie die Ausgabe der Informationen über die gefundenen Dokumente.

SEARCH_PAGE_RESULT

Alle Befehle zur Steuerung der ausgegebenen Informationen über gefundene Dokumente notieren Sie innerhalb dieses Blocks.

Syntax:

```
[!--SEARCH_PAGE_RESULT--]
  [weitere Termplate-Befehle]
[!--/SEARCH_PAGE_RESULT--]
```

Für die Ausgabe von Informationen über Suchtreffer stehen Ihnen folgende Template-Befehle zur Verfügung:

`RESULT_NR` = Laufende Nummer des Ergebnisses.

`PAGE_RANK` = Die Gewichtung des Ergebnisses. Es handelt sich um einen abstrakten Wert, der anhand der Häufigkeit des Suchbegriffs im betreffenden Dokument ermittelt wird.

`PAGE_PERCENT` = Gewichtung dieses Ergebnisses. Es handelt sich nicht um einen Prozentwert! Dieser Befehl wurde aus Kompatibilitätsgründen zur alten Volltextsuche übernommen. Der angezeigte Wert entspricht dem Ergebnis, das Sie mit dem Befehl PAGE_RANK bekommen.

`PAGE_WORDS` = Anzahl der Wörter im Dokument.

`PAGE_FOUND_WORDS` = Eine Liste der Begriffe, die im betreffenden Dokument gefunden wurden.

`PAGE_FILE` = Der Dateiname des Dokuments.

`PAGE_TYPE` = Beliebiger Text, der den Typ des gefundenen Dokuments angibt. Diese Angabe stammt vom jeweils verarbeitenden Parser. Beispiel: '*Portable Document Format (PDF)*' oder '*HTML*'.

`PAGE_SIZE` = Dokumentengröße in Kilobytes.

`PAGE_SIZE_BYTE` = Dokumentengröße in Bytes.

`PAGE_URL` = Pfadangabe und Dateiname des Dokuments.

`PAGE_DATE` = Änderungsdatum des Dokuments.

`PAGE_TIME` = Uhrzeit der letzten Änderung des Dokuments.

`PAGE_SIZE` = Größe des Dokuments in Kilobyte.

`PAGE_META:XXX` = Ausgabe der Textbausteine, die während der Indizierung aus dem Dokument extrahiert wurden.

`XXX` = Textbaustein Name

Optional können Sie noch festlegen, welche Länge der Textbaustein haben muss und auf welche Länge dieser reduziert werden soll. Dies erreichen Sie, indem nach dem Textbausteinnamen ein `"=CHAR(MIN=YYY;MAX=YYY;CUT=YYY)"` gesetzt wird. Anstelle von `"CHAR"` kann auch `"WORD"` gesetzt werden.

Bei `"CHAR"` beziehen sich die Angaben von `MIN, MAX` und `CUT` auf Buchstaben, bei `"WORD"` auf Wörter. `MIN, MAX` und `CUT` können je nach Bedarf gesetzt werden. `MIN` legt fest, wie lang der Textbaustein mindestens sein muss, damit dieser verwendet wird. `MAX` gibt die maximale Länge des Textbausteins an. Mit `CUT` kann der Textbaustein auf eine beliebige Länge festgelegt werden, der Rest wird abgeschnitten und nicht angezeigt.

Es können auch mehrere Textbausteinnamen hintereinander, getrennt durch ein `"||"`, angegeben werden, so dass der erste Textbaustein nicht angezeigt wird, weil dieser z.B. zu wenig Wörter enthält, der nächste getestet wird usw.

Beispiel:

Wenn der Titel mindestens ein Zeichen enthält, wird dieser ausgegeben, ansonsten soll der Text „Kein Titel“ dargestellt werden.

```
[!--PAGE_META:TITLE=CHAR(MIN=1) || "Kein Titel"--]
```

Im folgenden Beispiel soll kein fester Text, sondern abhängig vom Vorhandensein des Wertes „formular_url“ entweder dieser Wert, oder andernfalls die URL des Dokuments selbst ausgegeben werden:

```
[!--PAGE_META:formular_url=CHAR(MIN=2) || "<!--PAGE_URL-->"--]
```

Es können auch zwei oder mehr Variablen geprüft werden:

```
[!--PAGE_META:name=CHAR(MIN=2) || title=CHAR(MIN=2) || "- kein Titel -"--]
```

`CURRENT_PAGE` = Aktuelle Ergebnisseite

`RESULT_BEGIN` = Beginn der laufenden Ergebnisnummer für die aktuelle Seite

`RESULT_END` = Ende der laufenden Ergebnisnummer für die aktuelle Seite

`RESULT_TOTAL` = Gesamtzahl der gefundenen Dokumente

3.9.3.5 Anzeige mehrerer Ergebnisseiten

Füllt das Suchergebnis mehrere Bildschirmseiten, stellt die Volltextsuche entsprechend der Konfiguration eine Möglichkeit zur Navigation zwischen diesen Seiten bereit. Im Template beeinflussen Sie diese Navigation mit den folgenden Anweisungen:

SEARCH_NEXT_PAGES ..... /SEARCH_NEXT_PAGES

Dies ermöglicht die Anzeige der nächsten und vorherigen Ergebnisseiten. Innerhalb dieser Sequenz werden folgende Befehle interpretiert

`PAGE_NR_URL` = Pfad und Dateiname zu dieser Ausgabeseite

`PAGE_NR` = Nummer der Ausgabeseite

`SKIP_CURRENT ..... /SKIP_CURRENT` = löscht den eingeschlossenen HTML-Code, wenn die Ergebnisseite gleich der aktuellen Ergebnisseite ist.

`SKIP_FIRST ..... /SKIP_FIRST` = löscht den eingeschlossenen HTML-Code, wenn die Ergebnisseite gleich der ersten Ergebnisseite ist.

`SKIP_LAST ..... /SKIP_LAST` = löscht den eingeschlossenen HTML-Code, wenn die Ergebnisseite gleich der letzten Ergebnisseite ist.

`SKIP_NOT_CURRENT ..... /SKIP_NOT_CURRENT` = löscht den eingeschlossenen HTML-Code, wenn die Ergebnisseite ungleich der aktuellen Ergebnisseite ist.

`PRINT_CURRENT ..... /PRINT_CURRENT` = zeigt den eingeschlossenen HTML-Code an, wenn die Ergebnisseite gleich der aktuellen Ergebnisseite ist.

`PRINT_FIRST ..... /PRINT_FIRST` = zeigt den eingeschlossenen HTML-Code an, wenn die Ergebnisseite gleich der ersten Ergebnisseite ist.

`PRINT_LAST ..... /PRINT_LAST` = zeigt den eingeschlossenen HTML-Code an, wenn die Ergebnisseite gleich der letzten Ergebnisseite ist.

NO_RESULT_DELETE ..... /NO_RESULT_DELETE

Der eingeschlossene HTML-Code wird gelöscht, wenn es keine Ergebnisse gibt.

NO_RESULT_PRINT ..... /NO_RESULT_PRINT

Der eingeschlossene HTML-Code wird gelöscht, wenn es Ergebnisse gibt.

NO_NEXTPAGE_PRINT ..... /NO_NEXTPAGE_PRINT

Es wird keine Seitenzahl angezeigt, wenn es Ergebnisse gibt.

SINGLE_PAGE_DELETE

Der eingeschlossene HTML-Code wird gelöscht, wenn es nur eine Ergebnisseite gibt.

SINGLE_PAGE_PRINT

Der eingeschlossene HTML-Code wird angezeigt, wenn es nur eine Ergebnisseite gibt.

Beispiel:

```
[!--NO_RESULT_DELETE--]
    [!--SEARCH_NEXT_PAGES--]
        <!--SINGLE_PAGE_DELETE-->
            Weitere Treffer:
            [!--SKIP_CURRENT--]
                <a href="[!--PAGE_NR_URL--]">
            [!--/SKIP_CURRENT--]
             [!--PAGE_NR--]
            [!--SKIP_CURRENT--]
                </a>
            [!--/SKIP_CURRENT--]
        <!--/SINGLE_PAGE_DELETE-->

        <!--SINGLE_PAGE_PRINT-->
            Die Suche ergab keine weiteren Ergebnisseiten.
        <!--/SINGLE_PAGE_PRINT-->
    [!--/SEARCH_NEXT_PAGES--]
[!--/NO_RESULT_DELETE--]
```

3.9.3.6 Tags zur Kontrolle der Suchindizierung

Mit speziellen Tags ist es möglich, die Indizierung der HTML-Dokumente zu beeinflussen. So ist es möglich, Teile von Dokumenten aus der Indizierung auszunehmen oder zusätzliche Metafelder für die Suche zu setzen.

<!--IFTS_NO_PARSE--> ..... <!--IFTS_PARSE-->

Normalerweise werden alle Texte, die im Body-Tag eines HTML-Dokuments stehen, beim Indizieren erfasst, so dass nach ihnen gesucht werden kann. Mit den Tags <!--IFTS_NO_PARSE--> und <!--IFTS_PARSE--> können Sie aber Bereiche definieren, deren Inhalt nicht indiziert wird. Dies ist meist sinnvoll bei Navigationen, Kopf- oder Fußzeilen sowie Inhalten der rechten Spalte.

Beispiel:

```
<html>
    <head>
        ...
    </head>
    <body>
        <!--IFTS_NO_PARSE-->
        ...
        Dieser Bereich wird nicht indiziert
        ...
        <!--IFTS_PARSE-->
        ...
        Dieser Bereich wird indiziert
        ...
```

```
        <!--IFTS_NO_PARSE-->
        ...
    </body>
</html>
```

<!--IFTS_META...-->

Mit Hilfe des Tags „IFTS_META“ können Metawerte speziell für die Suche ergänzt werden. Groß- und Kleinschreibung muss bei der Angabe der Metavariablen nicht beachtet werden; sie müssen jedoch in der Konfigurationsdatei `index.conf` definiert worden sein.

Beispiel:

```
<html>
    <head>
        <title>Seitentitel</title>
    </head>
    <body>
        ...
        <!--IFTS_META_TITLE ergänzt für Volltextsuche-->
        ...
    </body>
</html>
```

Die Anzeige des Titels im Suchergebnis würde in diesem Beispiel „Seitentitel ergänzt für Volltextsuche“ lauten.

Kapitel 4. Flexmodule und Slots

Flexmodule sind HTML-Bausteine, auf die ein Benutzer beim Erstellen eines Artikels zugreifen kann. Er kann sie beliebig oft in beliebiger Reihenfolge kombinieren.

Sie können beliebige HTML- und Perl-Bausteine erstellen und als Flexmodul in Imperia zur Verfügung stellen. Imperia erwartet diese Module im Verzeichnis `/site/flex` auf dem Develop-System. Eine Einschränkung der Auswahl verfügbarer Module in einem Template können Sie über so genannte Valid- und Invalidcodes vornehmen. Die folgende Tabelle gibt Aufschluss über die im Zusammenhang mit Flexmodulen gebräuchliche Terminologie.

Begriff	Bedeutung
Flexmodul	Der Begriff Flexmodul bezeichnet die einzelne Datei, die statische oder dynamische Bestandteile des Templates enthält. Diese können entweder HTML- oder Perlcode zur Erzeugung von HTML enthalten.
Flexmodulleiste	Über die Flexmodulleiste lassen sich die vorhandenen Flexmodule im Template auswählen. Ein Template kann mehrere Flexmodulleisten enthalten.
Flexmodulinstanz	Die Flexmodulinstanz ist ein mit der Flexmodulleiste in das Template eingefügtes Flexmodul. Von einem Flexmodul lassen sich in einer Flexmodulleiste beliebig viele Instanzen erzeugen.

Tabelle 4.1. Begriffe zum Thema Flexmodule

Mit einer Flexmodulleiste (siehe Kapitel **Flexmodule** im Userhandbuch) verwalten Sie die Flexmodule im Template. Dieses Steuerelement können Sie dem Benutzer in zwei mitgelieferten Versionen anbieten. Diese unterscheiden sich im Umfang der Bedienungsmöglichkeiten der Flexmodulleiste. Darüber hinaus kann die Flexmodulleiste vom Kunden mit einem eigenen "Skin" versehen werden.

Normal

Dem Benutzer stehen alle Bedienungselemente zur Verfügung. Dies ist die Default-Einstellung.

Abb. 4.1: Flexmodulleiste im Normal-Modus

Kompakt

Diese Version ist mit eingeschränkten Funktionen ausgestattet. Der Benutzer kann keine Flexmodul-Kombinationen speichern, öffnen oder löschen. Außerdem besteht keine Möglichkeit zur Übergabe von Parametern an aufgerufene Modulinstanzen.

Abb. 4.2: Flexmodulleiste im Kompakt-Modus

Es können beliebig viele Flexmodul(-Instanzen) in einer Flexmoduleleiste und beliebig viele Flexmodulleisten in einem Template vorkommen.

Welche dieser beiden Varianten zum Einsatz kommt, bestimmen Sie durch Aufrufparameter bei der Einbindung der Flexmodulleiste im Template (siehe Abschnitt 4.2 **Einbau in ein Template** auf Seite 104). Mehr Informationen über die Aufrufparameter für Flexmodulleisten finden Sie in Abschnitt 4.3 **Parameter** auf Seite 105.

4.1 Aufbau eines Flexmoduls

Flexmodule können HTML- oder Perl-Code enthalten. Beide Varianten haben eine eigene Dateiendung. Speichern Sie HTML-Flexmodule mit der Endung `.htms` ab und Perl-Flexmodule mit der Endung `.perl`, bzw. `.pl`. Jedes Flexmodul besteht aus einem Header und einem Rumpf mit dem eigentlichen Code. Im Header definieren Sie folgende Eigenschaften:

- Beschreibung des Flexmoduls
- Valid- und Invalidcodes (optional)
- Autor des Flexmoduls (optional)

Der Header enthält Informationen, die in der Flexmodulleiste und in der Flexmodul-Verwaltung erscheinen. Hierzu gehört eine Beschreibung (`DESCRIPTION`) sowie die Angabe des Autors des Flexmoduls und Valid- bzw. Invalidcodes. Die Angabe der `DESCRIPTION` ist bindend. Die übrigen Angaben sind optional. Ebenfalls bindend ist die Trennung des Headers vom Rumpf durch eine Leerzeile. Beispiel für den Header eines Flexmoduls:

```
DESCRIPTION: Textarea
AUTHOR: Imperia Sample
VALIDCODES: kultur,sport,veranstaltungen

[Flexmodul-Code]
```

Achtung:

Valid- oder Invalidcodes müssen Sie im Header eines Flexmoduls noch vor der Leerzeile eintragen.

Der Rumpf enthält den eigentlichen Code, den der Templateprozessor in das Template einfügt, wenn ein Benutzer das Flexmodul in der Flexmodulleiste auswählt. Es kann beliebigen HTML- oder Perlcode enthalten.

Beispiel:

```
[Header]

<textarea name="IMPERIA" rows="7" cols="40" wrap="soft">Text 1</textarea>
<textarea name="IMPERIA:text" rows="7" cols="40" wrap="soft">Text 2</textarea>
```

Hinweis:

Eingabefelder in einem Flexmodul können sowohl benannt als auch anonym sein. Bei benannten Feldern ergänzt Imperia den Namen jedes Feldes mit einem Unterstrich gefolgt von der Nummer der Flexmodulleiste gefolgt von einem weiteren Unterstrich und der Nummer der Flexmodulinstanz innerhalb der Flexmodulleiste (siehe Abschnitt 4.6.1.1 **<!--FLEX_INDEX-->** *auf Seite 110 und Abschnitt 4.6.1.2* **<!--FLEX_ID-->** *auf Seite 110).*

4.2 Einbau in ein Template

Auf Flexmodule wird über die Flexmodulleiste zugegriffen. Die Flexmodulleiste kann durch die Art des Aufrufs in verschiedenen Ausprägungen und zusätzlich mit unterschiedlichen Startparametern aufgerufen werden:

```
<!--INSERT_FLEXMODULE-->
<!--INSERT_FLEXMODULE:COMPACT-->
<!--INSERT_FLEXMODULE:VALIDCODE=xxx-->
<!--INSERT_FLEXMODULE:SAVE-->
```

Beim Erstellen eines Dokuments mit dem entsprechenden Template wird die Flexmodulleiste angezeigt und der User kann aus den Flexmodulen auswählen, die aufgrund des aktuellen Aufrufs verwendet werden können (siehe auch Kapitel **Flexmodule** im Userhandbuch).

4.3 Parameter

Im Folgenden werden alle Parameter aufgelistet, die mit dem Flexmodulleisten-Aufruf übergeben werden können. Die Syntax ist wie folgt:

```
<!--INSERT_FLEXMODULE:Parameter1:Parameter2:ParameterN-->
```

4.3.1 COMPACT

Dieser Parameter sorgt dafür, dass ein mitgelieferter Skin zur Darstellung der Flexmodulleiste verwendet wird. Die Darstellung ist kleiner und hat weniger Bedienungselemente. Die Angabe eines Skins wird bei der Verwendung von COMPACT ignoriert.

Abb. 4.3: Flexmodulleiste im Kompakt-Modus

4.3.2 SKIN

Verwenden Sie diesen Parameter, um von Ihnen definierte Skins für die Flexmodulleiste und die Steuerelemente von Flexinstanzen in einem Flexmodul-Aufruf einzubinden. Die Syntax lautet folgendermaßen:

```
<!--INSERT_FLEXMODULE:SKIN=name-->
```

Der Name ist hierbei nur der erste Teil der unter `site/skin/flex` liegenden Skindateien. In diesem Verzeichnis liegen zwei Beispieldateien, `div.control.htms` und `div.instance.htms`. Um diese beiden Dateien einzubinden, müssten Sie also den Parameter wie folgt notieren:

```
<!--INSERT_FLEXMODULE:SKIN=div-->
```

Wichtig:

Bitte beachten Sie, dass Sie die Skin-Dateien für Flexmodul-Kontrollleiste und Instanz-Kontrollleiste gleich benennen müssen, also z.B. `div.control.htms` *und* `div.instance.htms`*, damit die Einbindung über diesen Parameter funktioniert.*

Abb. 4.4: Flexmodulleiste im Kompakt-Modus

4.3.3 VALIDCODE = *Wert*

Mit diesem Parameter wird die Validierung der Flexmodule gestartet. Alle Flexmodule,

- deren Validcodes den Wert des Parameters enthalten,
- deren Invalidcodes den Wert des Parameters **nicht** enthalten,
- die weder Valid- noch Invalidcodes enthalten,

werden in der Auswahlliste der Flexmodulleiste angezeigt.

4.3.4 VALIDEXP=*Wert*, INVALIDEXP=*Wert*

Dies ist eine erweiterte Syntax für die Validierung der Flexmodule. Sie erlaubt die Verwendung von Regulären Ausdrücken zur Prüfung, ob ein Flexmodul angezeigt werden soll oder nicht.

Es werden beide Reguläre Ausdrücke für jedes Flexmodul in der Form `^EXP$/` geprüft, wobei `EXP` dem Text entspricht, der als Parameter angegeben wurde.

Zuerst werden alle Flexmodule mit der INVALIDEXP geprüft, danach mit der VALIDEXP. Der erste Treffer entscheidet.

INVALIDCODES werden ignoriert. Hat ein Flexmodul keine VALIDCODES, wird der Reguläre Ausdruck gegen einen leeren String geprüft. Wird keine Übereinstimmung gefunden, wird das Flexmodul nur dann angezeigt, wenn keine VALIDEXP angegeben wurde.

Beispiele:

Ausdruck	Erklärung
`VALIDEXP=imperia.*`	Stimmt mit allen VALIDCODES überein, die mit Imperia beginnen.
`VALIDEXP=imperia.*:INVALIDEXP=.*not`	Findet alle Flexmodule, deren Validcodes nicht mit not enden, aber mit Imperia beginnen.
`INVALIDEXP=.*`	Zeigt eine leere Auswahlliste in der Flexsteuerung und schaltet das Hinzufügen neuer Instanzen aus.
`VALIDEXP=`	Findet alle Flexmodule, die keine VALIDCODES haben.
`VALIDEXP=abc.*:INVALIDEXP=abc.*`	Zeigt eine leere Auswahlliste in der Flexsteuerung, weil bereits alle Flexmodule mit dem INVALIDCODE ausgeschaltet wurden.

Tabelle 4.2. Beispiele Valid- und Invalidcodes

4.3.5 SAVE

Dieser Parameter aktiviert das SAVE-Flag, so dass der erzeugte HTML-Code dieser Flexmodulleiste im Metafeld `__imperia_flexcontent_<!--FLEX_INDEX-->` gespeichert werden kann.

4.3.6 INDEX=*xxx*

Dieser Parameter setzt den Index der Flexmodulleiste manuell. Er kann verwendet werden, um die Reihenfolge der Flexmodulleiste innerhalb eines Templates zu verändern, ohne den Index ebenfalls zu verändern. Es ist weiterhin sehr praktisch für Copyseiten mit einem anderen Template, in dem einige Flex-Elemente nicht verwendet werden.

4.3.7 ML_INDEX=xxx

In mehrsprachigen Dokumenten müssen Sie Flexmodule mit einem festen Index versehen. Verwenden Sie hierfür den Parameter *ML_INDEX*. Bei dieser Variante des Index-Parameters ergänzt der Templateprozessor die Indizes der auf die Hauptsprache folgenden Kopien der Flexmodulleiste für die einzelnen Sprachversionen automatisch in Tausenderschritten. Das Zählintervall dieser automatischen Ergänzung lässt sich mit der Systemkonfigurations-Variablen `ML_RANGE` auf einen beliebigen Wert setzen.

Hinweis:

Auch Imperia-Blöcke müssen Sie mit diesem Parameter indizieren.

4.3.8 PARAMETERS=*xxx*

Dieser Einstellung gibt an, welche Parameter der Benutzer an das einzufügende Flexmodul übergeben kann. Als Standard werden zwei Comboboxen angezeigt, mit deren Hilfe der User zwei Zahlen auswählen kann. Beispiele:

```
PARAMETERS=text,text
```

Erzeugt in der Flexmodulleiste zwei Textboxen. Für jede Textbox kann ein Defaulttext angegeben werden, dieser muss jedoch in eckige Klammern gesetzt werden:

```
PARAMETERS=[text;default=my+default+value],select
```

Dies erzeugt eine Textbox mit dem Text `my default value`. Alle Texte müssen URI-encoded maskiert werden. Ein Leerzeichen wird durch ein Pluszeichen maskiert, andere Sonderzeichen werden mit Hilfe eines Prozentzeichens und einer hexadezimalen Zahl maskiert.

```
PARAMETERS=none
```

Blendet die interaktiven Elemente zur Parametereingabe aus.

4.3.9 ALLOW=*xxx*,*yyy*

Mit Hilfe dieses Parameters können bestimmte Bedienungsmöglichkeiten ausgeschlossen werden. Als Standard sind alle Aktionen erlaubt. Mehrere Aktionen werden durch Kommata voneinander getrennt. Mögliche Aktionen sind:

Aktion	Beschreibung
`insert`	neue Instanzen einfügen
`delete`	vorhandene Instanzen löschen
`move`	Reihenfolge vorhandener Instanzen verändern
`all`	alle Aktionen sind erlaubt
`none`	keine Aktion ist erlaubt

Tabelle 4.3. Aktionen für ALLOW

Hinweis:

Dieses Feature verhindert nicht, dass der User die Flexhistory verändert, sondern versteckt lediglich die entsprechenden Bedienungselemente.

4.3.10 LABEL

Der in diesem Parameter gespeicherte Label wird im Flex-Flexxer (siehe Abschnitt 4.9 **Flex-Flexxer** auf Seite 112) in den Auswahllisten verwendet, um die Flexmodulliste identifizieren zu können. Das Label wird im Aufruf für eine Flexmodulliste angegeben und kann in der Reihe der Parameter an beliebiger Stelle erscheinen:

```
<!--INSERT_FLEXMODUL:andere Parameter:LABEL=Text:andere Parameter-->
```

4.3.11 POSITION_SELECT=*all | element | none*

Mit diesem Parameter steuern Sie die Anzeige der Selectbox zur Positionierung in der Flexmodulliste und in der Titelleiste der einzelnen Flexmodulinstanzen (siehe Kapitel "Flexmodul einfügen" und "Flexmodule verschieben" im Userhandbuch).

Beispiele:

Parameter	Beschreibung
`"POSITION_SELECT=all"`	Die Selectboxen mit den Positionslisten werden in der Flexmodulliste und in den Titelleisten der einzelnen Flexmodulinstanzen angezeigt (Default).
`"POSITION_SELECT=element"`	Die Selectboxen mit den Positionslisten werden nur in der Flexmodulliste angezeigt, aber nicht in den Titel-Leisten der einzelnen Instanzen. Diese Einstellung bietet sich an, wenn Sie voraussichtlich sehr viele Modul-Instanzen einsetzen, da der Aufbau der vielen Selectboxen für die Instanzen den Seitenaufbau spürbar verlangsamt.
`"POSITION_SELECT=none"`	Weder für die Flexmodulliste noch für die Instanzen wird eine Selectbox zur Positionierung angezeigt.

Tabelle 4.4. Mögliche Werte und ihre Bedeutung

4.3.12 INSERT_POSITION=*bottom | top*

Mithilfe dieses Parameters legen Sie fest, ob eine neue Flexmodulinstanz ober- oder unterhalb der vorigen in das Dokument eingefügt wird.

Beispiele:

Parameter	Beschreibung
`"INSERT_POSITION=bottom"`	Neue Flexmodul-Instanzen werden unterhalb der bestehenden eingefügt (Default).
`"INSERT_POSITION=top"`	Neue Flexmodul-Instanzen werden oberhalb der bestehenden eingefügt.

Tabelle 4.5. Mögliche Werte und ihre Bedeutung

4.4 Zugriff auf Slots und Flexmodule einschränken

Grundsätzlich sind alle Slot- oder Flexmodule im Auswahlfeld der Flexmodulleiste aufgeführt und stehen so dem Redakteur zur Verfügung. Die Liste der verfügbaren Module lässt sich aber auch einschränken. So können Sie in einem Template ausschließlich die jeweils passenden Flexmodule zur Auswahl anbieten. Dabei ist es möglich, Module wahlweise gezielt in einem Template zuzulassen oder auszuschließen.

Imperia nutzt dafür die so genannten Validcodes bzw. Invalidcodes. Beim Aufruf der Flexmodulleiste im Template übergeben Sie als Parameter einen entsprechenden Code, mit dem Sie bestimmen, welche Flexmodule darin verfügbar sind. Dafür können Sie sowohl Zahlen als auch Buchstaben verwenden. Imperia gleicht diesen Code mit etwaigen Valid- und Invalidcodes aus den Headern der einzelnen Flexmodule ab und lädt nur die passenden Module in die Flexmodulleiste.

Dabei fungiert der im Template übergebene Code-Parameter gleichzeitig für die Zulassung und für den Ausschluss verschiedener Flexmodule.

In der Praxis bedeutet das:

- Ein Aufruf mit der Codeziffer 5 (`<!--INSERT_FLEXMODULE:VALIDCODE=5-->`) schließt alle Flexmodule mit dem *VALIDCODE* 5 ein, alle Flexmodule mit dem *INVALIDCODE* 5 jedoch aus.
- Mit Hilfe dieser Funktion kann die Liste der in einem Template verfügbaren Flexmodule den Bedürfnissen angepasst werden.

Validcodes bzw. Invalidcodes notieren Sie im Header eines Flexmoduls mit folgender Syntax:

```
VALIDCODES:[code1],[code2],[...]
INVALIDCODES:[code1],[code2],[...]
```

Mit *VALIDCODES* legen Sie fest, wann das betreffende Modul in der Auswahl erscheinen soll. Die *INVALIDCODES* verwenden Sie für den expliziten Ausschluss eines Moduls. Mehrere Codes notieren Sie als kommaseparierte Liste ohne Leerzeichen. Als Code kommen Zahlen und / oder Buchstaben in Frage.

Beispiel:

```
VALIDCODES:1,news,5,7
INVALIDCODES:2,archiv,6,8
```

Flexmodule ohne Valid- oder Invalidcodes im Header erscheinen grundsätzlich in einer Flexmodulleiste, auch wenn diese mit einem Validcode-Parameter versehen ist.

Hinweis:

Codes müssen in einem Flexmodul eindeutig sein. Sie können einen Code also nicht gleichzeitig als Validcode und Invalidcode einsetzen.

Im Template ergänzen Sie den Aufruf der Flexmodulleiste entsprechend durch den gewünschten Code. Die Syntax hierfür sieht folgendermaßen aus:

```
<!--INSERT_FLEXMODULE:VALIDCODE=[code]-->
```

Beispiele:

```
<!--INSERT_FLEXMODULE:VALIDCODE=5-->
<!--INSERT_FLEXMODULE_COMPACT:VALIDCODE=archiv-->
```

Die Flexmodulleisten aus diesem Beispiel enthalten nun alle Flexmodule, die mit den Validcodes „5“ bzw. „archiv“ ausgestattet sind. Außerdem sind alle Flexmodule aufgeführt, in deren Header weder Valid- noch Invalidcodes definiert sind. Alle Flexmodule mit dem Invalidcode 5 sind ausgeschlossen.

4.5 Flexmodule mit Perl-Code

Anstelle von HTML-Code können Flexmodule auch Perl-Code enthalten. Wird ein solches Flexmodul aufgerufen, führt Imperia den enthaltenen Code aus und gibt das Ergebnis im Template aus. Flexmodule, die Perl-Code enthalten, müssen die Dateiendung `.perl`, bzw. `.pl` haben.

Beispiel für ein Flexmodul mit Perl-Code:

```
#DESCRIPTION: Perl-Beispiel
#AUTHOR: Imperia Sample

$new = "Dies ist ein einfaches Beispiel, um zwei Zahlen in einem";
$new .= "Flexmodul zu addieren.";
$new .= ($param1 + $param2);
```

Im Unterschied zu HTML-Flexmodulen müssen Sie vor allen Header-Zeilen ein Rautezeichen '#' notieren. Nur dann kann Imperia den Perl-Code aus dem Flexmodul auführen.

Das Flexmodul aus obenstehendem Beispiel übernimmt als Parameter die Werte der Drop-Down-Listen im Abschnitt **Parameter** der Flexmodulleiste. Den HTML-Code für die Ausgabe speichert das Modul in der Variablen `$new`.

Mit einem Perl-Flexmodul können Sie auf alle Daten zugreifen, die in Imperia bekannt sind:

Element	Beschreibung
`%SYSTEM_CONF`	Die Daten der Systemkonfiguration
`$local_metainfo`	Alle Metainformationen dieser Flexmodulinstanz.
`$metainfo`	Alle Metainformationen im Dokument.
`$mode`	Der Modus, in dem sich das Dokument befindet (**EDIT**, **PREVIEW** etc.).
`$userconf`	Zugriff auf die Informationen zum Benutzer. `%USER_CONF`
`@parameters`	Liste der übergebenen Parameter
`$params`	Referenz auf den Hash mit den Flexvariablen (siehe Abschnitt 4.6 **Variablen in Flexmodulen** auf Seite 110). Zugriff erfolgt beispielsweise über $params->{'flex_id'} für die ID der Flex-Instanz.
`$param1`	Erster übergebener Parameter. Alias für $parameters[0].
`$param2`	Zweiter übergebener Parameter. Alias für $parameters[1].

Tabelle 4.6. Elemente und ihre Bedeutung

Sie können in einem Flexmodul beliebigen Perl-Code ausführen lassen. Ein weiteres Beispiel ist eine Tabelle mit variabler Spalten- und Zeilenanzahl. Diese bestimmt der Benutzer über die Drop-Down-Listen für Parameter in der Flexmodulleiste. Hier der Code dazu:

```
#DESCRIPTION: Perl-Tabelle
#AUTHOR: Imperia

$new = "<TABLE>\n";
for ($i = 0; $i < $param1; $i++) {
  if ($i < 1) { $bgm = "bgcolor=\#aaaaaa";} else { $bgm = "";}
  $new .= "<TR $bgm>\n";

  for ($j = 0; $j < $param2; $j++) {
    if ($j < 1) { $bgm = "bgcolor=\#aaaaaa";} else { $bgm = "";}
    $new .= "<TD $bgm>\n";
    $new .= '<INPUT NAME="IMPERIA:text" SIZE="15" TYPE="text" VALUE="" />';
    $new .= "</TD>\n";
    }
  $new .= "</TR>\n";
}
$new .= "</TABLE>\n";
```

4.6 Variablen in Flexmodulen

Auch innerhalb von HTML-Flexmodulen besteht die Möglichkeit, auf verschiedene Variablen zuzugreifen. Der Zugriff erfolgt im Code des Flexmoduls in HTML-Kommentaren. Beispiel:

```
<!--FLEX_ID-->
```

4.6.1 Liste der Variablen

4.6.1.1 <!--FLEX_INDEX-->

Diese Variable enthält einen numerischen Index, der mit 0 beginnt und sich für jede Flexmodulleiste im Template fortlaufend um eins erhöht.

4.6.1.2 <!--FLEX_ID-->

Diese Variable enthält eine numerische ID der Instanz des Flexmoduls, die mit 0 beginnt und sich für jedes Flexmodul einer Flexmodulleiste fortlaufend um eins erhöht.

4.6.1.3 <!--FLEX_NAME-->

Diese Variable enthält den Namen des Flexmoduls, der identisch ist mit dem Dateinamen des Flexmoduls ohne die Endung (`.htms` oder `.perl`/`.pl`).

4.6.1.4 <!--FLEX_PARAM1-->... <!--FLEX_PARAM*n*-->

Diese Variablen enthalten die Parameter, die der User über die Drop-Down-Listen im Abschnitt Parameter der Flexmodulleiste einstellt.

4.6.1.5 <!--XX-FLEX-*Meta-Variable*-->

Diese Variable enthält den Wert einer benannten Flexmodul-Metavariablen. Imperia expandiert diese Variable erst für die Vorschau bzw. beim Speichern des Dokuments. Außerdem können Sie weitere Modifizierer für Variablen aus Imperia-Blöcken oder Arrayblöcken, Escaping-Modes und Indizes für Zugriffe auf Elemente aus Arrays in Metafeldern verwenden. Dazu einige Beispiele:

```
<!-- Beispiel für einen Imperia-Block im Flexmodul -->
<!--XX-FLEX-IBLOCK-mytext-->

<!-- Beispiel für einen Zugriff auf das erste Element eines Arrays
  in einem Metafeld im Flexmodul -->
<!--XX-FLEX-mytext[0]-->
```

```
<!-- Beispiel für einen Zugriff mit Escaping-Modus -->
<!--XX-FLEX-HTMLBR:mytext-->
```

4.6.1.6 <!--XXOBJ-*FLEX-Meta-Variable*-->

Handelt es sich bei dem Inhalt des Metafelds um eine Referenz auf ein MDB-Objekt oder eine per Grafiker-Upload integrierte Datei, müssen Sie mit XXOBJ auf das Metafeld zugreifen. Nur so kann der Templateprozessor die Referenz auf die Datei auflösen. Auch hier ist die Verwendung von zusätzlichen Modifizierern und Indizes möglich (siehe vorhergehenden Abschnitt).

4.6.1.7 <!--XX-FLEX-*Meta-Wert*-->

Diese Variable wird nur innerhalb einer Instanz eines Flexmoduls verwendet. Sie enthält einen eindeutigen numerischen Wert, den das System jeder unbenannten Meta-Variablen innerhalb der Instanz zuweist. Damit lassen sich auch unbenannte Meta-Variablen eindeutig referenzieren.

4.6.1.8 <!--FLEX_POSITION-->

Diese Variable beschreibt die Position der gegenwärtigen Flexmodul-Instanz innerhalb der Flexmodulleiste. Die erste Instanz hat die Position 0. Es wird von oben nach unten gezählt. Im Gegensatz zur FLEX_ID ändert sich die FLEX_POSITION einer Instanz, wenn oberhalb Instanzen entfernt oder hinzugefügt werden.

4.6.1.9 <!--FLEX_COUNTER-->

Diese Variable enthält den um Eins erhöhten Wert von FLEX_POSITION.

4.6.1.10 <!--FLEX_TOTAL-->

Gesamtanzahl der Instanzen innerhalb des Flexelements.

4.6.1.11 <!--FLEX_NEXT_ID-->

Die ID, die Imperia der nächsten Instanz in einem Flex-Element beim Einfügen zuweist.

4.7 Inhalt außerhalb eines Flexmoduls verwenden

Sie können ausschließlich Inhalte aus benannten Metafeldern außerhalb eines Flexmoduls referenzieren. Über den Namen des Feldes sowie den Flex-Index und die Flex-ID lässt sich die Referenz auf jedes Feld eindeutig herstellen. Der Aufruf baut sich wie folgt auf:

```
<!--XX-Metafeldname_Flex-Index_Flex-ID-->
```

Diese Variable greift auf den Inhalt der Meta-Variablen in einer bestimmten Instanz in einer bestimmten Flexmodulleiste zu. Die Zuordnung geschieht dabei durch Flex-Index und Flex-ID.

Dazu ein Beispiel:

Angenommen ein Template enthält drei Flexmodulleisten. Diese haben jeweils zwei Instanzen. Jede Instanz beinhaltet ein Eingabefeld mit dem Namen `street` und ein weiteres mit dem Namen `city`.

Der Flex-Index für die erste Flexmodulleiste ist 0, für die zweite ist er 1 und für die dritte 2. Imperia nummeriert jede Flexmodul-Instanz analog dazu: Die erste Instanz jeder Flexmodulleiste erhält als Flex-ID die 0, die zweite die 1.

Auf jede dieser Flexmodul-Instanzen können Sie auch außerhalb des Flexmoduls referenzieren. Um zum Beispiel den Inhalt des Feldes `city` in der zweiten Instanz aus der ersten Flexmodulleiste abzurufen, müssen Sie folgende Meta-Variable ansprechen:

```
<!--XX-city_0_1-->
```

Die Nummerierung der Flexmodulleisten und -Instanzen bleibt auch dann erhalten, wenn ein Benutzer deren Reihenfolge im Dokument nachträglich ändert.

Für den Zugriff auf Metafelder innerhalb von Flexmodulen gibt es außerdem noch eine verkürzte Notation, bei der Imperia Flex-Index und Flex-ID des betreffenden Moduls automatisch ergänzt:

```
<!--XX-FLEX-Metafeldname-->
```

4.8 Feste Instanzen einfügen

Sie können Instanzen eines Flexmoduls auch so in ein Template einbinden, dass die Benutzer diese beispielsweise nicht mehr durch andere Module ersetzen oder umpositionieren können. Solche Flexmodule funktionieren im Grunde ähnlich wie ein Code-Include. Verwenden Sie dazu folgende Syntax:

```
<!--INSERT_FLEXINSTANCE:ID=ID:NAME=myFlex-->
```

Dies fügt eine Instanz des Flexmoduls mit dem Namen **myFlex** ein. Die Flex-ID ergibt sich aus dem Wert des Parameters *ID*.

Wollen Sie der Flexmodul-Instanz Parameter übergeben, notieren Sie beispielsweise folgenden Aufruf:

```
<!--INSERT_FLEXINSTANCE:ID=tab:NAME=tabelle:PARAMETERS=3/2-->
```

Dieser Beispielaufruf fügt eine Instanz des Flexmoduls **tabelle** ein und übergibt die Parameter *3* und *2* (Spaltenanzahl und Zeilenanzahl). Die Flex-ID wird auf den Wert `tab` gesetzt.

Verändern Sie die Flex-IDs nicht mehr, nachdem Sie das erste Dokument mit diesem Template angelegt haben.

4.8.1 Optionale Instanzen

Optionale Instanzen funktionieren wie feste Instanzen, mit dem Unterschied, dass ein Benutzer diese wahlweise ein- oder ausblenden kann. Per Default sind optionale Instanzen ausgeblendet und der Benutzer sieht beim Bearbeiten des Dokuments nur eine Titelleiste mit dem Namen des eingefügten Flexmoduls und einem Button zum Einblenden:

Abb. 4.5: Titelleiste einer Flexinstanz mit Button zum Einblenden

Um eine optionale Instanz in ein Template einzufügen, verwenden Sie den zusätzlichen Parameter *OPTIONAL*:

```
<!--INSERT_FLEXINSTANCE:ID=i:NAME=input:OPTIONAL=yes-->
```

Klickt der Benutzer auf den Button zum Einblenden, erscheinen die Eingabeelemente des betreffenden Flexmoduls. Bei eingeblendeter Flex-Instanz ist außerdem in der Titelleiste ein Button zum Ausblenden der Instanz sichtbar.

4.9 Flex-Flexxer

Der Flex-Flexxer bietet die Möglichkeit, komplette Flexmodul-Sets zu kopieren, zu löschen, an ein anderes Flexmodul-Set anzuhängen oder vor einem anderen Flexmodul-Set einzufügen.

Die Funktion können Sie mit Hilfe des folgenden Codes an beliebiger Stelle beliebig oft in ein Template einbauen:

```
<!--INSERT_FLEX_FLEXXER-->
```

Im EDIT-Modus ist dann ein Icon sichtbar. Ein Klick auf dieses Icon öffnet ein Popup-Fenster, in dem Sie Funktionen auswählen können:

Abb. 4.6: Dialog Flex-Flexxer

Dieses Fenster enthält drei Auswahllisten, mit deren Hilfe Sie die gewünschte Aktion einstellen:

Quelle

Wählen Sie die ID der zu kopierenden Flexmodulleiste anhand der angezeigten Label aus. Lesen Sie zu den Labels auch Abschnitt 4.3 **Parameter** auf Seite 105.

Aktion

In diesem Auswahlfeld bestimmen Sie die durchzuführende Aktion. Zur Verfügung stehen **Ersetzen**, **Leeren**, **Anhängen** und **Am Anfang einfügen**.

Ziel

Legen Sie die ID der Ziel-Flexmodulleiste fest.

4.10 Slots

Slots sind eine weitere Möglichkeit, wiederkehrende Template-Bausteine zu gruppieren. Die Handhabung von Slots im Template entspricht weitgehend der von Flexmodulen. Innerhalb eines Slots können Sie auch Flexmodule aufrufen. Das ermöglicht das Schachteln von Flexmodulen. Slots lassen sich jedoch nicht schachteln.

4.10.1 Aufbau eines Slot-Moduls

Slot-Module können HTML oder Perl-Code enthalten. Speichern Sie die Dateien im Verzeichnis `site/slot` Ihres Entwicklungs-Systems ab. Eine Slot-Verwaltung über die Benutzeroberfläche von Imperia nach dem Muster der Flexmodul-Verwaltung ist derzeit nicht implementiert. In Syntax und Struktur gleichen Slot-Module den Flexmodulen. Ein Beispiel für ein einfaches HTML-Slot-Modul verdeutlicht dies:

```
❶AUTHOR: superuser superuser
DESCRIPTION: 50-50
VALIDCODES: news,termine
❷
<div style="float:left; width:49%">
❸ <!--INSERT_FLEXMODULE:INDEX=100-->
</div>
<div style="float:left; width:49%">
 <!--INSERT_FLEXMODULE:INDEX=200-->
</div>
<div style="clear:both">
<br /> <hr />
</div>
❹
```

❶ Der Kopf des Slot-Moduls. Hier sind die selben Angaben möglich wie bei Flexmodulen. Lesen Sie hierzu auch Abschnitt 4.1 **Aufbau eines Flexmoduls** auf Seite 103.
❷ Eine Leerzeile trennt den Kopf des Moduls vom Rumpf mit dem eigentlichen Code.
❸ Aufruf eines Flexmoduls im Slot. Die Vergabe eines festen Index ist verbindlich, damit der Templateprozessor eingegebene Inhalte auch nach Verschieben des Slot-Moduls zuordnen kann.
❹ Eine Leerzeile schließt das Modul ab.

4.10.2 Slots im Template aufrufen

Der Aufruf eines Slots im Template gleicht weitgehend dem eines Flexmoduls. Allerdings gibt es für Slots ausschließlich die kompakte Variante der Steuerleiste (vgl. Abschnitt 4.3.1 **COMPACT** auf Seite 105).

```
<!--INSERT_SLOT-->
<!--INSERT_SLOT:VALIDCODE=xxx-->
```

Auch bei Slots haben Sie die Möglichkeit, die Auswahl zur Verfügung stehender Module durch die Verwendung von Validcodes einzuschränken (vgl. Abschnitt 4.3.3 **VALIDCODE** = *Wert* auf Seite 105 und Abschnitt 4.3.4 **VALIDEXP**=*Wert*, **INVALIDEXP**=*Wert* auf Seite 105). Der Einsatz von Validcodes und -Expressions ist analog zu Flexmodulen.

4.10.3 Einschränkungen bei der Verwendung von Slots

Derzeit sind nicht alle Flexmodul-Features auch für Slots implementiert. Folgende Einschränkungen existieren für die Verwendung von Slots:

- Bei Slots ist keine Parameterübergabe möglich wie sie für Flexmodule in Abschnitt 4.3.8 **PARAMETERS**=*xxx* auf Seite 106 beschrieben ist.
- Weitere Slots lassen sich nur an letzter Stelle einfügen. Nachträgliches Umpositionieren ist jedoch möglich.

Wichtig:

Damit Slots mit Flexmodulen sich umpositionieren lassen, müssen Sie die Flexmodulaufrufe mit einem Index versehen.

- Die Verwendung anonymer Variablen ist in Slots nicht möglich.

4.10.4 Variablen in Slots

Slots verwenden die selben Variablen wie Flexmodule. Bei der Referenzierung auf diese Variablen ersetzen Sie das Schlüsselwort `FLEX` durch `SLOT`. Hierzu einige Beispiele:

Variable	Bedeutung
`<!--SLOT_INDEX-->`	Numerischer, mit 0 beginnender Index einer Slotleiste. Erhöht sich für jede Slotleiste im Template um eins.
`<!--SLOT_ID-->`	Numerische, mit 0 beginnende ID der Instanz eines Slotmoduls innerhalb einer Slotleiste. Erhöht sich für jedes Slotmodul einer Slotleiste fortlaufend um eins.
`<!--SLOT_NAME-->`	Der Name des Slotmoduls. Entspricht dem Dateinamen ohne Endung.

Tabelle 4.7. Einige Slot-Variablen

Eine komplette Auflistung der verfügbaren Variablen finden Sie in Abschnitt 4.6.1 **Liste der Variablen** auf Seite 110.

Wichtig:

Da es nicht möglich ist, beim Aufruf Parameter an Slots zu übergeben, gibt es die Variable `<!--SLOT_PARAM-->` *nicht.*

4.10.5 Slot-Inhalte referenzieren

Um die Eindeutigkeit und Referenzierbarkeit von Metafeldern in Slots sicherzustellen, hängt das System an jeden Metafeldnamen in einem Slot einen Index und eine ID an. Diese müssen Sie beim Referenzieren auf die Metafeldinhalte in einem Slot ebenfalls bei den Metafeldnamen ergänzen (vgl. Abschnitt 4.7 **Inhalt außerhalb eines Flexmoduls verwenden** auf Seite 111).

Findet der Zugriff innerhalb des Slot-Moduls selbst statt, können Sie den Modifier `SLOT` verwenden, um eine automatische Ergänzung von Index und ID herbeizuführen. Mehr Informationen zu Modifiern und deren Verwendung finden Sie in Abschnitt 6.16.1 **Modifier für XX-Variablen** auf Seite 145. Zugriffe auf Arrayblöcke und Imperiablöcke erfordern zusätzlich noch die jeweiligen Modifier. Eine Ausnahme bilden hierbei Flexmodule in Slots. Diese können Sie direkt mit dem Modifier `FLEX` referenzieren.

4.10.6 Hierarchie beim Einsatz in Templates

Bei der Verwendung wiederkehrender Strukturelemente in Templates ist folgende Hierarchie vorgegeben:

Element	darf enthalten
`SLOT`	`FLEXMODUL`, `IMPERIABLOCK`, `ARRAYBLOCK`
`FLEXMODUL`	`IMPERIABLOCK`, `ARRAYBLOCK`
`IMPERIABLOCK`	`ARRAYBLOCK`
`ARRAYBLOCK`	-

Tabelle 4.8. Hierarchie wiederkehrender Elemente

Kapitel 5. SiteActive

Die automatische Seitengenerierung in Imperia geschieht mit einem Modul namens SiteActive. Damit können Sie mit wenigen Anweisungen sehr komplexe Seiten automatisch erzeugen lassen. Diese Anweisungen sind in einem bestimmten Zeitintervall ausführbar oder wenn Änderungen in einem so genannten Alarm-Verzeichnis stattfinden, also beim Anlegen, erneuten Freischalten oder Löschen von Dokumenten.

In einem einfachen Fall führt Imperia folgende Arbeitsschritte aus:

1. Benutze eine bestimmte Datei als Template für die erzeugte HTML-Seite.
2. Öffne ein bestimmtes Verzeichnis.
3. Suche in diesem Verzeichnis rekursiv nach Dateien, deren Namen einem bestimmten Muster entsprechen.
4. Sortiere die gefundenen Treffer so, dass beispielsweise die neuesten (bzw. die ältesten) Dateien am Anfang der Trefferliste stehen. Die Trefferliste lässt sich nach zahlreichen weiteren Kriterien sortieren. Lesen Sie hierzu auch Abschnitt 5.3.4 **SORT: Treffer-Liste sortieren** auf Seite 121.
5. Speichere das Resultat in einer Datei mit einem bestimmten Namen in einem bestimmten Verzeichnis.

Die entsprechenden Anweisungen notieren Sie in einem so genannten SiteActive-Template. Sie können dafür entweder die Imperia-eigene SiteActive-Syntax oder Perl verwenden. Das SiteActive-Template enthält neben dem SiteActive-Code auch das Layout der Übersichtsseite. Damit dient es auch als Vorlage der Übersichtsseite, die Sie generieren möchten.

Der Inhalt für die Übersichtsseite wird den Dokumenten entnommen, die das SiteActive entsprechend den darin vermerkten Kriterien findet. Hierzu stehen Ihnen YY-Variablen zur Verfügung (siehe Abschnitt 6.19 **YY-Variablen (nur SiteActive)** auf Seite 146), mit deren Hilfe Sie auf alle Meta-Variablen eines gefundenen Dokuments zugreifen können.

Wird der SiteActive-Code in regelmäßigen Abständen auf dem Zielsystem ausgeführt, erhält man als Ergebnis ständig aktuelle Übersichtsseiten. Voraussetzung hierfür ist, dass der Hintergrund-Daemon **Hermes** auch auf dem Zielsystem läuft.

Hinweis:

Beachten Sie, dass jeder SiteActive-Durchlauf auf dem Server eine entsprechende Last erzeugt.

Die automatische Ausführung eines SiteActives wird über einen Systemdienst gesteuert. Dies erfolgt auf zwei verschiedene Arten :

- Über ein Zeitintervall
- Über ein Alarmverzeichnis

Um ein SiteActive zeitgesteuert auszuführen, stellen Sie im Systemdienst das entsprechende Intervall ein. Auf dem Zielsystem, auf dem die Ausführung des SiteActive-Codes meist stattfindet, überwacht der Hintergrund-Daemon die Systemdienste und startet SiteActive dann in dem eingestellten Intervall.

Wichtig:

Neue Systemdienste und Änderungen an einem bestehenden Systemdienst werden erst nach einem Abgleich der Systemdateien auf dem Zielsystem aktiv.

Die Steuerung über ein Alarmverzeichnis bietet sich an, wenn beispielsweise eine Übersichtsseite über die letzten Nachrichten gepflegt werden soll. Man geht in einem solchen Fall davon aus, dass jede neue Nachricht als eigenes Dokument vorliegt.

Ein solches Dokument landet beim Freischalten in einem bestimmten Verzeichnis auf dem Zielsystem. Gleichzeitig überträgt Imperia eine weitere Datei vom Entwicklungs- zum Zielsystem, das so genannte *notification file*, das im Verzeichnis `/site/incoming` abgelegt wird. Diese Datei enthält unter anderem den Pfad des gerade freigeschalteten Dokuments.

Der Hintergrund-Daemon überprüft das Verzeichnis `/site/incoming` ständig auf neue *notification files*. Findet er eine solche Datei, überprüft er, ob ein Systemdienst existiert, der als Alarmverzeichnis das Verzeichnis des neuen Dokuments oder ein darüber liegendes Verzeichnis enthält.

Existiert ein solcher Systemdienst, dann startet der Hintergrund-Daemon das Skript `site_active.pl` und übergibt diesem Skript die SiteActive-Parameter aus dem Systemdienst.

Das Skript `site_active.pl` liest die Parameter ein und bearbeitet den SiteActive-Code in den IMPERIA-Blöcken des durch den Parameter `TEMPLATE` bestimmten SiteActive-Templates. Daraus resultiert eine Liste mit Dokumenten, die so genannte Trefferliste. Diese Liste bleibt für den gesamten SiteActive-Durchlauf erhalten. Wenn Sie also innerhalb eines Templates mehrere Blöcke mit SiteActive-Anweisungen verwenden, die unterschiedliche Dokumente durchsuchen oder mit verschiedenen Einstellungen arbeiten sollen, müssen Sie vor dem Beginn eines neuen Blocks mit Anweisungen die Liste des vorhergehenden Blocks löschen. Lesen Sie hierzu auch Abschnitt 5.3.6 **Funktionen für die interne Trefferliste** auf Seite 125. Außerdem legt Imperia für Perl- und HTML-SiteActive-Blöcke jeweils getrennte Trefferlisten an.

Das Skript `site_active.pl` extrahiert den durch die YY-Variablen referenzierten Inhalt aus den Dokumenten in der Trefferliste und setzt ihn an Stelle der YY-Variablen ein (lesen Sie Abschnitt 6.19 **YY-Variablen (nur SiteActive)** auf Seite 146 für Informationen zu YY-Variablen). Dabei schränken etwaige LIMIT-Anweisungen die Trefferliste ein. Danach generiert es den HTML-Code für die zu erzeugende Seite.

Weitere Parameter lassen sich über die Konfiguration des Systemdiensts bzw. beim manuellen Aufruf im Querystring mitgeben. Diese sind in SiteActive dann als FF-Variablen referenzierbar. Lesen Sie mehr zu FF-Variablen in Abschnitt 6.3 **FF-Variablen (nur SiteActive)** auf Seite 138.

SiteActives können Sie sowohl auf dem Develop-System als auch auf dem Live-System verwenden. Der Aufruf kann automatisch über einen Systemdienst oder manuell über einen speziellen Link erfolgen (siehe Abschnitt 5.5 **SiteActives manuell aufrufen** auf Seite 135).

Die Voraussetzungen für SiteActives sind:

- Es muss ein SiteActive-Template existieren. Lesen Sie hierzu Abschnitt 5.4 **Beispiele für SiteActives** auf Seite 130.
- Sollen SiteActives automatisch ablaufen, muss der Hintergrund-Daemon Hermes auf dem jeweiligen Server gestartet sein. Lesen Sie hierzu das Kapitel **Der Daemon Hermes** im Administrationshandbuch.
- Für den automatischen Start von SiteActive muss ebenfalls ein Systemdienst für jede SiteActive-Seite existieren. Lesen Sie hierzu Kapitel **Systemdienste** im Administrationshandbuch.

5.1 Ablauf und Bestandteile von SiteActives

In einem einfachen Fall führt Imperia folgende Arbeitsschritte aus:

1. Auswahl eines Templates für die zu erzeugende HTML-Seite.
2. Öffnen eines oder mehrerer Verzeichnisse, in denen sich die Quelldaten befinden.
3. Rekursive Suche nach Dateien, deren Namen einem bestimmten Muster entsprechen.
4. Aufbereitung gefundener Treffer. Die Liste der Treffer lässt sich beispielsweise nach zahlreichen Kriterien sortieren. Lesen Sie hierzu auch Abschnitt 5.3.4 **SORT: Treffer-Liste sortieren** auf Seite 121.
5. Speichern des Resultats in einer Ergebnisseite mit einem bestimmten Namen in einem bestimmten Verzeichnis.

Der Inhalt für eine SiteActive-Seite stammt aus den Dokumenten, die nach von Ihnen festgelegten Kriterien als Treffer zählen. SiteActives können Sie sowohl auf dem Entwicklungs-System als auch auf dem Ziel-System verwenden. Der Aufruf kann automatisch über einen Systemdienst oder manuell über einen speziellen Link erfolgen (siehe Abschnitt 5.5 **SiteActives manuell aufrufen** auf Seite 135).

5.1.1 Voraussetzungen

Voraussetzung für die Nutzung von SiteActives ist ein laufender Hintergrund-Daemon **Hermes** auf dem System, auf dem Sie SiteActive-Seiten generieren lassen möchten. Außerdem benötigen Sie ein zusätzliches Zielsystem mit dem Zielformat *Notification* für jedes Zielsystem, auf dem Sie SiteActives einsetzen möchten. Für das Standard-Zielsystem legt Imperia automatisch ein zusätzliches Notification-Zielsystem an, bei zusätzlichen Zielsystemen müssen Sie dies selbst erledigen. Notifications lösen ereignisgesteuerte SiteActives aus, übertragen aber auch die Dokumenteninhalte auf die Zielsysteme, die Grundlage für alle SiteActives sind.

5.1.2 Systemdienst

Die automatische Ausführung eines SiteActives steuern Sie über einen Systemdienst. Man unterscheidet zwischen zeitgesteuerten und ereignisgesteuerten Systemdiensten.

Hinweis:

Beachten Sie, dass jede SiteActive-Ausführung auf dem Server Last erzeugt. Vermeiden Sie also nach Möglichkeit unnötiges Ausführen, indem Sie Zeitintervalle nicht zu kurz wählen oder ereignisgesteuerte SiteActives einsetzen.

Neben dem Ausführungszeitpunkt legen Sie bei der Systemdienstkonfiguration auch fest, welches Template für das auszuführende SiteActive zu nutzen ist und wo die dabei erzeugte Datei unter welchem Namen zu speichern ist. Weitere Parameter sind in SiteActive-Templates als FF-Variablen referenzierbar. Lesen Sie mehr zu FF-Variablen in Abschnitt 6.3 **FF-Variablen (nur SiteActive)** auf Seite 138.

Die Konfiguration der Systemdienste ist ist eingehend im Abschnitt "**Systemdienste**" im Kapitel "**Administration des Systems**" im Administrationshandbuch beschrieben.

5.1.3 Notification

Beim Freischalten eines Dokuments überträgt Imperia neben dem Dokument eine weitere Datei vom Entwicklungs- zum Zielsystem, das so genannte *notification file*. Diese Datei speichert Imperia im Verzeichnis `/site/incoming` ab. Sie enthält den Zielpfad des zugehörigen Dokuments sowie dessen komplette Metainformation. Der Hintergrund-Daemon überprüft das Imperia-Verzeichnis `/site/incoming` ständig auf neue *notification files*. Stimmt der Zielpfad eines Dokuments oder ein darüber liegendes Verzeichnis mit dem Alarmverzeichnis eines Systemdiensts überein, startet der Hintergrund-Daemon den betreffenden Systemdienst. Hierzu ruft er das Skript `cgi-bin/site_active.pl` auf, wobei die Konfiguration des Systemdiensts die Parameter für den Aufruf bestimmt. Damit beginnt der eigentliche SiteActive-Durchlauf.

5.1.4 SiteActive-Template

In einem SiteActive-Template können Sie mehrere IMPERIA-Blöcke verwenden. Alle IMPERIA-Blöcke werden nacheinander in einem Durchlauf bearbeitet. Hierbei ist zu beachten, dass die interne Liste der gefundene Dokumente und eventuelle Einschränkungen (LIMIT HITS, LIMIT BY METAFIELD etc.) nach der Bearbeitung eines IMPERIA-Blocks nicht automatisch gelöscht werden. Dies muss explizit im nächsten IMPERIA-Block geschehen (siehe Abschnitt 5.4.3 **Beispiel 2: limitierte Linkliste** auf Seite 131).

5.2 SiteActive-Templates

Im Unterschied zu einem gewöhnlichen Dokument stammt der Inhalt für ein SiteActive-Dokument nicht aus einer Benutzereingabe, sondern aus den Metainformationen anderer bereits bestehender Imperia-Dokumente. Neben dem Markup für die Layout-Vorlage enthält ein SiteActive-Template Anweisungen, die festlegen, welche Dokumente als Grundlage dienen und welche Informationen daraus zu extrahieren sind. Site Active Templates können Imperia-spezifische Anweisungen sowie Perl-Code enthalten. Anders als herkömmliche Imperia-Templates, liegen SiteActive-Templates in der Document-Root des Webservers.

SiteActive-Templates lassen sich mit oder ohne Imperia verwalten. Welche Variante die sinnvollere ist, ergibt sich aus der Häufigkeit der Änderungen am Layout oder den statischen Inhalten eines mit SiteActive erzeugten Dokuments und den Zugriffsmöglichkeiten auf das System.

5.2.1 SiteActive-Templates mit Imperia verwalten

Sie können ein SiteActive-Template wie normale Dokumente in einer Rubrik in Imperia verwalten. Diese Vorgehensweise bietet sich unter folgenden Umständen an:

- Das Layout des Dokumentes unterliegt häufigen Änderungen.
- Neben den automatisch generierten Inhalten enthält das Dokument redaktionell gepflegte Bestandteile.
- Zum Zielsystem besteht kein Zugriff auf Dateisystemebene.

Dabei stehen Ihnen dann die Verwaltungsfunktionen von Imperia, wie Archiv, Workflow und Schreibtisch zur Verfügung, um eine geregelte Pflege des SiteActive-Templates zu gewährleisten. Über die Template-Funktionen zum Ausblenden bestimmter Template-Code-Abschnitte in verschiedenen Modi, kann der SiteActive-Code beispielsweise im Edit- und Preview-Modus ausgeblendet werden.

Prinzipiell kann jedes Imperia-Dokument als SiteActive-Template dienen. Es muss lediglich SiteActive-Code enthalten und an eine Stelle freigeschaltet werden, auf die auch ein Systemdienst verweist.

5.2.2 SiteActive-Templates ohne Imperia verwalten

Es ist nicht zwingend erforderlich, ein SiteActive-Template mit Imperia zu pflegen. Sie können Ihre Vorlage auch alternativ direkt auf das Zielsystem transferieren. Dafür sollten folgende Bedingungen gegeben sein:

- Zum Zielsystem besteht Zugriff auf Dateisystemebene.
- Änderungen am Layout des Templates kommen gar nicht oder nur sehr selten vor.
- Das automatisch erzeugte Dokument enthält keinerlei redaktionell gepflegte Inhalte.

In diesem Fall benötigen Sie lediglich eine HTML-Datei mit dem Layout der mit SiteActive zu erzeugenden Übersichtsseite und dem SiteActive-Code zur Extraktion des Inhaltes aus anderen Dokumenten. Diese Datei legen Sie dann auf dem Zielsystem unter dem Pfad ab, auf den auch der Parameter `TEMPLATE` des zugehörigen Systemdienstes verweist.

Änderungen am SiteActive-Template sind dann aber nur über Zugriffe auf Dateisystemebene möglich.

5.3 Syntax-Referenz

Für die automatisierte Seitengenerierung mit SiteActive stellt Imperia eine Reihe eigener Template-Befehle bereit. Neben diesen hier beschriebenen Befehlen können Sie auch die im Abschnitt 3.4.15 **IF-Abfragen** auf Seite 67 erklärten IF-Abfragen verwenden.

Hinweis:

Schlüsselwörter für bedingte Anweisungen in SiteActive-Blöcken können Sie auch mit einem vorangestellten "$" notieren. Es sind also sowohl die Schreibweisen #IF, #ELSE, #ELSIF, #ENDIF als auch $IF, $ELSE, $ELSIF, $ENDIF gültig.

Außer der Imperia-Syntax können Sie auch Perl für die Erstellung von SiteActives nutzen. Für die meisten der hier beschriebenen Template-Befehle sind äquivalente Perl-Methoden in Imperia implementiert. Diese sind im Abschnitt 7.3 **Personalisierungsfunktionen** auf Seite 153 beschrieben.

5.3.1 IMPERIA: SiteActive-Anweisungen einschließen

SiteActive-Code leiten Sie im SiteActive-Template mit dem Tag `<IMPERIA>` ein und schließen Ihne mit dem Tag `</IMPERIA>` ab. Zwischen diesen beiden Tags erscheinen die Schlüsselwörter der Imperia-SiteActive-Syntax. Diese sind in Großbuchstaben zu notieren. Syntax:

```
<IMPERIA>

[SiteActive-ANWEISUNGEN]

</IMPERIA>
```

SiteActive-Anweisungsblöcke mit Perl-Code schließen Sie wie folgt ein:

```
<IMPERIA lang=perl>

[PERL-SiteActive-Anweisungen]

</IMPERIA>
```

Innerhalb eines SiteActive-Templates sind mehrere Blöcke mit SiteActive-Code möglich. Das System führt alle Blöcke eines Templates hintereinander aus.

Wichtig:

Imperia löscht die Trefferliste der Dokumente, die sich anhand der SiteActive-Anweisungen am Anfang eines IMPERIA-Codeblocks aufbaut nach der Abarbeitung aller Anweisungen dieses Blocks nicht automatisch. Die Trefferliste sowie etwaige damit verbundene Einschränkungen (LIMIT HITS, LIMIT BY METAFIELD etc.) bleiben also nach dem Durchlauf des SiteActives erhalten. Beim nächsten SiteActive-Block erstellt das System zwar eine neue Trefferliste, überschreibt damit aber nicht die bereits bestehende, sondern hängt die neuen Treffer lediglich an die bestehende Liste an.

Dies gilt auch für SiteActives aus unterschiedlichen Templates, die zu unterschiedlichen Zeitpunkten laufen und unabhängig davon, ob für die unterschiedlichen Trefferlisten völlig unterschiedliche Auswahlkriterien bestehen. Sollten Sie nicht wünschen, dass ein Block von SiteActive-Anweisungen mit einer Trefferliste arbeiten, die zumindest teilweise aus einer vorherigen Ausführung eines SiteActives stammt, müssen Sie am Anfang des Blocks zuerst die Trefferliste aus dem vorherigen Durchlauf löschen. Hierzu nutzen Sie die Anweisung CLEARLIST (siehe auch Abschnitt 5.4.3 **Beispiel 2: limitierte Linkliste** *auf Seite 131 und Abschnitt 5.3.6* **Funktionen für die interne Trefferliste** *auf Seite 125) oder das Perl-Äquivalent clearlist() (siehe auch Abschnitt 7.3.1, „clearlist()“).*

.

5.3.2 READDIR: Quellverzeichnis bestimmen

Das Verzeichnis, in dem SiteActive nach Treffern sucht, bestimmen Sie mit *READDIR*:

```
READDIR = "/Verzeichnis1"
```

oder

```
READDIR = "Verzeichnis1/Verzeichnis2"
```

Diese Funktion bestimmt ausgehend vom Document-Root das zu durchsuchende Verzeichnis. Dies schließt eventuell vorhandene Unterverzeichnisse mit ein. Beispiel:

```
READDIR = "sport/segeln"
```

Mit dieser Einstellung durchsucht SiteActive unterhalb der Document-Root des Webservers das Verzeichnis `sport/segeln` einschließlich etwaiger Unterverzeichnisse.

Zusätzlich kann bestimmt werden, ob und ab welcher Tiefe Unterverzeichnisse in die Suche einbezogen werden sollen. Hierzu werden Parameter übergeben:

```
READDIR = "Verzeichnis[-Ebenen][-Starttiefe]"
```

Ohne Parameter durchsucht SiteActive alle Verzeichnisse bis zu einer Tiefe von 99 Ebenen.

Achtung:

Sollten Ihre Verzeichnisnamen Minus-Zeichen enthalten (z.B.: `/movies-2-99`*), besteht die Gefahr, die hier beschriebene Funktionalität auszulösen. Das führt zu unerwünschten Resultaten. Verwenden Sie daher in Verzeichnisnamen anstelle des Minus-Zeichens ein anderes Zeichen, zum Beispiel den Unterstrich.*

Beispiele:

```
READDIR = "/movies-0"
```

Diese Syntax sucht nur im Verzeichnis `/movies`, nicht in Unterverzeichnissen.

```
READDIR = "/movies-1"
```

Diese Syntax sucht im Verzeichnis `/movies` und in allen Unterverzeichnis der ersten Ebene, also zum Beispiel `/movies/comedy` und `/movies/action`

```
READDIR = "/movies-1-2"
```

Diese Syntax sucht nicht im Verzeichnis `/movies`, aber in den Verzeichnissen `/movies/comedy` und `/movies/comedy/00267`, aber nicht mehr in tieferen Verzeichnissen.

5.3.2.1 SETDIR: Quellverzeichnis definieren

```
SETDIR "Verzeichnis1/Verzeichnis2"
```

Diese Funktion wird im Zusammenhang mit der `dynamic.conf` verwendet. Mit ihr ist es möglich, die Suche nach Treffern auf einen Teil der vorhandenen Verzeichnisse einzuschränken.

Es werden lediglich die Werte für die Variable <!--dirlevel:*n*--> gesetzt, die von der `dynamic.conf` verwendet wird.

Diese Variable wird auch von `READDIR` gesetzt, jedoch durchsucht `READDIR` physikalisch das gesamte angegebene Verzeichnis und benötigt wesentlich mehr Zeit. Beispiel:

```
SETDIR "news/2001"
```

Mit dieser Einstellung enthält die Variable <!--dirlevel:1--> den Wert `news` und <!--dirlevel:2--> den Wert `2001`.

5.3.3 FILEMASK: Suchmuster bestimmen

`FILEMASK = "`*Regulärer Ausdruck*`"`

Um das Suchmuster zu bestimmen, wird die Funktion `FILEMASK` verwendet. Alle Dateien, deren Name nicht mit dem hier angegebenen Suchmuster übereinstimmt, werden ignoriert. Beispiel:

```
FILEMASK = ".*\.htm[lsx]?$"
```

Dieses Beispiel findet alle Dateien, die mit dem Suffix `.htm`, `.html`, `.htms` oder `.htmx` enden.

5.3.4 SORT: Treffer-Liste sortieren

Die Treffer-Liste kann alphabetisch oder numerisch sortiert werden. Werden keine Einstellungen bezüglich der Sortierrichtung gemacht, wird die Trefferliste aufsteigend numerisch beziehungsweise aufsteigend alphabetisch sortiert.

5.3.4.1 Nach Datum sortieren

Um Treffer nach Datum zu sortieren, verwenden Sie die Funktion `SORT LATEST FIRST` bzw. `SORT OLDEST FIRST`.

```
# neueste Dokumente zuerst:
SORT LATEST FIRST

# älteste Dokumente zuerst:
SORT OLDEST FIRST
```

Die Funktion `SORT LATEST FIRST` sortiert die jüngsten Treffer an den Anfang der Liste, `SORT OLDEST FIRST` sortiert die ältesten Treffer an den Anfang der Liste. Es sind keine weiteren Parameter nötig.

5.3.4.2 Nach dem gesamten Pfad sortieren

Mit der Funktion `SORT BY FILENAME` sortieren Sie die Trefferliste nach dem kompletten Pfad einer Datei.

```
SORT BY FILENAME
```

Besteht der Pfad ganz oder teilweise aus fortlaufenden Nummern, beispielsweise durch Verwendung der Variablen `<!--count-->` oder über Zeitvariablen (z. B.: `<!--hour-->_<!--minute-->_<!--second-->`) im Verzeichnisnamen, kann die Funktion sehr effektiv arbeiten. Weitere Parameter sind nicht nötig.

5.3.4.3 Nach Pfadbestandteilen sortieren

Um die Trefferliste nach Pfadbestandteilen zu sortieren, nutzen Sie die Funktion `SORT BY DIRELEM`. Als Parameter übergeben Sie zwei numerische Werte.

```
SORT BY DIRELEM "Wert1" TO "Wert2"
```

5.3.4.4 Nach Meta-Variablen sortieren

Mit der Funktion `SORT BY METAFIELD` sortieren Sie die Trefferliste nach einer bestimmten Meta-Variablen. Als Parameter übergeben Sie den Namen der Meta-Variablen, nach der sortiert werden soll.

```
SORT BY METAFIELD "Meta-Variablenname"
```

Beispiel:

```
SORT BY METAFIELD "title"
```

Diese Einstellung sortiert die Trefferliste alphabetisch nach dem Seitentitel. Ebenfalls möglich ist die Verwendung von Meta-Variablen, die numerische Werte enthalten.

Um die Trefferliste nach mehreren Meta-Variablen zu sortieren, verwenden Sie die Funktion `SORT BY MULTIFIELD`:

```
SORT BY MULTIFIELD "PRÄFIXMeta-Variable1, PRÄFIXMeta-Variable2, PRÄFIXMeta-VariableX"
```

Mit dem Präfix spezifizieren Sie den zu erwartenden Inhalt des betreffenden Metafelds. Folgende Präfixe sind möglich:

Präfix	Bedeutung
#	Verwenden Sie dieses Präfix, wenn das betreffende Metafeld numerische Daten enthält und Imperia entsprechend sortieren soll.
~	Das Metafeld beinhaltet ISO-Latin-1-Daten, die ohne Beachtung von Groß- und Kleinschreibung zu sortieren sind.
!	Das Metafeld enthält ISO-Latin-1-Daten. Die Sortierreihenfolge soll der Collating-Sequence entsprechen (normalerweise ASCII-Reihenfolge).
+	Aufsteigend sortieren.
-	Absteigend sortieren.

Tabelle 5.1. Präfixe für Multifield-Sortierung

Die Reihenfolge der Sortierung ist abhängig von der Reihenfolge der angegebenen Metafelder.

5.3.4.5 RANDOM PICK: Zufällige Treffer

```
RANDOM PICK
RANDOM PICK Zahl
```

Um zufällig Treffer aus der Trefferliste anzeigen zu lassen, verwendet man die Funktion `RANDOM PICK` bzw. `RANDOM PICK` *Zahl*. Nach jedem SiteActive-Durchlauf werden andere Treffer angezeigt.

Die Funktion `RANDOM PICK` zeigt zufällige Treffer aus der Liste. Die Anzahl der angezeigten Treffer kann zusätzlich über eine LIMIT HITS-Anweisung eingeschränkt werden.

Bei der Funktion `RANDOM PICK` *Zahl* wird die Anzahl der anzuzeigenden Treffer mit Hilfe des numerischen Parameters *Zahl* bestimmt.

5.3.5 LIMIT HITS: Trefferliste eingrenzen / filtern

Mit den Funktionen zur Begrenzung der Trefferliste kann angegeben werden, wie viele der gefundenen Treffer angezeigt werden sollen. Mehrere aufeinander folgende LIMIT-Anweisungen sind mit einem logischen UND verknüpft.

Insbesondere zusammen mit SORT-Anweisungen kann mit Hilfe von `LIMIT HITS` sehr schnell beispielsweise eine Liste der 10 aktuellsten Treffer auf einer Übersichtsseite plaziert werden.

5.3.5.1 Trefferanzeige begrenzen

Verwenden Sie folgende Syntay, um die Trefferanzeige zu begrenzen:

```
LIMIT HITS Zahl
```

Der Parameter *Zahl* bestimmt die Anzahl der im fertigen Dokument angezeigten Treffer. Beispiel:

```
LIMIT HITS 10
```

Die Anzahl der angezeigten Treffer ist auf maximal 10 begrenzt.

5.3.5.2 Trefferliste auf einen Bereich begrenzen

Wollen Sie nur einen bestimmten Bereich aus der Trefferliste verwenden, benutzen Sie die Funktion LIMIT HITS "*Wert 1*" TO "*Wert 2*", um diesen Bereich zu definieren.

```
LIMIT HITS "Wert 1" TO "Wert 2"
```

Geben Sie als Parameter zwei numerische Werte an. Der erste bezeichnet den Anfang des Bereichs. Der zweite bezeichnet dessen Ende. Die weitere Verarbeitung der Trefferliste findet nur innerhalb dieses Bereichs statt (einschließlich Anfang und Ende).

5.3.5.3 Trefferliste durch eine Meta-Variable begrenzen

Mit der Funktion `LIMIT BY` schränken Sie die Anzahl der angezeigten Treffer abhängig von einer Meta-Variablen ein.

`LIMIT BY` *Meta-Variable* `"`*Wert 1*`" TO "`*Wert 2*`"`
`NOCASELIMIT BY` *Meta-Variable*`"`*Wert 1*`" TO "`*Wert 2*`"`

Neben der Angabe der Meta-Variablen müssen Sie hier numerische Grenzwerte angeben. Notieren Sie den Namen der Meta-Variablen ohne HTML-Kommentarzeichen. Hierzu ein Beispiel:

```
LIMIT BY limit "2" TO "5"
```

Dieses Beispiel setzt das Vorhandensein der Meta-Variablen `limit` in den durchsuchten Dokumenten voraus. In der Trefferliste erscheinen nur Dokumente, die in der Meta-Variablen `limit` die Werte 2, 3, 4 oder 5 enthalten.

Wenn Sie möchten, dass für den Meta-Variablennamen keine Groß- oder Kleinschreibung beachtet wird, verwenden Sie statt `LIMIT` das Schlüsselwort `NOCASELIMIT`.

5.3.5.4 REJECT: Bestimmte Dateien ausschließen

Mit der Funktion *REJECT* schließen Sie bestimmte Dateien aus der Trefferliste aus.

`REJECT "`*Pfad*`"`

Den Pfad der auszuschließenden Dateien notieren Sie in doppelten Anführungszeichen.

5.3.5.5 ALLOW: Ausschluss bestimmter Dateien aufheben

Wenn Sie in einem SiteActive-Block bestimmte Dokumente mit *REJECT* (siehe Abschnitt 5.3.5.4 **REJECT: Bestimmte Dateien ausschließen** auf Seite 124) von der Trefferliste ausgeschlossen haben, können Sie diesen Ausschluss mit *ALLOW* wieder aufheben:

`ALLOW "`*Pfad*`"`

Geben Sie den Pfad in doppelten Anführungszeichen an.

5.3.5.6 FILTER: Trefferliste filtern

`FILTER "`*Meta-Variable*`" CONTAINS "`*Regulärer Ausdruck*`"`

Mit dem Schlüsselwort `FILTER` definieren Sie einen Filter für die Trefferliste. Filter müssen Sie **VOR** der Anweisung `READDIR` definieren, damit sie wirksam sind. Mit einem zusätzlichen Operator steuern Sie die Wirkungsweise des Filters. Folgende Operatoren stehen zur Verfügung:

- (NOT)CONTAINS
- (NOT)SMALLER
- (NOT)GREATER
- (NOT)EQUALS

Bei den Operatoren `CONTAINS` und `NOTCONTAINS` können Sie Reguläre Ausdrücke verwenden. `(NOT)SMALLER` und `(NOT)GREATER` sind numerische Vergleiche. `EQUALS` und `NOTEQUALS` sind case-insensitiv.

Für weitere Informationen zur Verwendung lesen Sie Abschnitt 5.4.4 **Beispiel 3: gefilterte Trefferliste** auf Seite 132.

Wichtig:

Einmal definierte Einschränkungen bleiben ebenso wie die zugehörige Trefferliste nach dem Durchlauf eines SiteActives erhalten. Wenn Sie keine Übernahme von Treffern und Einschränkungen aus vorherigen SiteActives wünschen, müssen Sie am Anfang des SiteActive-Blocks die Anweisung CLEARLIST notieren bzw. das Perl-Äquivalent clearlist(). Beide Anweisungen beinhalten auch das Löschen sämtlicher Einschränkungen. Lesen Sie hierzu auch Abschnitt 5.4.3 **Beispiel 2: limitierte Linkliste** *auf Seite 131 und Abschnitt 5.3.6* **Funktionen für die interne Trefferliste** *auf Seite 125, bzw. Abschnitt 7.3.1, „clearlist()".*

5.3.6 Funktionen für die interne Trefferliste

5.3.6.1 CLEARLIST: Interne Trefferliste löschen

```
CLEARLIST
```

Wünschen Sie keine Übernahme von Trefferlisten und Einschränkungen aus vorhergehenden SiteActive-Blöcken bzw. -Durchläufen, müssen Sie die interne Trefferliste am Anfang eines SiteActive-Blocks löschen, da ansonsten der neue Block mit den im vorhergehenden Block gefundenen Treffern und zugehörigen Einschränkungen arbeitet. Verwenden Sie hierzu die Anweisung `CLEARLIST`.

Hinweis:

Diese Funktion löscht auch etwaige Einschränkungen, die für die vorhergehende Trefferliste bestehen (LIMIT HITS, LIMIT BY METAFIELD etc.).

5.3.6.2 CLEARLIMIT: Beschränkungen aufheben

```
CLEARLIMIT
```

Mit `CLEARLIMIT` löschen Sie lediglich die Einschränkungen aus einer vorhergehenden Trefferliste, nicht aber die Liste.

5.3.6.3 REVERSE LIST: Trefferliste umkehren

Mit der Funktion `SORT BY METAFIELD` sortieren Sie die Trefferliste nach einer bestimmten Meta-Variablen. Als Parameter übergeben Sie den Namen der Meta-Variablen, nach der sortiert werden soll.

```
SORT BY METAFIELD "Meta-Variablenname"
```

Beispiel:

```
SORT BY METAFIELD "title"
```

Diese Einstellung sortiert die Trefferliste alphabetisch nach dem Seitentitel. Ebenfalls möglich ist die Verwendung von Meta-Variablen, die numerische Werte enthalten.

Verwenden Sie die Funktion *REVERSE LIST*, um die Reihenfolge der Treffer umzukehren.

```
REVERSE LIST
```

Diese Funktion bietet sich im Zusammenhang mit folgenden Funktionen an:

```
SORT BY FILENAME
SORT BY METAFIELD
SORT BY DIRELEM
```

Um die Trefferliste nach mehreren Meta-Variablen zu sortieren, verwenden Sie die Funktion `SORT BY MULTIFIELD`:

```
SORT BY MULTIFIELD "PRÄFIXMeta-Variable1, PRÄFIXMeta-Variable2, PRÄFIXMeta-VariableX"
```

Mit dem Präfix spezifizieren Sie den zu erwartenden Inhalt des betreffenden Metafelds. Folgende Präfixe sind möglich:

Präfix	Bedeutung
#	Verwenden Sie dieses Präfix, wenn das betreffende Metafeld numerische Daten enthält und Imperia entsprechend sortieren soll.
~	Das Metafeld beinhaltet ISO-Latin-1-Daten, die ohne Beachtung von Groß- und Kleinschreibung zu sortieren sind.
!	Das Metafeld enthält ISO-Latin-1-Daten. Die Sortierreihenfolge soll der Collating-Sequence entsprechen (normalerweise ASCII-Reihenfolge).
+	Aufsteigend sortieren.
-	Absteigend sortieren.

Tabelle 5.2. Präfixe für Multifield-Sortierung

Die Reihenfolge der Sortierung ist abhängig von der Reihenfolge der angegebenen Metafelder.

5.3.6.4 IF LIST: Ausgabe in Abhängigkeit von der Trefferlistenlänge

Sie können die Ausgabe von Text abhängig von der Trefferlistenlänge steuern. Dabei legen Sie fest, dass der Text nur erscheint, wenn die Trefferliste länger oder kürzer als ein bestimmter Wert ist, bzw. genau eine bestimmte Länge hat. Verwenden Sie hierzu folgende Syntax:

```
IF LIST GREATER X PRINT "Mehr als X Treffer."
IF LIST SMALLER X PRINT "Weniger als X Treffer."
IF LIST EQUALS X PRINT "Genau X Treffer."
```

Der Parameter *X* ist hierbei Platzhalter für eine Zahl, mit der Sie die gewünschte Länge der Trefferliste einstellen. Mit den Operatoren *GREATER*, *SMALLER* und EQUALS steuern Sie, ob die Trefferliste mehr als, weniger als oder genau die definierte Anzahl Treffer enthalten soll.

5.3.7 PRINT: Text ausgeben

Mit dieser Funktion geben Sie bei der Bearbeitung des SiteActive-Blocks Text aus. Innerhalb des auszugebenden Texts können Sie HTML-Tags zur Formatierung verwenden. Das Schlüsselwort für diese Funktion ist `PRINT`.

```
PRINT "Text"
PRINT 'Text'
```

Hinweis:

Diese Funktion können Sie nicht innerhalb von `FOREACH FOUND`*-Schleifen verwenden.*

Setzen Sie den auszugebenden Text in doppelte oder einfache Anführungszeichen. Beispiele:

```
PRINT 'Eine Liste <b>aller</b> gefundenen Treffer:'
PRINT "Eine Liste <b>aller</b> gefundenen Treffer:"
```

5.3.8 DYNAMIC: Ersetzung durch `dynamic.conf` ein- und ausschalten

Innerhalb eines SiteActive-Blocks können Sie eine zusätzliche Ersetzung innerhalb der Treffer durch die `dynamic.conf` aktivieren und deaktivieren.

```
# dynamic.conf deaktivieren:
DYNAMIC DISABLE

# dynamic.conf aktivieren:
DYNAMIC ENABLE
```

5.3.9 FOREACH Found-Schleifen

Mit `FOREACH FOUND`-Schleifen können Sie Anweisungen für jeden relevanten Treffer ausführen. Die Syntax ist wie folgt:

```
FOREACH FOUND {
  Anweisung 1
  Anweisung 2
  Anweisung 3
    ...
}
```

So steuern Sie die eigentliche Ausgabe eines SiteActives. Notieren Sie in der Schleife eine Reihe von Anweisungen, um diese für jeden gefundenen Treffer ausführen zu lassen. Das folgende Beispiel demonstriert, wie Sie mit einer Schleife aus Pfad, Dateinamen und Titel der Dokumente in einer Trefferliste eine einfache Linkliste aufbauen:

```
FOREACH FOUND {
  <a href="<!--YY-directory-->/<!--YY-filename-->"><!--YY-title-->
  </a><br />
}
```

Ein weiteres Beispiel:

```
FOREACH FOUND {
  <tr>
    <td bgcolor="#c0c0c0" height="10px">
      <a href="<!--YY-directory-->/<!--YY-filename-->"><!--YY-title--></a>
    </td>
  </tr>
}
```

Hier erzeugt die Schleife für jeden Treffer eine Tabellenreihe.

5.3.9.1 IF-Abfragen in FOREACH FOUND-Schleifen

Innerhalb einer FOREACH FOUND-Anweisung kann mit Hilfe von IF-Abfragen überprüft werden, ob

- eine Datei vorhanden oder nicht vorhanden ist,
- ein Wert definiert oder nicht definiert ist,
- zwei Werte gleich oder ungleich sind.

Die Syntax für eine IF-Abfrage ist wie folgt:

```
IF (NOT) EXISTS Bedingung PRINT Text
IF (NOT) DEFINED Bedingung PRINT Text
IF (NOT) EQUALS Bedingung PRINT Text
```

Beispiele:

```
FOREACH FOUND {
IF EXISTS "<!--YY-directory-->/teaser.html" PRINT
  <a href="<!--YY-directory-->/teaser.html"><!--YY-teaser--></a>
}
```

Hinweis:

Die `PRINT`*-Anweisung ist in diesem Fall Bestandteil der* `IF EXISTS`*-Syntax und fällt nicht unter die in Kapitel 5.3.7 beschriebene Syntax.*

Hier wird geprüft, ob die Datei `<!--YY-directory-->/teaser.html` existiert. Wenn sie existiert, wird der Inhalt der Meta-Variablen `<!--YY-teaser-->` ausgegeben.

Um zu prüfen, ob eine Datei vorhanden ist, wird folgende Syntax verwendet:

```
FOREACH FOUND {
IF NOT EXISTS "<!--YY-directory-->/teaser.html"
PRINT "Die Datei existiert nicht mehr."
}
```

Für `IF (NOT) DEFINED` und `IF (NOT) EQUALS` gilt die analoge Syntax.

5.3.9.2 PERLEVAL: Einzeiligen Perl-Code ausführen

Mit dieser Funktion können Sie eine einzelne Zeile Perl-Code ausführen, ohne das Perl-Plug-In für SiteActive zu verwenden. Der auszuführende Perl-Code muss dabei in einer Zeile hinter dem Schlüsselwort `PERLEVAL` stehen.

```
PERLEVAL Perl-Code
```

PERLEVAL liefert auch den Rückgabewert des Ausdrucks. Verwenden Sie beispielsweise die Funktion `print`, liefert PERLEVAL neben der Ausgabe des Textes noch eine 1 als Rückgabewert zurück. Innerhalb einer FOREACH FOUND-Schleife sollten Sie Text also nicht mit *PERLEVAL* und `print` ausgeben.

Beispiel:

```
PERLEVAL "<!--YY-teaser-->" unless  ("<!--YY-teaser-->" eq "")
```

Der Inhalt der Meta-Variablen `teaser` des entsprechenden Dokuments wird ausgegeben, sofern sie nicht leer ist.

Beispiel für eine IF-Abfrage in PERLEVAL:

```
<tr>
  PERLEVAL "<!--YY-thema-->" !~ /myTopic/ ? '<td class="green">'
  : '<td class="red">'
  <!--YY-titel-->
  </td>
</tr>
```

Hinweis:

Geschweifte Klammern müssen Sie durch einen vorangestellten Backslash maskieren.

Passt der Wert der Meta-Variablen `thema` nicht auf den String `myTopic`, wird der String nach dem Fragezeichen ausgegeben. Stimmen der Wert und der String überein, wird der String nach dem Doppelpunkt ausgegeben.

5.3.9.3 DISRUPT: FOREACH FOUND-Schleifen unterbrechen

```
DISRUPT EVERY Zahl LINES BY Text
DISRUPT END
```

Mit dieser Funktion können Sie in die Ausgabe einer `FOREACH FOUND`-Schleife nach einer bestimmten Anzahl von Treffern immer wieder Text einfügen. Das kann zum Beispiel HTML-Code sein, der ein Bild lädt oder eine horizontale Linie darstellt, mit der die Trefferliste unterteilt wird.

Eine DISRUPT-Anweisung besteht aus einer Startanweisung, die auch den einzufügenden Text enthält, und einer Endanweisung. Die Startanweisung muss vor dem Schlüsselwort `FOREACH FOUND` stehen, die Endanweisung hinter der geschweiften Klammer, mit der die `FOREACH FOUND`-Schleife beendet wird. Beispiel:

```
DISRUPT EVERY 2 LINES BY <td bgcolor="#c0c0c0" height="10px">
 </td>
FOREACH FOUND {
  <tr>
    <td width="180">
      <a href="<!--YY-directory-->/<!--YY-filename-->"><!--YY-title--></a>
      </td>
  </tr>
}
DISRUPT END
```

Mit diesem Code wird die erzeugte Liste nach jedem zweiten Eintrag durch eine gefärbte Tabellenzelle unterbrochen.

Eine `DISRUPT`-Anweisung wird Template weit solange berücksichtigt, bis die `DISRUPT END`-Anweisung gefunden wird.

5.3.9.4 LOOPCOUNT: Anzahl der vollendeten Schleifendurchläufe

```
<!--LOOPCOUNT+Startwert-->
```

Die Anzahl der vollendeten Schleifendurchläufe wird mit Hilfe der Variablen `<!--LOOPCOUNT+`*Startwert*`-->` ermittelt. Diese Variable benötigt einen Startwert, der nach jedem Durchlauf einer `FOREACH FOUND`-Schleife um eins erhöht wird. Beispiel:

```
FOREACH FOUND {
Treffer <!--LOOPCOUNT+1--><br />
  <a href="<!--YY-directory-->/<!--YY-filename-->"><!--YY-title-->
  </a><br />
}
```

5.3.10 IACTIVE: SiteActive-Code auslagern

Wenn Sie den selben SiteActive-Code in mehreren Übersichtsseiten verwenden wollen, die sich nur in einigen wenigen Parametern unterscheiden, bietet sich die Auslagerung des SiteActive Codes in eine externe Datei an.

Erstellen Sie dazu eine Datei, die den SiteActive-Code enthält, und legen Sie diese wahlweise an einer der beiden folgenden Stellen auf dem Zielsystem ab:

1. `pfad/zur/DOCUMENT-ROOT/FILENAME`.
2. `pfad/zum/Imperia-site-Verzeichnis/activelist/FILENAME`.

Um die SiteActive-Anweisungen in Ihrem Template verfügbar zu machen, fügen Sie an der Stelle, an der später im Dokument die extrahierten Inhalte erscheinen sollen, die folgende Anweisung ein:

```
<!--IACTIVE:FILENAME-->
```

Neben dem Namen der Datei mit dem SiteActive-Code kann `FILENAME` dabei auch eine Pfadangabe mit Verzeichnisnamen enthalten. Das ist beispielsweise nützlich, wenn Sie Ihre SiteActive-Include-Dateien in einem eigenen zugriffsgeschützten Verzeichnis in der Document-Root ablegen wollen.

Gegenüber der Anweisung `CODEINCLUDE` hat `IACTIVE` den Vorteil, dass sich so eingebundener Code an der selben Stelle erneut einbinden lässt, nachdem die Ersetzung stattgefunden hat. Ein `CODEINCLUDE` wird einfach durch den Code aus der externen Datei ersetzt. Danach gibt es im betreffenden Dokument keinen Hinweis mehr darauf, dass an dieser Stelle externe Bestandteile eingebunden worden sind.

Wenn Sie hingegen `IACTIVE` verwenden, sieht die betreffende Stelle nach der Ausführung des SiteActive-Codes in etwa folgendermaßen aus:

```
<!--IACTIVE:FILENAME-->
mit SiteActive erzeugter Inhalt
<!--/IACTIVE:FILENAME-->
```

Findet Imperia diese Stelle bei der Ausführung eines Systemdienstes im Dokument, wird sie zunächst wieder durch den Verweis auf den externen SiteActive-Code ersetzt und anschließend expandiert:

```
<!--IACTIVE:FILENAME-->
Inhalt der Datei FILENAME
<!--/IACTIVE:FILENAME-->
```

Danach werden die Anweisungen aus dem Code-Block interpretiert. Auf diese Weise können SiteActive-Template und Zieldatei physikalisch die selbe Datei sein.

5.4 Beispiele für SiteActives

In den folgenden Beispielen werden einige häufig verwendete SiteActive-Funktionen benutzt. Eine Beschreibung aller Funktionen finden Sie in Abschnitt 5.3 **Syntax-Referenz** auf Seite 119.

Die Beispiele verzichten weitgehend auf das Layout der Ausgabe. Das Layout kann im erzeugten HTML innerhalb der FOREACH FOUND-Schleife mit Hilfe von Cascading Stylesheets oder ähnlichen Techniken angepasst werden.

Hinweis:

Sie sollten sicherstellen, dass diese Beispiele nicht auf ein produktives Zielsystem gelangen, auf dem Ihre Website läuft, um Beeinträchtigungen Ihres Projekts zu vermeiden. Wir empfehlen für das Nachprogrammieren der hier aufgeführten Beispiele das Aufsetzen eines Test-Zielsystems.

5.4.1 Voraussetzungen

Alle hier vorgestellten Beispiele benötigen folgende Voraussetzungen:

- Ein SiteActive-Template, das den SiteActive-Code enthält und als Layout-Vorlage für die zu erzeugende Übersichtsseite dient.
- Einen Systemdienst steuert, zu welchem Zeitpunkt bzw. bei welchem Ereignis die Übersichtsseite erzeugt werden soll. Lesen Sie hierzu auch Kapitel **Systemdienste** im Administrationshandbuch.
- Dokumente, aus denen gewünschten Informationen gelesen werden können. Dies geschieht mit Hilfe der YY-Variablen, die auf die entsprechenden Meta-Variablen der gefundenen Dokumente zugreifen (siehe Abschnitt 6.19 **YY-Variablen (nur SiteActive)** auf Seite 146).

5.4.2 Beispiel 1: Einfache Linkliste

Um eine Linkliste aufbauen zu können, wird von jedem Dokument der Pfad und der Titel benötigt. Beide Informationen werden von SiteActive mit Hilfe der YY-Variablen aus den Treffer-Dokumenten gelesen. Hierbei wird der vollständige Pfad mit den Werten der Meta-Variablen `directory`, `filename` gebildet. Der Link-Text soll durch den Titel des Zieldokuments gebildet werden. Hierzu wird die Meta-Variable `title` verwendet.

In diesem Beispiel werden alle Dokumente berücksichtigt, die sich in oder unterhalb des als `READDIR` angegebenen Verzeichnisses befinden. Es gibt keinerlei Einschränkung für die erzeugte Liste.

In der erzeugten HTML-Seite werden die Links durch Zeilenschaltungen voneinander getrennt. Es ist jedoch auch möglich, innerhalb der FOREACH FOUND-Schleife eine Tabelle um eine Zeile pro gefundenem Dokument zu ergänzen (siehe hierzu die FOREACH FOUND-Schleife von Beispiel 2).

Eine einfache Linkliste wird wie folgt erzeugt:

```
<IMPERIA>
  CLEARLIST ❶
  FILEMASK = ".*\.htm[lsx]?$" ❷
  READDIR = "/path/to/documents" ❸
  FOREACH FOUND { ❹
    <a href="<!--YY-directory-->/<!--YY-filename-->"><!--YY-title--></a><br />
  }
</IMPERIA>
```

❶ Eventuell vorhandene Trefferliste und Trefferlimit aus einem vorhergehenden SiteActive-Block löschen.
❷ Regulärer Ausdruck, auf den die Dateinamen der Dokumente passen müssen, damit sie als Treffer gelten.
❸ Verzeichnis bestimmen, in dem rekursiv nach Treffern gesucht werden soll.
❹ Dies ist die Ausgabe-Routine. Der in den geschweiften Klammern angegebene HTML-Code wird für jeden Treffer in die HTML-Seite eingefügt. Dabei werden die YY-Variablen durch die entsprechenden Werte aus den Treffer-Dokumenten ersetzt.

5.4.3 Beispiel 2: limitierte Linkliste

Dieses Beispiel verwendet die gleichen Informationen wie Beispiel 1. Das Auslesen dieser Informationen geschieht wieder mit Hilfe der YY-Variablen.

Für die Trefferliste in diesem Beispiel existieren zwei Einschränkungen, nämlich die Anzahl der anzuzeigenden Links und die Sortierung der Trefferliste. Die Sortierung sorgt dafür, dass die jüngsten Dokumente am Anfang der Liste erscheinen, die maximal 25 Dokumente umfassen soll.

Eine Linkliste mit limitierter Länge kann wie folgt erzeugt werden:

```
<table>
  <IMPERIA>
    CLEARLIST ❶
    FILEMASK = ".*\.htm[lsx]?$"
    READDIR = "/path/to/documents" ❷
    SORT LATEST FIRST ❸
    LIMIT HITS 25  ❹
    LIMIT BY limit_char "A" TO "E" ❺
    FOREACH FOUND { ❻
      <tr>
        <td>
          <a href="<!--YY-directory-->/<!--YY-filename-->"><!--YY-title--></a>
        </td>
      </tr>
    }
  </IMPERIA>
</table>
```

❶ Eventuell vorhandene Trefferliste und Trefferlimit aus einem vorhergehenden SiteActive-Block löschen.
❷ Das Verzeichnis bestimmen, in dem rekursiv nach Treffern gesucht werden soll.
❸ Die Trefferliste so sortieren, dass die jüngsten Treffer in der Liste oben erscheinen.
❹ Die Trefferliste auf fünf Einträge reduzieren.
❺ Die Trefferliste anhand der Meta-Variablen `limit_char` auf die Dokumente begrenzen, in denen diese Meta-Variable einen Wert hat, der zwischen "A" und "E" liegt. Alternativ kann anstelle der Funktion `LIMIT BY` auch die Funktion FILTER (siehe Abschnitt 5.3.5.6 **FILTER: Trefferliste filtern** auf Seite 124) verwendet werden.
❻ Den HTML-Code in den geschweiften Klammern für jeden Treffer in die zu erzeugende HTML-Seite einfügen und in diesem HTML-Code die YY-Variablen durch die Werte aus den Treffer-Dokumenten ersetzen.

5.4.4 Beispiel 3: gefilterte Trefferliste

Neben den Möglichkeiten, die Trefferliste mit `LIMIT` einzuschränken, können auch Filter gesetzt werden. Ein solcher Filter muss allerdings vor dem Befehl `READDIR` definiert werden.

Um eine Linkliste mit Dokumenten zu erzeugen, deren Titel mit dem Buchstaben "A" beginnt, kann folgender SiteActive-Code verwendet werden:

```
<table>
  <IMPERIA>
    CLEARLIST
    FILEMASK = ".*\.htm[lsx]?$"
    FILTER "title" CONTAINS "^A" ❶
    READDIR = "/path/to/documents"
    SORT LATEST FIRST
    FOREACH FOUND {
      <tr>
        <td>
          <a href="<!--YY-directory-->/<!--YY-filename-->"><!--YY-title--></a>
        </td>
      </tr>
    }
  </IMPERIA>
</table>
```

❶ Hier wird der Filter so gesetzt, dass nur Dokumente beachtet werden, deren Meta-Variable `title` mit dem Buchstaben `A` beginnt. Es können Reguläre Ausdrücke verwendet werden.

5.4.5 Beispiel 4: Einfache Übersichtsseite

In diesem Beispiel wird eine einfache Übersichtseite erzeugt, die neben dem Titel auch einen kurzen Teaser-Text zu dem verlinkten Dokument enthält. Zusätzlich dient der Titel als Link zum entsprechenden Dokument.

Voraussetzung für dieses Beispiel ist das Vorhandensein folgender Meta-Variablen in den Dokumente:

- `directory`, `filename`, `title`
- `teaser`

Die Teaser-Texte werden, analog zu Beispiel 2, in eine Tabelle plaziert. Damit die Übersichtsseite nicht zu lang gerät, wird die Anzahl der erlaubten Teaser-Texte auf zehn beschränkt. Weiterhin werden nur die jüngsten Dokumente berücksichtigt.

```
<table>
  <IMPERIA>
    CLEARLIST
    FILEMASK = ".*\.htm[lsx]?$"
    READDIR = "/path/to/documents"
    SORT LATEST FIRST
    LIMIT HITS 10  ❶
    FOREACH FOUND { ❷
      <tr>
```

```
        <td>
          <a href="<!--YY-directory-->/<!--YY-filename-->"><!--YY-title--></a>
        </td>
      </tr>
      <tr>
        <td>
          <!--YY-teaser-->
        </td>
      </tr>
      <tr>
        <td> </td>
      </tr>
    }
  </IMPERIA>
</table>
```

❶ Die Trefferliste auf zehn Einträge reduzieren.
❷ Pro gefundenem Dokument wird die Tabelle wie folgt erweitert:

- eine Zeile mit dem verlinkten Titel
- eine Zeile mit dem Teaser-Text
- eine leere Zeile, die als Trenner zu den Tabellen-Zeilen des nächsten Dokuments dient

5.4.6 Beispiel 5: Einfache Übersichtsseite mit Perl

Wird Perl in einem SiteActive verwendet, müssen andere Schlüsselwörter für die einzelnen SiteActive-Funktionen verwendet werden. In Kapitel Abschnitt 7.3.1 **Syntax-Referenz** auf Seite 154 sind die entsprechenden Schlüsselwörter aufgelistet.

Imperia verwendet für Perl in einem SiteActive ein spezielles Plug-In, dem bestimmte Variablen, Hashes und Listen bereits bekannt sind. Dies sind insbesondere die Liste `@FILELIST`, in der alle Dokumente aufgelistet sind, die beispielsweise durch `filemask()` und `ireaddir()` als Treffer erkannt wurden.

Dieses Beispiel setzt in den Dokumenten das Vorhandensein folgender Meta-Variablen voraus:

- `directory` und `filename` (sind automatisch in jedem Dokument vorhanden)
- `title`
- `teaser`

```
<table border="0" cellspacing="0" cellpadding="0">
<IMPERIA lang=perl> ❶
clearlist();
filemask('.*\.htm[lsx]?$');
ireaddir ('/path/to/documents');

foreach my $file (@FILELIST) {  ❷
  my $directory = $FILE_META_INFO{$file}->getValues('directory'); ❸
  my $filename = $FILE_META_INFO{$file}->getValues('filename');

  my $title = $FILE_META_INFO{$file}->getValues('title');
  my $teaser = $FILE_META_INFO{$file}->getValues('teaser');

      print <<EOF; ❹
       <tr>
         <td><a href="$directory/$filename"><b>$title</b></a></td>
       </tr>
       <tr>
         <td>$teaser</td>
       </tr>
       <tr>
          <td> </td>
       </tr>
EOF
```

```
}
</IMPERIA>
</table>
```

❶ Das `lang`-Attribut des `IMPERIA`-Tags auf den Wert *perl* setzen, damit Imperia Perl als Interpreter für den SiteActive-Code verwendet.

❷ Imperia iteriert in dieser Schleife über alle Dokumente, die in der Liste `@FILELIST` eingetragen sind. Die Variable `$file` referenziert dann bei jeder Iteration den dem aktuellen Dokument entsprechenden Pfad und Dateinamen.

❸ Hier wird der Perl-Variablen `$directory` der Wert der Meta-Variablen `directory` zugewiesen. Dies geschieht mit Hilfe des Hashes `$FILE_META_INFO`, der die MetaInfo-Objekte der Dokumente enthält. Über die Funktion `getValues()`, die als Parameter den Namen der gewünschten Meta-Variablen des Dokuments erhält, wird der Wert dieser Meta-Variablen ausgelesen und in der Variablen gespeichert.

❹ Dies ist die Ausgabe. Innerhalb der Tabellenstruktur werden die Variablen automatisch durch ihren Wert ersetzt und ausgegeben.

5.4.7 Beispiel 6: Übersichtsseite mit Perl und alternierendem Layout

Dieses Beispiel erzeugt eine Übersichtsseite, in der das Layout mit Hilfe einer Counter-Variablen alternierend verändert wird.

```
<IMPERIA lang=perl>
  clearlist();
  filemask('.*\.htm[lsx]?$');
  ireaddir ('/documents');
  my $count = 0; ❶

  foreach my $file (@FILELIST) {
  ❷
    my $url = $FILE_META_INFO{$file}->getValues('directory') . '/' .
    $FILE_META_INFO{$file}->getValues('filename') ;
    my $image = $FILE_META_INFO{$file}->getValues('teaser_img');
    my $title =  $FILE_META_INFO{$file}->getValues('title');
    my $teaser = $FILE_META_INFO{$file}->getValues('teaser');

    if ($count++%2) { ❸
      print <<EOF;
        <tr>
          <td>
            <a href="$url"><b>$title</b></a><br/>
            $teaser
          </td>
          <td>
            <img src="$teaser_img">
          </td>
        </tr>
EOF

    } else {
      print <<EOF;
        <tr bgcolor="#cacaca">
          <td >
            <img src="$teaser_img">
          </td>
          <td>
            <a href="$url"><b>$title</b></a><br/>
            $teaser
          </td>
        </tr>
EOF
    }
  }
</IMPERIA>
```

Hinweis:

Geschweifte Klammern müssen Sie durch einen vorangestellten Backslash maskieren.

❶ Initialisierung einer Counter-Variable, mit deren Hilfe die Alternation des Layouts gesteuert werden wird.
❷ Speicherung der relevanten Meta-Variablen aus den Dokumenten in Perl-Variablen.
❸ Überprüfung der Counter-Variablen `$count`. Ist der Wert von `$count` rein durch zwei teilbar, liefert die Bedingung *wahr* zurück und der erste HTML-Code-Abschnitt wird ausgegeben. Liefert die Bedingung *falsch* zurück, wird der HTML-Code-Abschnitt des `else`-Zweiges ausgegeben.

   Die beiden HTML-Ausgabe-Abschnitte unterscheiden sich lediglich in der Anordnung des Bildes und der Texte.

5.5 SiteActives manuell aufrufen

Wenn Sie ein SiteActive während der Entwicklung testweise ausführen wollen, ist das Triggern des entsprechenden Systemdiensts oft zu umständlich. Auch wollen Sie möglicherweise bei den Probeläufen keine Datei generieren. Hier bietet sich der manuelle Aufruf des SiteActive-Skripts an. Dabei übergeben Sie die Parameter, die normalerweise in der Konfiguration des Systemdiensts stehen, als CGI-Parameter:

```
http://my.site.de/cgi-bin/site_active.pl?TEMPLATE=/siteactive/template.html
```

Der Aufruf kann über die Adresszeile des Browsers oder über einen Link erfolgen. Das Ergebnis des SiteActives erscheint direkt im Browserfenster.

Wenn Sie das Ergebnis eines manuellen SiteActive-Aufrufes in einer Datei speichern wollen, müssen Sie beim Aufruf zusätzlich den Parameter *SAVEPAGE* übergeben. Aus Sicherheitsgründen unterbindet Imperia jedoch per Default manuelle SiteActive-Aufrufe, die Dateien schreiben. Deswegen müssen Sie außerdem noch ein Passwort mit übergeben. Dieses Passwort definieren Sie in der Datei `site/config/system.conf` mit der Variablen `SAVEPAGE-PASSWORD`. Hier ein Beispiel für einen vollständigen Aufruf eines SiteActives aus der Adresszeile des Browsers inklusive SAVEPAGE und Passwort:

```
http://my.site.de/cgi-bin/site_active.pl?TEMPLATE=/siteactive/template.html&
  SAVEPAGE=/dynamic/index.html&SAVEPAGE-PASSWORD=geheimes_Passwort
```

Hinweis:

Das obenstehende Beispiel ist aus Darstellungsgründen umbrochen.

Achtung:

Die Verzeichnisse bzw. das SiteActive-Template, die mit den Parametern `TEMPLATE` *und* `SAVEPAGE` *übergeben werden, liegen unterhalb des Wurzelverzeichnisses (Document-Root).*

5.6 SmartMeta

Mit SmartMeta haben Sie die Möglichkeit, Meta-Variablen in Abhängigkeit von den Werten anderer Meta-Variablen Werte zu zuweisen. Diese Imperia-Funktion steuern Sie über die Datei `smart-meta.conf` aus dem Verzeichnis `/site/config`. Diese Datei muss auf jedem System vorhanden sein, auf dem Sie SmartMeta mit SiteActives einsetzen möchten.

Die Wertzuweisungen betreffen nur das mit SiteActive generierte Dokument. Die Quelldokumente aus der Trefferliste bleiben unverändert.

Hinweis:

Das gleichnamige Workflow-Plug-In bedient sich der selben Funktionalität (siehe Abschnitt **Workflow-Plug-Ins** *aus dem Kapitel* **Workflows** *im Administrationshandbuch). Wertzuweisungen, die dieses Plug-In durchführt, definieren Sie ebenfalls in der Datei* `smart-meta.conf`.

Sobald eine entsprechende Konfigurationsdatei vorhanden ist, prüft SiteActive automatisch bei der Seitengenerierung, ob die darin definierten Bedingungen für die Wertzuweisungen gegeben sind und führt diese dann aus. Dabei findet die Prüfung und Bearbeitung der Metainformationen des SiteActive-Dokuments mit SmartMeta zuerst statt. Erst danach erfolgt die Ausführung der SiteActive-Anweisungen.

5.6.1 SmartMeta konfigurieren

Jede Zeile der `smart-meta.conf` enthält nebeneinander zwei Meta-Variablennamen-Wert-Paare. Das linke Paar bestimmt die Meta-Variable, die als Prüfkriterium für die Wertzuweisung fungiert sowie den Wert, bei dem diese stattfindet. Das rechte Paar bestimmt die zu ändernde Meta-Variable und den zu zuweisenden Wert. Metafeldname und -wert sind jeweils durch => getrennt. Die Syntax einer Zeile der `smart-meta.conf` ist wie folgt:

Meta-Variable1 = Wert1 => Meta-Variable2 = Wert2

Mit dieser Anweisung erreichen Sie, dass `Meta-Variable2` den `Wert2` zugewiesen bekommt, wenn `Meta-Variable1` `Wert1` hat.

Sie können auch mehrere Wertzuweisungen gleichzeitig vornehmen:

Meta-Variable 1 = Wert 1 => Meta-Variable 2 = Wert 2
=> Meta-Variable 3 = Wert 3
=> Meta-Variable 4 = Wert 4

Weitere Beispiele und Erläuterungen finden Sie in der Datei `smartmeta.conf` im Verzeichnis `site/config`.

Kapitel 6. Variablen

Dieses Kapitel gibt Ihnen eine Übersicht über alle Variablen, die in Templates verwendet werden können.

Beachten Sie unbedingt die Reihenfolge der Ersetzung von Variablen beim Parsen der Dokumente. Jede Variable wird zu einem bestimmten Zeitpunkt vom Parser durch den entsprechenden Inhalt ersetzt.

Da ein Dokument möglicherweise von mehr als einem Parser bearbeitet werden kann, besteht bei der Verwendung ungeeigneter Variablen die Gefahr, dass sie bei einem früheren Parsen schon ersetzt wurden und von nachfolgenden Parsern nicht mehr gefunden werden können.

Die Reihenfolge der Ersetzung von Variablen ist wie folgt:

- XX-Variablen
- Variablen vom Typ `<!--ZZ-datum-->`
- TM-Variablen (nur SiteActive)
- YY-Variablen (nur SiteActive)
- FF- und ZZ-Variablen (nur SiteActive)
- MM-Variablen
- UU-Variablen (nur Personalisierung)
- GG-Variablen (nur SiteActive)

Variablen werden in Imperia auf zwei verschiedene Arten aufgerufen:

- <!--*Name-Parameter*-->
- Falls mehrere Parameter zugelassen sind, werden diese durch Doppelpunkte voneinander getrennt. Beispiel:

  <!--*Name-Parameter1*:*Parameter2*-->

Im Folgenden finden Sie eine Liste aller Variablen, die in Imperia 8 verwendet werden:

- CC-Variablen
- dirlevel-Variablen *
- ENV-Variablen
- FF-Variablen (nur SiteActive)
- GG-Variablen
- KK-Variablen
- MEMBER_OF-Variablen
- MM-Variablen
- XX-mode-Variablen
- SECTION-Variablen *
- SECTION_DESCR-Variablen (Shortcut für SECTION:NAME)
- SYSTEM_CONF-Variablen *
- TCOUNTER-Variablen *
- TM-Variablen (nur SiteActive)
- TMDEF-Variablen (auch TMDEFINED)
- USER_CONF-Variablen *

- XX-OBJ-Variablen
- ENV-Variablen
- MEMBER_OF-Variablen
- UU-Variablen (nur Personalisierung)
- GG-Variablen (nur SiteActive)
- Xdir-Variablen *
- Xdate-Variablen *
- Xtime-Variablen *
- XX-Variablen
- XXDEF-Variablen (auch XXDEFINED)
- XXOBJ-Variablen
- YY-Variablen (nur SiteActive)
- ZZ-Variablen (nur SiteActive)

Hinweis:

Für die in dieser Liste mit einem Stern (*) gekennzeichneten Variablen gilt die Doppelpunkt-Syntax (siehe oben).

6.1 CC-Variablen

Nur der Template-Prozessor des Develop-Systems verarbeitet diesen Variablentyp. Die Funktionalität ist ähnlich wie die der XX-Variablen, mit dem Unterschied, dass Sonderzeichen URL-maskiert werden. Es handelt sich also um einen Alias für `XX-URI`.

Jedes Sonderzeichen wird dabei durch ein Triplet der Form `%xx` ersetzt, wobei xx für den hexadezimalen Code des entsprechenden Zeichens steht.

CC-Variablen werden wie XX-Variablen entfernt, wenn die entsprechende Meta-Variable nicht vorhanden ist.

6.2 dirlevel-Variablen

Auch diesenVariablentyp verarbeitet nur der Template-Prozessor des Develop-Systems. Eine Variable vom Typ `dirlevel` liefert einen Teil des Dokumenten-Pfades.

Mit einem Parameter bestimmen Sie, welchen Teil des Pfades die Variable liefert. Dazu ein Beispiel:

Angenommen, ein Dokument liegt unter dem Pfad `/sport/golf/pga-tour`. Die Variable `<!--dirlevel:1-->` enthält dann den Wert *sport*, die Variable `<!--dirlevel:2-->` enthält den Wert `golf` und die Variable `<!--dirlevel:3-->` enthält den Wert `pga-tour`.

6.3 FF-Variablen (nur SiteActive)

FF-Variablen greifen auf Formularvariablen des SiteActive-Templates zu. Handelt es sich bei dem SiteActive-Template um ein Imperia-Dokument, stehen im SiteActive-Code alle Metainformationen des SiteActive-Templates in FF-Variablen zur Verfügung.

Übergabe-Parameter eines Systemdienstes sind ebenfalls mit FF-Variablen abrufbar.

Dazu ein Beispiel:

```
TEMPLATE=/site_active/template.html:SAVEPAGE=/save/page/index.html:
dir=/sport
```

Die Variable <!--FF-dir--> erhält nun den Wert /sport und ist im Dokument verwendbar. Dadurch ist beispielsweise folgendes möglich:

```
<IMPERIA>
  READDIR "<!--FF-dir-->"
</IMPERIA>
```

Die Variable dir aus der Konfiguration des Systemdienstes dient hier dazu, im SiteActive-Code das READDIR zu setzen.

Achten Sie auf die korrekte Groß- und Kleinschreibung. Die Variable foo ist eine andere als die Variable Foo. Weiterhin ist es möglich, bei der Auswertung eines Templates oder Dokuments zu einer FF-Variablen Werte zu addieren oder zu subtrahieren:

<!--FF-*Name*+*Zahl*-->
<!--FF-*Name*-*Zahl*-->

Gibt es kein Formularattribut mit dem entsprechenden Namen, erhält die FF-Variable einen leeren String als Wert. Die Expandierung von FF-Variablen findet vor der Auswertung von IF-Abfragen statt. Daher können Sie FF-Variablen auch innerhalb von IF-Abfragen verwenden.

6.4 GG-Variablen

Diesen Variablentyp können Sie nur innerhalb von SiteActives und für die Personalisierung verwenden. Er stellt die Auswertungen von <IMPERIA lang=perl>-Blöcken im Template zur Verfügung. Die Syntax <!--GG-*Name*--> wird durch den Wert Name des Hashes %$GLOBAL ersetzt.

Die Ergebnisse eines <IMPERIA lang=perl>-Blockes lassen sich mit Hilfe folgender Syntax abspeichern:

$GLOBAL->{*Name*}= *berechneter Wert*

Innerhalb eines Templates können Sie nun mit Hilfe der Variablen <!--GG-*Name*--> auf diesen gespeicherten Wert zugreifen.

GG-Variablen sind nicht in IF-Abfragen nutzbar, da ihre Expandierung erst nach deren Auswertung passiert. In FOREACH FOUND-Schleifen lassen sie sich jedoch anwenden.

6.5 KK-Variablen

Dieser Variablentyp dient ausschließlich zur Eingabe von Metainformationen in einem MetaEdit-Schritt. Eingegebene Metainformationen sind am Ende dieses Workflow-Schrittes mit KK-Variablen referenzierbar.

Wenn Sie beispielsweise ein Eingabefeld *Name* definieren, ist dieses über die Variable <!--KK-*Name*--> verfügbar. Die Ersetzung erfolgt beim Verlassen des MetaEdit-Plug-Ins mit **Speichern**. Gibt es für eine Variable kein Eingabefeld, bzw. trägt der Benutzer dort nichts ein, erhält die KK-Variable einen Leerstring als Wert.

Beispiel:

Bei einer mehrsprachigen Website stehen für ein Dokument vier Template-Varianten zur Auswahl. Der Benutzer wählt bei der Eingabe der Metainformationen eine davon für die Erzeugung seines Dokuments aus. Mit dieser Auswahl soll er automatisch die Templates für die Copy-Seiten bestimmen. Dazu könnten Sie in der Metadatei folgenden Code verwenden:

```
SELECTION "template:Bitte w&auml;hlen Sie ein Template"
        OPTION "tpl01:Variante 1"
        OPTION "tpl02:Variante 2"
        OPTION "tpl03:Variante 3"
        OPTION "tpl04:Variante 4"
ENDSEL

HIDDEN "copy1:<!--XX-directory-->/index_en.html:TEMPLATE=
```

```
        <!--KK-template-->_en"
HIDDEN "copy1:<!--XX-directory-->/index_fr.html:TEMPLATE=
        <!--KK-template-->_fr"
HIDDEN "copy1:<!--XX-directory-->/index_de.html:TEMPLATE=
        <!--KK-template-->_de"
```

Hinweis:

Die Hidden-Anweisungen in obenstehendem Code-Beispiel sind aus Darstellungsgründen umbrochen.

Mit der Selectbox weist der Benutzer der Variablen einen Wert zu. Dieser ist am Ende des Meta-Edit-Schritts mit `<!--KK-template-->` abrufbar. Mit XX-Variablen lassen sich in diesem Workflow-Schritt nur bereits zuvor gesetzte Variablen referenzieren. In obigem Beispiel ist dies der Verzeichnisname, bei dem der Zugriff über `<!--XX-directory-->` erfolgt.

Ein weiteres Beispiel:

Statt einer Template-Auswahl könnten Sie auch unterschiedliche Selectboxen für die Rubrik und die Priorität des Artikels anbieten. Anhand dieser Informationen ließe sich dann der Name des zu verwendenden Templates bilden:

```
HIDDEN "template:<!--KK-rubrik--><!--KK-prio-->"
```

Zusätzlich könnten Sie vorher "Konstanten" deklarieren, um die Verbindung zwischen der Kombination aus Rubrik und Dringlichkeit und der Templatenummer herzustellen:

```
HIDDEN "boerse01:0150"
HIDDEN "boerse02:0160"
HIDDEN "neues01:0170"
HIDDEN "neues02:0170"
HIDDEN "sport01:0150"
HIDDEN "sport02:0160"
```

Die Bestimmung des zu verwendenden Templates geschieht dann wie folgt:

```
HIDDEN "template:<!--KK-<!--KK-rubrik--><!--KK-dringl-->-->"
```

6.6 MM-Variablen

Dieser Variablentyp bietet wie YY-Variablen Zugriff auf die Metafeldinhalte eines Quelldokuments für SiteActive-Seiten. Jedoch erfolgt die Ersetzung von MM-Variablen vor der von YY-Variablen. Das ermöglicht Ihnen in SiteActives Konstrukte wie das folgende:

```
<!--YY-<!--MM-rubrik--><!--MM-dringl-->-->
```

Nachdem die beiden MM-Variablen expandiert sind, ergibt sich daraus der Name der YY-Variablen. Diese wird danach eingesetzt. Voraussetzung ist natürlich, dass eine Meta-Variable mit dem zusammengesetzten Namen existiert.

6.7 SECTION-Variablen

Mit diesem Variablentyp greifen Sie auf die Rubrik-Informationen des aktuell verwendeten Templates zu.

Die Syntax einer SECTION-Variable ist wie folgt:

```
<!--SECTION:Parameter>
```

Mögliche Parameter sind:

`DESCR`
: Liefert die Beschreibung der Rubrik.

`NAME`
: Liefert den Namen der Rubrik.

`DIRECTORY`
: Liefert das Standard-Verzeichnis der Rubrik

`TEMPLATE`
: Liefert das Standard-Template der Rubrik.

`FILENAME`
: Liefert den Standard-Dateinamen der Rubrik.

META_INFO_Metafeldname, bzw. Metafeldname
: Liefert die im Feld *Metafeldname* hinterlegten Rubrik-Metainformationen.

Die Einstellung dieser Parameter nehmen Sie in der Rubriken-Verwaltung vor. Sie können die dort gesetzten Werte jedoch durch Workflow-Plug-Ins oder durch Benutzereingaben in der Metadatei oder im Template überschreiben lassen.

Zusätzlich können Sie auch Daten einer übergeordneten Rubrik auslesen. Hierzu müssen Sie die Ebene der gewünschten Rubrik als zweiten Parameter übergeben. Auf die Hauptrubrik greifen Sie dabei mit dem Wert *1* zu. Beispielsweise lesen Sie auf die Beschreibung der Hauptrubrik mit folgender Syntax aus:

```
<!--SECTION:DESCR:1-->
```

Aus Gründen der Abwärtskompatibilität ist die Syntax `<!--SECTION_DESCR-->`, die Ihnen ebenfalls die Beschreibung der Rubrik liefert, weiterhin verwendbar. In neuen Projekten sollten Sie sie jedoch nicht mehr verwenden.

6.7.1 Zugriff auf Metainformationen von Rubriken

Wie auch im Abschnitt **Rubriken bearbeiten** im Kapitel **Struktur** des Administrationshandbuchs beschrieben, können Sie in Imperia auch Rubriken mit Metainformationen versehen. Diese sind in allen Dokumenten der Rubrik sowie denen etwaiger Unterrubriken abrufbar. Der Zugriff erfolgt über eine Section-Variable:

```
<!--SECTION:META_INFO_Metafeldname-->

<!--SECTION:Metafeldname-->

<!--SECTION:Metafeldname[index]-->
```

Das Präfix *META_INFO_* ist hierbei optional und dient lediglich der besseren Unterscheidung von dokumentintern gültigen Meta-Variablen. Enthält das betreffende Metafeld ein Array, geben Sie hinter dem Metafeldnamen in eckigen Klammern den Index des gewünschten Elements aus dem Array an.

6.8 SYSTEM_CONF-Variablen

Mit Hilfe dieses Variablentyps können Sie alle in der Datei `/site/config/system.conf` definierten Systemeinstellungen abfragen. Die Syntax für diesen Variablentyp ist folgendermaßen:

```
<!--SYSTEM_CONF:Parameter-->
```

Achtung:

Die alte Imperia-Syntax mit Minuszeichen anstelle des Unterstrichs wird nicht mehr unterstützt.

Der Platzhalter **Parameter** steht hierbei für eine beliebige Konfigurationsvariable aus der Datei `/site/config/system.conf`. Ist dort ein Wert für die entsprechende Variable definiert, bekommen Sie diesen zurückgeliefert. Anderenfalls erscheint ein leerer String. Sie können beispielsweise folgende Werte abfragen:

`IMPERIA-VERSION`
: Liefert die Programmversion.

`REG_NAME`
: Liefert den Namen, unter dem Imperia lizenziert ist. Dieser Name erscheint auch am unteren Rand des Hauptmenüs.

`LANGUAGE-SYSTEM`
: Liefert die Standard-Sprache des Imperia-Systems.

`OPERATING-SYSTEM`
: Liefert den Typ des Betriebssystems, auf dem Imperia installiert wurde.

`SITE-DIR`
: Liefert den kompletten Pfad des SITE-Verzeichnisses.

`CGI-DIR`
: Liefert den kompletten Pfad des CGI-BIN-Verzeichnisses.

Prinzipiell können Sie alle Einstellungen aus der Datei `/site/config/system.conf` abfragen. Eine komplette Liste der verfügbaren Variablen finden Sie im Anhang des Administratorhandbuchs.

Auf SYSTEM-CONF-Variablen kann Escaping angewandt werden (siehe Abschnitt 6.21 **Escaping Modes** auf Seite 149)

6.9 TCOUNTER-Variablen

Dieser Variablentyp steht nur auf dem Develop-System für einfache arithmetische Rechenoperationen zur Verfügung. Die Syntax für diese Variable ist wie folgt:

`<!--TCOUNTER:`*Meta-Variable Operator Wert*`-->`

Die Expandierung von TCOUNTER-Variablen findet noch vor der von XX-Variablen statt. Das ermöglicht Ihnen beispielsweise Konstrukte folgender Form:

```
<!--XX-text<!--TCOUNTER:an+1-->-->
```

6.10 TM-Variablen (nur SiteActive)

Nutzen Sie TM-Variablen, um auf die Meta-Informationen des SiteActive-Templates zuzugreifen. Variablen dieses Typs liefern einen Leerstring zurück, wenn die entsprechende Meta-Variable nicht existiert.

Wichtig:

Um TM-Variablen nutzen zu können, müssen Sie das SiteActive-Template als Dokument in Imperia anlegen. Nur so können Sie über diesen Variablentyp referenzierbare Informationen darin abspeichern.

Dazu ein Beispiel:

Bei der Erstellung des SiteActive-Templates mit Imperia speichern Sie ein Verzeichnis in der Meta-Variablen mit dem Namen `dir`.

Nun ist Folgendes möglich:

```
<IMPERIA>
  READDIR "<!--TM-dir-->"
  ...
</IMPERIA>
```

Bei der Verarbeitung dieses IMPERIA-Blocks ersetzt Imperia die Variable `<!--TM-dir-->` durch das Verzeichnis, das Sie bei der Erstellung des SiteActive-Templates für die Variable `dir` angegeben haben. SiteActive führt nun also die Anweisungen des IMPERIA-Blocks aus und sucht in diesem Verzeichnis sowie in allen enthaltenen Unterverzeichnissen nach Treffern. Sollten Sie jedoch vergessen haben, eine Variable namens `dir` zu definieren, erscheint an dieser Stelle ein Leerstring und das SiteActive durchsucht keine Verzeichnisse.

Alternativ könnten Sie das SiteActive-Template direkt in das zu durchsuchende Verzeichnis freischalten. In diesem Fall können Sie als `READDIR` einfach `<!--TM-directory-->` angeben.

6.11 TMDEF-Variablen

Dieser Variablentyp ist nur in mit Imperia gepflegten SiteActive-Templates verfügbar und liefert `wahre` Werte (Ziffern) oder `falsch` (Ziffer 0) zurück. Existiert im SiteActive-Template eine entsprechende Meta-Variable und ist sie mit einem nicht leeren Text oder Wert belegt, ist auch die TMDEF-Variable *wahr*. In diesem Fall liefert die Abfrage die Anzahl der Zeichen des Metafeldinhalts. Ansonsten enthält sie den Wert *falsch* (Ziffer 0).

Sie können damit überprüfen, ob ein Feld existiert bzw. nicht leer ist.

Beispiel:

```
FOREACH FOUND {
        <div class="inner">
                IF "<!--TMDEF-category-->" PRINT <!--TM-category-->:
                <a href="<!--YY-directory-->/<!--YY-filename-->"><!--YY-title--></a><br />
                <span class="ant"><!--YY-headline_0_0--> =>> mehr</span>
        </div>
}
```

Obenstehender SiteActive-Code gibt vor den Elementen einer Trefferliste den Rubriknamen des SiteActives aus, sofern dieser definiert ist.

6.12 USER_CONF-Variablen

Die in der User-Verwaltung gespeicherten Daten des aktuell im System angemeldeten Benutzers können Sie mit Hilfe dieser Variablen auslesen. Gemeint ist hierbei nicht der Autor des Dokuments, sondern der Benutzer, der das Dokument im Moment bearbeitet.

Mit folgender Syntax fragen Sie Benutzer-Variablen ab:

```
<!--USER_CONF:Wert-->
```

Achtung:

Die alte Imperia-Syntax mit Minuszeichen anstelle des Unterstrichs ist nicht mehr gültig.

Sie können folgende Werte abfragen:

Variable	Beschreibung
homepage	Startseite
country	Land
telnumber	Telefonnummer
language	Sprache
cellular	Mobiltelefon-Nummer
name	Name
fname	Vorname
city	Stadt
faxnumber	Faxnummer
comment	Kommentar
zip	PLZ
login	Anmeldekennung/Login
email	E-Mail-Adresse
street	Straße

Tabelle 6.1. Felder der Userverwaltung

Auf USER-CONF-Variablen kann Escaping angewandt werden (siehe Abschnitt 6.21 **Escaping Modes** auf Seite 149)

6.13 Xdir-Variablen

Dieser Variablentyp ist nur aus Kompatibilitätsgründen zu früheren Imperia-Versionen weiterhin gültig. Sie sollten ihn in neuen Projekten nicht mehr verwenden. Nutzen Sie stattdessen die Variable `dirlevel` (siehe Abschnitt 6.2 **dirlevel-Variablen** auf Seite 138).

6.14 Xdate-Variablen

Dieser Variablentyp liefert Ihnen verschiedene Datumsformate. Die Syntax ist wie folgt:

`<!--Xdate:`*Format*`-->`

Geben Sie anstelle des Platzhalters *Format* eines der folgenden Datumsformate an:

Formatname	Ausgabe
default	15.08.2001
full	15.08.2001
inverse	2001-08-15
afiles	2001-08-15 22:13
iso	20010815
normal	15.08.2001
american	15.08.2001
imperia	15.08.2001 22:03
finddate	2001.08.15 22:03

Tabelle 6.2. Datums-Parameter

6.15 Xtime-Variablen

Dieser Variablentyp liefert verschiedene Uhrzeitformate. Die Syntax ist folgendermaßen:

```
<!--Xtime:Format-->
```

Verwenden Sie anstelle des Platzhalters *Format* eines der folgenden Uhrzeitformate:

Format	Ausgabe
default	22:14
normal	22:14:18
full	22:14:18
compact	22:14
hour	22
minute	14
second	18

Tabelle 6.3. Uhrzeitformate

6.16 XX-Variablen und XXOBJ-Variablen

Diese Variablen lassen sich sowohl im MetaEdit- als auch im Bearbeiten-Schritt eines Workflows bearbeiten. Es findet grundsätzlich eine Ersetzung statt. Das heißt, ein Ausdruck in der Form <!--XX-*Name*--> bleibt auf keinen Fall in dieser Form im erzeugten HTML-Code des Dokuments bestehen. Er liefert entweder den Inhalt des entsprechenden Metafelds oder einen Leerstring, wenn die entsprechende Meta-Variable nicht existiert bzw. nicht gesetzt ist.

XX-Variablen können außer im MetaEdit- oder Edit-Plug-In auch innerhalb von <IMPERIA></IMPERIA>-Blöcken verwendet werden. Dies gilt allerdings nur, wenn das SiteActive-Template ein Imperia-Dokument ist. Sie werden dort allerdings nicht immer ersetzt, sondern nur, wenn tatsächlich die entsprechende Meta-Variable existiert. Existiert die Meta-Variable nicht, bleibt der Ausdruck unverändert und erscheint auch in der fertigen HTML-Seite.

Mit <!--XX-*Meta-Variable*--> können Sie alle Metainformationen abrufen, die in einem Dokument vorhanden sind. Sobald Sie ein Dokument speichern, auch schon in der Dokument-Vorschau, werden XX-Variablen durch den Wert der entsprechenden Meta-Variablen ersetzt. Enthält die betreffende Meta-Variable eine Liste von Werten, können Sie hinter dem Metafeldnamen den Index des gewünschten Elements angeben: <!--XX-*Meta-Variable[index]*-->.

Handelt es sich bei der Metainformation, die Sie abrufen wollen, um ein Medienobjekt, das Sie entweder aus der Mediendatenbank oder dem Grafiker-Upload eingefügt haben, müssen Sie statt <!--XX-*Meta-Variable*--> das Tag <!--XXOBJ-*Meta-Variable*--> verwenden. Imperia setzt bei diesem Variablentyp den Pfad zu der entsprechenden Mediendatei ein und löst diesen im PREVIEW- und SAVE-Modus des Dokuments auf.

6.16.1 Modifier für XX-Variablen

Stammen die zu referenzierenden Metafelder aus einem Flexmodul, Imperiablock oder Arrayblock, müssen Sie vor dem Metafeldnamen einen Modifier einfügen, um dies zu kennzeichnen. Dies geschieht mit folgender Syntax:

```
<!--XX-MODIFIER-Metafeldname-->
```

Folgende Modifier kennt Imperia:

Modifier	Bedeutung
SLOT	Modifier für Metafelder aus Slotmodulen.
FLEX	Modifier für Flexmodulinhalte.
IBLOCK	Das Metafeld liegt in einem Imperiablock.
ARRAY	Verwenden Sie diesen Modifier für Metafelder aus Arrayblöcken.

Tabelle 6.4. Modifier für XX-Variablen

Durch den Modifier ergänzt Imperia den Metafeldnamen automatisch durch die entsprechenden Index- und ID-Nummern. Das gilt allerdings nur, wenn der Zugriff innerhalb des betreffenden Elements erfolgt. Wollen Sie beispielsweise in einem Template außerhalb eines Flexmoduls auf dessen Inhalte zugreifen, müssen Sie Flex-Index und Flex-ID des betreffenden Moduls an den Metafeldnamen anhängen. Lesen Sie hierzu auch Abschnitt 4.7 **Inhalt außerhalb eines Flexmoduls verwenden** auf Seite 111.

6.17 XXDEFINED-Variablen

Dieser Variablentyp wird sowohl vom MetaEdit- als auch vom Edit-Plug-In bearbeitet und liefert die Werte 1 oder 0 zurück. XXDEF-Variablen nehmen den Wert 1 an, wenn die entsprechende Meta-Variable existiert und mit einem nichtleeren Text oder Wert belegt ist. Ansonsten enthalten sie den Wert 0. Sie werden verwendet, um zu prüfen, ob eine Meta-Variable existiert bzw. nicht leer ist.

Hinweis:

In einer #IF-Abfrage im Template sind Anführungszeichen für XXDEFINED-Variablen nicht erforderlich.

6.18 XX-Modus-Variablen

In der Variablen XX-Modus ist der Modus gespeichert, in dem sich ein Dokument befindet. Sie kann folgende Werte haben:

- EDIT
- SAVE
- PREVIEW
- REPARSE

Lesen Sie zu diesem Thema auch Abschnitt 3.4.5 **Modus abfragen und überprüfen** auf Seite 55.

6.19 YY-Variablen (nur SiteActive)

Diese Variablen werden in FOREACH-FOUND-Schleifen in SiteActive-Templates verwendet. Darüber kann auf die Meta-Informationen der gefundenen Dokumente zugegriffen werden.

Beispiel:

Angenommen, die Datei `index.html` liegt im Verzeichnis `/sport/segeln/00333`. Um nun per SiteActive einen Link auf diese Seite zu erhalten, wird innerhalb der FOREACH-FOUND-Schleife folgender Code eingegeben:

```
<a href="<!--YY-directory-->/<!--YY-filename-->"><!--YY-title--></a>
```

Beim Durchlauf wird für jedes gefundene Dokument `<!--YY-directory-->/<!--YY-filename-->` mit dessen Verzeichnispfad ersetzt, so dass am Ende in dem mit SiteActive erzeugten Dokument ein gültiger Link ausgehend vom Document-Root entsteht. Der Linktext wird aus der Ersetzung von `<!--YY-title-->` durch den Dokumententitel gebildet.

Mit Hilfe von <!--YY-*Meta-Variable*--> kann innerhalb einer FOREACH-FOUND-Schleife auf jede Meta-Variable eines gefundenen Dokuments zugegriffen werden.

In YY-Variablen kann, ähnlich wie in Templates, eingestellt werden, ob und wie der Inhalt der Variablen maskiert werden soll. Hierzu wird der Aufruf der YY-Variable wie folgt ergänzt:

`<!--YY-TEXT:`*Meta-Variablenname*`-->`

Mit dieser Syntax wird der Inhalt der referenzierten Meta-Variablen maskiert, so dass beispielsweise das Zeichen "<" als "<" angezeigt wird.

Weiterhin kann angegeben werden, welche Zeichen escaped werden sollen. Andere Zeichen werden dann nicht escaped. Hierzu wird der Aufruf der YY-Variablen wie folgt ergänzt:

`<!--YY-TEXT:(':"):`*Meta-Variablenname*`-->`

Die Zeichen, die escaped werden sollen, werden in runden Klammern durch Doppelpunkte voneinander getrennt geschrieben. Mit der obigen Syntax würden die Zeichen ' und " escaped. Anstelle der Zeichen können auch die entsprechenden ASCII-Codes angegeben werden:

`<!--YY-TEXT:(34:39):`*Meta-Variablenname*`-->`

Beispiel:

Der Inhalt der referenzierten Meta-Variablen `test` ist ">, <, ", &". Die FOREACH FOUND-Schleife im SiteActive-Template ist wie folgt:

```
FOREACH FOUND {
  <tr>
    <td><!--YY-test--></td>
    <td><!--YY-TEXT:test--></td>
    <td><!--YY-TEXT(<:>):test--></td>
  </tr>
}
```

Die von SiteActive erzeugte HTML-Seite enthält nun folgenden Quelltext:

```
<tr>
  <td>>, <, ", &</td>
  <td>&gt;, &lt;, &quot;, &amp;</td>
  <td>&#62;, &#60;, ", &</td>
</tr>
```

6.20 ZZ-Variablen (nur SiteActive)

Mit Hilfe von ZZ-Variablen kann auf Datum und Uhrzeit zugegriffen werden.

Standardmäßig geben ZZ-Variablen Zeit und Datum in Greenwich Mean Time (GMT) wieder. Ein korrekter Vergleich mit Zeitvariablen, die die Lokalzeit angeben, ist damit nicht möglich. Stattdessen muss XstrftimeGMT zum Vergleich herangezogen werden.

Zusätzlich können einfache Berechnungen durchgeführt werden. Die Syntax einer ZZ-Variable ist wie folgt:

`<!--ZZ-`*Format*`-->`

Anstelle des Platzhalters *Format* kann eines der folgenden Formate angegeben werden:

day

Liefert den aktuellen Tag im Monat als Zahl.

mon

Liefert den aktuellen Monat als Zahl mit führenden Nullen bei einstelligen Monatszahlen. Beispiel: `08` für August.

month/fullmonth

Liefert den aktuellen Monat als Text.

fullyear

Liefert das aktuelle Jahr als vierstellige Zahl.

shortyear

Liefert das aktuelle Jahr als zweistellige Zahl.

year

Liefert die Anzahl der Jahre seit 1900. Dieses Format sollte nur zu Berechnungen verwendet werden.

time

Liefert die Uhrzeit im Format HH:MM.

longtime

Liefert die Uhrzeit im Format HH:MM:SS.

hour

Liefert die Stunden der aktuellen Uhrzeit.

min

Liefert die Minuten der aktuellen Uhrzeit.

sec

Liefert die Sekunden der aktuellen Uhrzeit.

dayofweek

Liefert den Wochentag als Text.

Feld(+|-)*Einheit*

Mit diesem Format können Sie zu einem der oben beschriebenen Formate eine Einheit addieren oder subtrahieren. Beispiel:

```
<!--ZZ-time+58MINS-->
```

In diesem Beispiel werden zur aktuellen Zeit 58 Minuten addiert.

Mögliche Einheiten sind:

Einheit	Beschreibung
SECS	Sekunden
MINS	Minuten
HOURS	Stunden
DAYS	Tage
WEEKS	Wochen
MONTHS	Monate
YEARS	Jahre

Tabelle 6.5. Einheiten

6.21 Escaping Modes

Wird Inhalt aus Variablen referenziert, kann dieser mit Hilfe von Escaping Modes modifiziert werden. Dadurch werden die Metainformationen an sich nicht verändert, sondern nur die Darstellung der Metainformationen im ausgegebenen Dokument.

Hinweis:

Wird bei der Referenzierung kein Modus angegeben, verwendet das System den Standardmodus RAW.

Wird ein Escaping angewandt, prüft Imperia zuerst unter site/modules/core/Dynamic/Escape/ nach, ob es ein entsprechendes Plug-In (z.B. JS.pm) gibt. Falls ja, würde das Plug-In verwendet, falls nein, wird einer der fest eingebauten Modi verwendet. Das sind:

- RAW
- HTML
- HTMLBR
- TEXT
- TEXTBR
- META
- FILESIZE
- JS
- URI

Manche Modi (derzeit TEXT, TEXTBR und FILESIZE) können mit Argumenten aufgerufen werden, z.B. so:

```
<!--XX-TEXT(':":60:38):attrib-->
```

6.21.1 RAW (Standardmodus)

- Eingegebener HTML-Code wird nicht escaped, verändert also das Layout der fertigen HTML-Seite.
- Gespeicherte Zeilenschaltungen werden nicht angezeigt.

6.21.2 HTML

Dieser Modus wurde aus Kompatibilitätsgründen für Imperia-Versionen <= 6.0.2 implementiert. Er hat die gleichen Eigenschaften wie der RAW-Modus.

6.21.3 HTMLBR

- Eingegebener HTML-Code wird nicht escaped, verändert also das Layout der fertigen HTML-Seite.
- Gespeicherte Zeilenschaltungen werden in
-Tags umgewandelt und in der fertigen HTML-Seite angezeigt.

6.21.4 TEXT

- Eingegebener HTML-Code wird escaped.
- Gespeicherte Zeilenschaltungen werden nicht angezeigt.

```
<!--XX-TEXT(':":60:38):attrib-->
```

In diesem Beispiel soll das Metafeld "attrib" ausgegeben werden, wobei einfache und doppelte Anführungszeichen sowie die Zeichen mit den Codes 60 (Kleiner-als-Zeichen)und 38 (Ampersand) durch die jeweiligen HTML-Entities ersetzt werden.

Werden beim Aufruf von TEXT keine Argumente mitgegeben, so werden einfache und doppelte Anführungszeichen, Kleiner-als, Größer-als und Ampersand durch ihre jeweiligen HTML-Entities ersetzt.

6.21.5 TEXTBR

- Eingegebener HTML-Code wird escaped.
- Gespeicherte Zeilenschaltungen werden in
-Tags umgewandelt und in der fertigen HTML-Seite angezeigt.

```
<!--XX-TEXTBR(':"):title-->
```

In diesem Beispiel soll das Metafeld "title" ausgegeben werden, wobei einfache und doppelte Anführungszeichen durch die jeweiligen HTML-Entities ersetzt werden.

Werden beim Aufruf von TEXTBR keine Argumente mitgegeben, so werden einfache und doppelte Anführungszeichen, Kleiner-als, Größer-als und Ampersand durch ihre jeweiligen HTML-Entities ersetzt.

6.21.6 FILESIZE

Dieser Escaping-Modus wandelt Dateigrößen, die als einfache Zahl vorliegen, in gut menschenlesbare Formate wie 7 kB oder 3.5 GB.

```
<!--XX-FILESIZE(digits=1 lingua=de no_si=1):filesize-->
```

Dabei bedeuten die Parameter im Einzelnen:

- digits: Anzahl der gewünschten Nachkommastellen. Default ist "1".
- lingua: Gibt an, in welchem Sprachformat das Ergebnis formatiert werden soll. Dies betrifft hier das Dezimaltrennzeichen.
- no_si: Standardmäßig rechnet FILESIZE mit SI-Einheiten, d.h. 1 Kilobyte (kB) = 1.000 Byte. Mit dem Setzen des Parameters no_si auf einen Wert ungleich null wird die Umrechnung in Zweierpotenzen vorgenommen, d.h. 1 Kilobyte (kB) = 1.024 Byte.

6.21.7 JS (Javascript-String)

- Eingegebener Code wird so escaped, dass er sicher an einen Javascript-String übergeben werden kann.
- Gespeicherte Zeilenschaltungen werden als \n ausgegeben.

```
<script type="text/javascript">
var eingabe="<!--XX-JS:eingabe-->";
alert(eingabe);
</script>
```

6.21.8 URI

Der eingegebene Text wird entsprechend dem MIME-Typ application/x-www-form-urlencoded escaped. Beispiel:

- eingegebener Text: `dies soll <b>fett </b>-gedruckt werden.`
- ausgegebener Text: `dies%20soll%20%3cb%3efett%3c%2fb%3e%2dgedruckt%20werden`

Achtung:

Die hier beschriebenen Modi sind nicht mit den Modi für die Programmierung von Input-Felder und Textareas in einem Template zu verwechseln. Wird bei der Programmierung des Templates ein Input-Feld oder eine Textarea mit einem ungültigen Modus vereinbart, kann auf den Inhalt der resultierenden Meta-Variablen nicht zugegriffen werden. Lesen Sie hierzu die Hinweise in den Abschnitten Abschnitt 3.3.3 **Input-Feld** *auf Seite 31 und Abschnitt 3.3.4* **Textarea** *auf Seite 31.*

Kapitel 7. Personalisierung

Dieses Kapitel wendet sich ausschließlich an erfahrene Entwickler. Es ist auf jeden Fall empfehlenswert, für die Personalisierung den Professional Service der Imperia AG in Anspruch zu nehmen.

In den folgenden Abschnitten werden die Funktionen zur Personalisierung einer Website mit Imperia 8 beschrieben.

7.1 Was ist Personalisierung?

Unter Personalisierung versteht man das auf den jeweiligen Besucher abgestimmte Anzeigen von Inhalten, basierend auf den über diesen Besucher gespeicherten Informationen.

Je umfangreicher die auf einer Website angebotenen Informationen sind, desto schwieriger wird es für einen Besucher, die für ihn interessanten Informationen zu finden. Zwar kann eine gute Struktur der Website und sinnvolle Such- und Überblicksfunktionen dies vereinfachen. Der Besucher muss die Handhabung dieser Hilfsmittel jedoch erst erlernen und sie bei jedem folgenden Besuch erneut anwenden. Falls sich die interessanten Informationsquellen an verschiedenen Stellen innerhalb der Website befinden, ist eine Navigation von einer Quelle zur anderen unvermeidlich. Hier setzt die Personalisierung an.

Eine personalisierte Website erlaubt es dem Besucher, interessante Informationen übersichtlich auf einer Seite zusammenzustellen und ohne zusätzliche Navigation zu erfassen. Der Aufwand, den der Besucher für eine wirkliche Nutzung der Website erbringen muss, wird verringert. Eine personalisierte Website ist von persönlich definiertem Nutzen, der nicht in erster Linie durch die Vorgaben des Webmasters bestimmt wird.

Notwendig für solche dynamischen Websites sind unter anderem folgende Punkte:

- Der bereitgestellte Content muss eine userdefinierte Zusammenstellung erlauben; Inhalt und Layout dürfen nicht unmittelbar voneinander abhängen.
- Die persönlichen Einstellungen eines Besuchers müssen über mehrere Sitzungen hinweg gespeichert werden.

Personalisierung mit Imperia 8 wird durch die Verwendung der folgenden zusätzlichen Imperia-Module verwirklicht:

- Dynamische Seitengenerierung
- Session-Management
- Profil-Datenbank
- Content-Datenbank
- Template-Interpreter

Diese Module und die Möglichkeit, innerhalb von Templates Perl-Code ausführen zu lassen, geben Ihnen die Möglichkeit, jedem Besucher einer personalisierten Website auf ihn zugeschnittenen Content mit passendem Layout anzubieten.

Meldet sich ein Besucher an, erhält er automatisch eine Session-ID, die in der URL während des gesamten Besuchs mitgeführt wird. Die Session-ID wird entweder einem in einer vorherigen Session gesetzten Cookie entnommen, dem Besucher nach Anmeldung über ein Formular anhand des gespeicherten Profils zugewiesen oder neu generiert.

Betritt ein Besucher eine personalisierte Website wird geprüft, ob der Besucher identifiziert werden kann. Hierzu wird ein Cookie abgefragt, das die Session-ID einer früheren Session enthält.

Kann ein Besucher nicht identifiziert werden, zum Beispiel, weil er Cookies nicht erlaubt oder zum ersten Mal die Seite besucht, wird beim Anmelden ein neues Profil und damit eine neue Session-ID erzeugt.

Wird das Cookie gefunden oder identifiziert sich der User über ein Anmeldeformular, wird das zur Session-ID gehörige Profil aus der Profil-Datenbank geöffnet und der Inhalt entsprechend den dort gespeicherten Einstellungen angeboten. Gleichzeitig wird das Cookie aktualisiert bzw. gesetzt.

7.2 Die Aufgaben der einzelnen Module

7.2.1 Dynamische Seitengenerierung

Die dynamische Seitengenerierung bearbeitet alle Inhalts-Anfragen. Sie erzeugt die gewünschte Seite aus den Metainformationen und dem Template sowie den Ergebnissen der Bearbeitung der aktiven Inhalte. Anschließend wird die fertige Seite an den anfragenden Browser ausgeliefert.

Zusätzlich wird auch die Session-ID mitgeliefert, um spätere Anfragen dieses Besuchers wieder eindeutig zuordnen zu können.

Damit die dynamische Generierung von Seiten funktioniert, muss der Hintergrund-Daemon auf dem Zielsystem laufen.

7.2.2 Session-Management

Das Session-Management ist für die Verwaltung der Besucherdaten, der Besucherprofile und der Session zuständig. Alle verfügbaren Daten eines Besuchers werden zu einem Profil zusammengefasst und in der Profil-Datenbank gespeichert.

Jeder Besucher wird über eine Session-ID identifiziert, die er wie folgt erhält:

- über eine neue Session für Besucher, die dem System nicht bekannt sind
- über ein Cookie aus einer früheren Session
- über ein in der Profil-Datenbank gespeichertes Profil (Anmeldung im System über ein dafür entwickeltes Formular)

Die Session-ID wird in der URL mitgeführt und dient bei allen Inhaltsanfragen während des Besuchs zur Identifikation.

Zu jeder Session-ID existiert ein Profil in der Profildatenbank, in dem alle während der Session gesammelten relevanten Daten des Besuchers gespeichert werden.

7.2.3 Profil-Datenbank

In der Profil-Datenbank werden alle Profile der Besucher gespeichert. Diese Profile können neben der Session-ID weitere beliebige Eigenschaften und Daten enthalten. Die Session-ID dient dabei als Schlüssel, um Besucherdaten wieder auffinden zu können.

7.2.4 Template-Interpreter

Der Template-Interpreter bearbeitet ein Template, in dem zum einen der statische Teil und zum anderen der dynamische Teil der angeforderten Seite definiert wird. Unter Berücksichtigung der vom Session-Management gelieferten Besucherdaten wird bestimmt, welcher Inhalt unter den gegebenen Bedingungen an welcher Stelle der fertigen Seite erscheint. Der entsprechende Inhalt wird aus der Content-Datenbank geholt.

Nachdem das Template geparst und die entsprechenden Inhalte eingefügt wurden, liefert der Template-Interpreter das Ergebnis an die dynamische Seitengenerierung.

7.2.5 Content-Datenbank

Die Content-Datenbank stellt alle Inhalte zur Verfügung. Das Einspielen dieser Inhalte erfolgt wie bei statischen Seiten durch Freischalten aus dem Produktionssystem.

7.3 Personalisierungsfunktionen

Grundsätzlich unterscheiden sich die Funktionen der Personalisierung nicht sehr von denen von SiteActive. Lediglich einige Erweiterungen bezüglich der verfügbaren Variablen wurden vorgenommen und mit den bereits vorhandenen SiteActive-Funktionen zu einem neuen Modul zusammengefasst.

Bei der Personalisierung werden im Gegensatz zu dem bisherigen Verfahren keine statischen HTML-Seiten auf dem Zielsystem gespeichert. Der Inhalt wird mit Hilfe des Skripts `site_master.pl` zur Laufzeit durch ein Template für den angemeldeten Besucher erzeugt.

Es ist möglich, eine einfache Personalisierung nur mit Hilfe des automatischen Session-Management sowie den erweiterten SiteActive-Funktionen und -Variablen zu realisieren. Es stehen dabei alle Möglichkeiten zur Verfügung, die in Abschnitt 7.5 **Verwalten von Userdaten** auf Seite 157 beschrieben sind.

Für die Realisierung umfangreicherer Personalisierungsprojekte kann in den verwendeten Templates Perl-Code ausgeführt werden. Dieser muss innerhalb folgender Tags erscheinen:

```
<IMPERIA lang="perl">
  Anweisungen
<IMPERIA>
```

Für solche IMPERIA-Codeblöcke stehen die im Folgenden beschriebenen Funktionen für die Personalisierung zur Verfügung.

7.3.1 Syntax-Referenz

Im Folgenden werden wir die Funktionen beschreiben, mit deren Hilfe eine personalisierte Website mit Imperia 8 erzeugt werden kann. Die einzelnen Code-Blöcke werden, wie oben bereits erwähnt, in Imperia-Blöcke (<IMPERIA></IMPERIA>) eingefasst. Zusätzlich wird über das `lang`-Attribut die Sprache PERL bestimmt (lang=perl).

In diesen Code-Blöcken stehen folgende Funktionen zur Verfügung.

clearlist()

Löscht die Trefferliste. Diese Funktion ist identisch mit dem Befehl CLEARLIST aus SiteActive.

filemask($mask)

Diese Funktion erhält als Parameter einen Regulären Ausdruck, um mit dem Perl-Matching-Operator den Dateinamen von jedem mit `ireaddir` (siehe unten) gefundenen Treffer zu vergleichen.

Ist identisch mit dem Befehl FILEMASK aus SiteActive.

ireaddir($Directory)

Diese Funktion erhält als Parameter das Verzeichnis ($Directory), das rekursiv nach Dateien durchsucht werden soll. Die Meta-Informationen dieser Dateien werden dabei eingelesen. Diese Funktion liefert die Trefferliste und ist identisch mit der Funktion READDIR aus SiteActive.

dynamic_enable($)

Bestimmt, ob eine Auswertung der Datei `dynamic.conf` durchgeführt oder unterdrückt wird. Mögliche Parameter sind `1`, die Auswertung der `dynamic.conf` wird durchgeführt, und `2`, die Auswertung der `dynamic.conf` wird unterdrückt.

Ist identisch mit den Befehlen DYNAMIC ENABLE und DYNAMIC DISABLE aus SiteActive.

reject($fullpath)

Verhindert, dass Dateien mit dem angegebenen Pfad in die Trefferliste aufgenommen werden. Es muss der gesamte Pfad ausgehend von Document-Root angegeben werden.

Ist identisch mit der Funktion REJECT aus SiteActive.

allow($fullpath)

Entfernt eine zuvor mit `reject($fullpath)` gesetzte Einschränkung. Es muss der gesamte Pfad ausgehend von Document-Root angegeben werden.

Ist identisch mit dem Befehl ALLOW aus SiteActive.

setdir($)

Setzt das Verzeichnis für die Variablen-Interpolation bei der dynamic.conf, falls kein READDIR durchgeführt wird.

Ist identisch mit der Funktion SETDIR aus SiteActive.

Sortierung der Trefferliste

Die folgenden Funktionen sortieren die Trefferliste entsprechend der Funktionen von SiteActive `sort_by_direlem`, `sort_by_age`, `sort_by_meta`, `sort_by_database`, `sort_oldest_first`, `sort_latest_first`, `sort_by_filename`, `sort_by_metafield`, `sort_by_multifield`.

get_rand_from_list($,$)

Wählt die übergebene Anzahl von zufälligen Elementen aus der übergebenen Liste aus. Entspricht dem Befehl RANDOM PICK aus SiteActive, kann jedoch beliebige Listen verarbeiten.

reverse_list($)

Dreht eine ggfs. bereits sortierte Liste um, so dass die letzten Elemente am Anfang der Liste stehen.

Ist identisch mit dem Befehl REVERSE LIST aus SiteActive.

read_hash($)

Liest die übergebene Datei, deren Endung `.hash` sein muss, aus dem Document-Root als Hash ein. Jede Zeile der Datei muss ein Schlüssel-Wert-Paar enthalten.

make_link($url,$title)

Erstellt für den aktuellen Besucher einen personalisierten Link mit `$url` als Ziel und `$title` als dargestelltem Text. Alternativ zu `$url` und `$title` kann auch der Meta-Hash des zu verlinkenden Eintrags übergeben werden. Beispiel: `make_link($FILE_META{"/movies/action/00912/index.html}");`

trim_blanks($)

Entfernt führende und abschließende Leerzeichen aus dem der Funktion übergebenen Text. Weiterhin werden mehrfache Leerzeichen, Tabulatoren, Newlines und Carriage-Returns in Leerzeichen umgewandelt.

deactivate_html($)

Ersetzt Zeichen wie <, >, &, " durch die entsprechenden HTML-Entities. Gibt den neuen Wert zurück. Wahlweise kann statt eines Strings auch eine Referenz auf einen String übergeben werden. Dann wird auch der übergebene Wert selbst verändert.

encode_url_parm($)

Erzeugt aus einem übergebenen String einen neuen String, in dem jeder Buchstabe aus dem Originalstring mit seinem ASCII-Code in hexadezimaler Schreibweise ersetzt wird. Gibt den neuen Wert zurück. Wahlweise kann statt eines Strings auch eine Referenz auf einen String übergeben werden. Dann wird auch der übergebene Wert selbst verändert.

decode_url_parm($)

Umkehrfunktion zu encode_url_parm($). Gibt den neuen Wert zurück. Wahlweise kann statt eines Strings auch eine Referenz auf einen String übergeben werden. Dann wird auch der übergebene Wert selbst verändert.

remove_mid_url($)

Entfernt die Session-ID aus der übergebenen URL. Gibt den neuen Wert zurück. Wahlweise kann statt eines Strings auch eine Referenz auf einen String übergeben werden. Dann wird auch der übergebene Wert selbst verändert.

insert_midurl($)

Fügt die Session-ID des aktuell angemeldeten Besuchers in die übergebene URL ein. Gibt den neuen Wert zurück. Wahlweise kann statt eines Strings auch eine Referenz auf einen String übergeben werden. Dann wird auch der übergebene Wert selbst verändert.

valid_id($)

Überprüft, ob die übergebene MID gültig ist.

7.3.2 Personalisierungsvariablen

Dieser Abschnitt listet alle Variablen auf, die in den Personalisierungs-Funktionen zur Verfügung stehen.

7.3.2.1 %FILE_META (Typ: Hash)

Beinhaltet alle Metainformationen zu allen in der Trefferliste, also dem Ergebnis von `ireaddir`, enthaltenen Dokumenten. Diese Variable entspricht den YY-Variablen innerhalb der FOREACH-FOUND-Schleifen von SiteActive.

Auf die Metainformationen einzelner Dokumente kann wie folgt zugegriffen werden:

```
foreach (keys %FILE_META){
   print make_link($FILE_META{$_});
   print "<br />\n";
};
```

7.3.2.2 $META (Typ: Referenz auf einen Hash)

Enthält die Metainformationen des aktuellen Dokuments. Entspricht dem Inhalt der TM-Variablen aus SiteActive.

7.3.2.3 $DATE (Typ:Referenz auf einen Hash)

Beinhaltet Datumsinformationen. Im Einzelnen sind dies:

Wert	Beschreibung
day	Tag im Monat
mon	Monat
month	Monatsname
fullmonth	identisch mit `month`
fullyear	vierstellige Jahresziffer
shortyear	zweistellige Jahresziffer
year	zweistellige Jahresziffer
time	Uhrzeit im Format hh:mm
longtime	Uhrzeit im Format hh:mm:ss
hour	Stunden zweistellig
min	Minuten zweistellig
minutes	Minuten zweistellig
sec	Sekunden zweistellig
second	Sekunden zweistellig
dayofweek	Wochentag ausgeschrieben
wday	Wochentagsziffer (0-6, Sonntag ist 0)

Tabelle 7.1. Datumsinformationen in $DATE

7.3.2.4 $FORM (Typ: Referenz auf einen Hash)

Im Allgemeinen ein leerer Hash. Beinhaltet die Formularparameter, die dem Skript `site_master` übergeben wurden. Ist identisch mit dem Inhalt der FF-Variablen aus SiteActive.

7.3.2.5 $XENV (Typ: Referenz Perl-Hash $ENV)

Beinhaltet alle Umgebungsvariablen. Beispiel

```
print $XENV->{DOCUMENT_ROOT}
```

7.3.2.6 $USER (Typ: Referenz auf einen Hash)

Beinhaltet die kompletten Informationen, die über den aktuell angemeldeten Besucher zur Verfügung stehen. Beispiel:

```
print $USER->{cellular}
```

7.3.2.7 @FILELIST

Enthält die Liste aller kumulierten, von `ireaddir` gefundenen Dokumente. Entspricht `keys(%FILE_META)`.

7.3.2.8 $FILE_META

Referenz auf %FILE_META.

7.3.2.9 $GLOBAL

Dies ist die einzige freie globale Variable, die während der gesamten Auswertung des Templates überall sichtbar bleibt. Sie dient dazu, zwischen verschiedenen Code-Blöcken innerhalb eines Templates Informationen auszutauschen. Sie ist normalerweise undefiniert; Sie sollten Sie daher vor Gebrauch mit einer Referenz auf einen leeren Hash vorbelegen. Lesen Sie hierzu auch Abschnitt 7.6.5 **Daten zwischen Code-Blöcken austauschen** auf Seite 162.

7.4 Komponenten der Personalisierung

Im Wesentlichen sind für die Entwicklung einer personalisierten Website folgende Komponenten von Interesse:

- /cgi-bin/site_muser.pl
- /site/modules/core/PersUtils.pm
- /site/modules/core/LiveUsersDB.pm

Diese Module bauen in der oben gezeigten Reihenfolge aufeinander auf.

Das Skript `site_muser.pl` stellt das Interface dar, mit dessen Hilfe die Funktionen der Personalisierung in Formularen, Buttons und Hyperlinks aufgerufen werden können.

Das Modul `PersUtils.pm` enthält eine Reihe von Hilfsfunktionen, die von site_muser.pl aufgerufen werden. Diese Hilfsfunktionen können auch innerhalb von `<IMPERIA></IMPERIA>`-Blöcken verwendet werden.

Das Modul `LiveUsersDB.pm` implementiert die Funktionen zum Anlegen, Laden, Speichern und Ändern von Besucherinformationen und wird von `site_muser.pl` und `PersUtils.pm` aufgerufen.

Zusätzlich wird es vom Perl-Plug-In benutzt, von dem die `<IMPERIA lang=perl>`-Blöcke bearbeitet werden, um die `$USER`-Variable mit den Besucherdaten des aktuell angemeldeten Besuchers zur Verfügung zu stellen. Diese Besucherdaten werden auch vom Modul `PageParser.pm` geladen, um die UU-Variablen ersetzen zu können.

7.5 Verwalten von Userdaten

Dieser Abschnitt befasst sich mit den Funktionen zur Verwaltung der Besucherdaten sowie zur Übergabe der Session- und User-ID.

Die Verwaltung von Besucherdaten ist so angelegt, dass beliebige Perl-Strukturen (auch verschachtelte Datenstrukturen) abgelegt werden können. Dies vereinfacht die Verwaltung von Listen, wie zum Beispiel einer Bookmark-Liste, und ist die Voraussetzung für die Speicherung der Werte von Gruppen gleichnamiger Checkboxen, ohne dass eine aufwendige Umkodierung notwendig wird. Solche Checkboxen werden in vielen Fällen verwendet, wenn beispielsweise Besucherpräferenzen abgefragt werden sollen.

Zur Verwaltung der Besucherdaten wird das Skript `/cgi-bin/site_muser.pl` verwendet. Über den Parameter `_cmd` werden verschiedene Kommandos übergeben. Beispiel:

```
/cgi-bin/site_muser.pl?_cmd=save&_MID=Session-ID&bookmarks=Wert
```

Innerhalb eines Aufrufs müssen alle Werte URL-encoded sein. Es findet zurzeit keine Überprüfung der Attributnamen statt, so dass die Gefahr besteht, Felder (z.B.: `login` und `password`) über manuell in die Adressleiste des Browsers eingegebene Werte zu zerstören.

Der Parameter `_cmd` kann folgende Kommandos enthalten:

new

Legt ein neues Besucherprofil an und erwartet folgende Felder bzw. kann diese verarbeiten:

Feld	Beschreibung
login	(**Pflichtfeld**) Enthält das gewählte Log-In-Kürzel
-password	(**Pflichtfeld**) Enthält das gewählte Passwort
-password2	Enthält die Bestätigung des gewählten Passworts.
_REQUIRED	Enthält die Liste der Pflichtfelder (z.B.: login,password,password2) ohne führende Minuszeichen.
email	Enthält die Email-Adresse, an die die Zugangsdaten gesendet werden sollen.
-emailpassword	Enthält 1 oder 0. Ist der Wert 1, wird das Passwort mit versendet.
	weitere, frei definierbare Felder, die weitere Besucheratributе (Vorname, Nachname etc.) enthalten.

Tabelle 7.2. Felder des Kommandos `new`

save

Schreibt Änderungen an einem bestehenden Profil sowie die übergebenen Formulardaten in die Benutzer-Datenbank. Es übergibt die gleichen Felder wie das Kommando `new`.

login

Meldet den Besucher an. Dieses Kommando erwartet folgende Felder:

Feld	Beschreibung
login	(**Pflichtfeld**) Enthält das gewählte Log-In-Kürzel
-password	(**Pflichtfeld**) Enthält das gewählte Passwort

Tabelle 7.3. Felder des Kommandos `login`

logout

Meldet einen angemeldeten Besucher ab. Dieses Kommando übergibt keine Felder.

add_field

Fügt weitere Informationen in ein bestehendes Feld ein. Die neuen Werte werden mit Semikolons von vorhandenen Werten getrennt und an den Anfang des Feldes gesetzt.

Dieses Kommando kann beliebige, frei definierbare Felder übergeben. Ausgeschlossen sind jedoch die Felder `login`, `-password`, `-password2`, `_REQUIRED`, `email` und `-emailpassword`. Beispiel:

```
/cgi-bin/site_muser.pl?_cmd=add_field&_MID=...&bookmarks=Liste
```

Dieser Aufruf fügt dem Besucherprofil eine Liste mit dem Namen `bookmarks` hinzu bzw. fügt einer bereits existierenden Liste die übergebenen Werte an den Anfang hinzu.

Existiert im fraglichen Besucherprofil noch keine Liste mit dem entsprechenden Namen, wird eine solche mit den übergebenen Werten angelegt.

In einem Aufruf können mehrere Listen mit einem oder mehreren Werten ergänzt oder angelegt werden:

```
/cgi-bin/site_muser.pl?_cmd=add_field&_MID=...&bookmarks=Liste
                                              &categories=Wert1
                                              &categories=Wert2
```

Hinweis:

Der Code wurde aus Platzgründen umgebrochen

Dieser Aufruf fügt an den Anfang der Liste `bookmarks` die Liste *Liste* ein und erzeugt bzw. ergänzt die Liste mit dem Namen `categories` mit den Werten *Wert1* und *Wert2*.

remove_field

Löscht den übergebenen Inhalt aus einem mit `add_field` erzeugten Feld, falls dieser bereits vorhanden ist. Es können beliebige, frei definierbare Felder übergeben werden. Ausgeschlossen sind jedoch die Felder `login`, `-password`, `-password2`, `_REQUIRED`, `email` und `-emailpassword`. Beispiel:

```
/cgi-bin/site_muser.pl?_cmd=remove_field&_MID=...&bookmarks=Liste
                                                &categories=Wert1
                                                &categories=Wert2
```

Mit diesem Aufruf werden die angegebenen Werte aus den entsprechenden Listen gelöscht. Sind die Werte in den Listen nicht vorhanden, wird von der Funktion keine Aktion durchgeführt.

7.6 Besucherdaten

Das System stellt zur Identifikation von Besuchern zwei verschiedene Möglichkeiten zur Verfügung:

- mit Hilfe eines bei einem vorherigen Besuch gesetzten Cookies
- Identifikation über die URL

Namen von Formularfeldern dürfen nicht mit einem Unterstrich oder einem Minuszeichen beginnen. Der führende Unterstrich im Namen eines Formularfeldes ist reserviert für interne Funktionen der Personalisierung, das führende Minuszeichen im Namen eines Formularfeldes ist für spezielle Sonderfunktionen reserviert.

7.6.1 Anmeldung eines Besuchers

Für das Anmelden eines Besuchers müssen im Anmeldeformular Felder mit folgenden vordefinierten Namen vorhanden sein:

- login
- -password

Im Feld mit dem Namen `login` wählt der Besucher das Kürzel, mit dem er sich später wieder im System identifiziert. Falls das eingegebene Kürzel bereits vergeben sein sollte, wird ein entsprechender Hinweis angezeigt. Das Log-In-Kürzel darf maximal 16 Zeichen lang sein und darf keine HTML-Sonderzeichen (z.B.: <, >, ", &) enthalten. Es empfiehlt sich, dies schon im Browser durch die Verwendung des Attributs `max-length="16"` einzuschränken.

Das Formularfeld mit dem Namen `-password` muss vom Typ `password` sein. In dieses Feld gibt der Besucher sein Passwort ein, das ihm die Anmeldung an das System erlaubt. Es kann zusätzlich auch ein weiteres Feld vom gleichen Typ mit dem Namen `-password2` zur Bestätigung der Passworteingabe existieren.

Eine Überprüfung, ob `-password` und `-password2` übereinstimmen, findet jedoch nur dann statt, wenn in einem Formularfeld des Typs `hidden` mit dem Namen `_REQUIRED` beide Feldnamen erscheinen. Der Wert dieses Feldes besteht aus den beiden, durch Kommata (ohne Leerzeichen!) voneinander getrennten Namen der Pflichtfelder, jedoch ggf. ohne führende Minuszeichen.

Die Felder `login` und `password` sind für die Anmeldung eines Besuchers immer Pflicht, auch dann, wenn sie nicht ausdrücklich im Feld `_REQUIRED` erscheinen.

Beim Abspeichern geänderter Besucherdaten sind diese Felder jedoch niemals Pflicht, selbst wenn sie im Feld `_REQUIRED` aufgeführt werden. Dies hat den Vorteil, dass bei den Parametern `new` und `save` des Kommandos `_cmd` das Feld `_REQUIRED` identisch belegt werden kann.

Damit sich ein neuer Besucher im System anmelden kann, muss das Skript `/cgi-bin/site_muser.pl` aufgerufen werden. Dieser Aufruf muss mindestens folgende Parameter per CGI übergeben:

_cmd="new"

Der Parameter `new` im Kommando `_cmd` veranlasst das Skript `site_muser.pl`, ein neues Profil anzulegen.

_LOCATION="*URL*"

Hiermit wird die URL übergeben, an die der Besucher weitergeleitet wird, wenn die Anmeldung erfolgreich verlief.

Falls `_LOCATION` nicht übergeben wird, gilt der Default-Wert `/home/index.htmx`.

_FAILURE="*URL*"

Hiermit wird die URL übergeben, an die der Besucher weitergeleitet wird, wenn die Anmeldung fehlgeschlagen ist.

Falls `_FAILURE` nicht übergeben wird, gilt der Default-Wert von `_LOCATION`.

_REQUIRED="*Feld 1*,*Feld 2*,*Feld 3*"

In dieser Variablen werden die Pflichtfelder definiert, die der Besucher ausfüllen muss, um sich anmelden zu können. Dies können zum Beispiel die Felder `fname` und `name` sein. Dieser Parameter wird nur von den Kommandos `new` und `save` ausgewertet.

_MID=

Dies ist die Session-ID des aktuell angemeldeten Besuchers, soweit vorhanden.

login=

Hier wird der vom Besucher eingegebene Log-In-Name übergeben. Dieser Wert wird im Feld `login` gespeichert.

password=

Hier wird das vom Besucher eingegebene Passwort übergeben. Der hier angegebene Wert wird im Feld `password` verschlüsselt gespeichert. Weitere Felder können nach Belieben übergeben und gespeichert werden.

Bei erfolgreicher Anmeldung werden diese Felder als Attribute des Besuchers abgespeichert. Weitere interessante Informationen über einen Besucher sind zum Beispiel der Name, das Geschlecht, der Wohnort und die Adresse.

Der einfachste Weg, um Besucher im System anzumelden, ist der Weg über ein Formular. Innerhalb des `FORM`-Tags werden die einzelnen Felder gefüllt und mit Hilfe eines Buttons wird die Anmeldung initiiert:

```
<form action="/cgi-bin/site_muser.pl" method="post">
  <input type="hidden" name="_cmd" value="new">
  <input type="text" name="login" size="30" maxlength="16">
  <input type="text" name="password" maxlength="16">
  <input type="text" name="name">
  <input type="text" name="fname">
  <input type="submit" value="Anmelden">
</form>
```

7.6.2 Zugangsdaten per Email verschicken

Eine Sonderfunktion bei der Anmeldung eines neuen Besuchers ist das Verschicken einer Email mit den neuen Zugangsdaten. Hierzu wurden zwei weitere Felder vordefiniert:

- email
- -emailpassword

Das Feld `email` dient zur Aufnahme der Email-Adresse des neuen Besuchers, das Feld `-emailpassword` zum Einschalten dieser Sonderfunktion. Das Feld `-emailpassword` konzipiert man dabei am besten als Feld des Typs `hidden` oder als Checkbox (Wert 0 oder 1).

Zur Verwendung dieser Sonderfunktion müssen in der Datei `/site/config/system.conf` verschiedene Werte definiert werden:

```
"SMTP-CLIENT" = "www.domain.com"
"SMTP-SERVER" = "smtp.domain.com"
"WEBMASTER" = "webmaster@mail.domain.com"
```

SMTP-CLIENT

Dies ist der Name des Webservers, auf dem die personalisierte Website läuft. Dies kann auch der physikalische Name des Rechners sein, auf dem mehrere virtuelle Webserver laufen. Wichtig ist, dass der SMTP-Server diesen Namen als Absender akzeptiert.

SMTP-SERVER

Dies ist der Name eines Rechners, der SMTP-Verbindungen akzeptiert und Email an den Empfänger verschickt bzw. weiterleitet.

WEBMASTER

Hier sollte eine gültige Email-Adresse eingetragen werden, an die sich Besucher bei technischen Problemen wenden können.

Wahlweise kann zusätzlich der Wert `SMTP-TIMEOUT` gesetzt werden, über den die Anzahl Sekunden bestimmt wird, die maximal bis zur Etablierung der Verbindung mit dem SMTP-Server gewartet werden soll. Der Default-Wert beträgt 45 Sekunden.

7.6.3 Besucherdaten ändern und ergänzen

Die gespeicherten Daten eines Besuchers können durch Aufruf des Skriptes `site_muser.pl` geändert bzw. ergänzt werden. Dabei können vorhandene Daten überschrieben oder ergänzt werden.

Mit dieser Funktion kann zum Beispiel der Klickpfad oder die Vorlieben des Besuchers gespeichert werden. Diese Daten stehen bei einem erneuten Anmelden des Besuchers wieder zur Verfügung und können entsprechend ausgewertet werden.

Folgende Parameter müssen übergeben werden:

- die Variable `_MID` mit der Besucheridentifikation
- das abzuspeichernde Feld mit dem Inhalt
- die Variable `_cmd` mit dem entsprechenden Wert
- die Variable `_LOCATION` mit der URL, zu der der Besucher nach erfolgreicher Änderung gelangt.
- die Variable `_FAILURE` mit der URL, zu der bei einem Fehler weitergeleitet wird.

Zur Übergabe der Variablen können zum Beispiel versteckte INPUT-Felder verwendet werden:

```
<form action="/cgi-bin/site_muser.pl" method="post">
<input type="hidden" name="_cmd" value="save">
<input type="hidden" name="_MID" value="<!--UU-MID-->">
<input type="hidden" name="_LOCATION" value="/home.htmx">
<input type="hidden" name="_cmd" value="save">
```

```
Handynummer eingeben<br />
<input type="text" name="cellular" value="<!--UU-cellular-->">
<input type="submit" value="Handynummer speichern">
</form>
```

7.6.4 Besucherdaten im Template verwenden

Innerhalb eines Templates kann auf alle gespeicherten Daten eines Besuchers zugegriffen werden. Hierzu werden UU-Variablen verwendet. Die Syntax für die Verwendung einer UU-Variable ist wie folgt:

```
<!--UU-Feld-->
```

Anstelle des Platzhalters *Feld* werden die Felder und Attribute angegeben, auf die zugegriffen werden soll. Hierbei stehen alle bereits im Profil des Besuchers gespeicherten Felder zur Verfügung. Beispiel:

```
<!--UU-fname-->
```

Diese Variable enthält den Wert des Feldes `fname` aus dem Profil des aktuell angemeldeten Besuchers. Dies funktioniert auch dann, wenn das Feld ein Array (eine Liste) oder einen Hash enthält.

Folgende Attribute sind bereits definiert:

UID

Enthält die vom System vergebene eindeutige User-ID des aktuell angemeldeten Besuchers.

MID

Enthält die Session-ID des aktuell angemeldeten Besuchers. In der vorliegenden Version ist sie identisch mit der User-ID.

login

Enthält die Log-In-Kennung des aktuell angemeldeten Besuchers.

7.6.5 Daten zwischen Code-Blöcken austauschen

Der Datenaustausch zwischen mehreren Code-Blöcken wird über eine globale Variable, den Hash `%$GLOBAL`, ermöglicht.

In diesem Hash könnte man beispielsweise speichern, welche Artikel in der aktuellen Seite bereits verlinkt sind, so dass mehrere Links zu ein und demselben Artikel vermieden werden können.

Oder Sie können darin berechnete Werte ablegen und diese mit Hilfe der SiteActive-Variablen der Form <!--GG-*Attributname*--> in ein Dokument einfügen. Dies funktioniert auch dann, wenn das Feld *Attributname* ein Array (eine Liste) oder einen Hash enthält.

7.6.6 Rückgabewerte von `site_muser.pl`

Das Skript `site_muser.pl` stellt das Interface zwischen Besucher und Personalisierung dar. Es liefert gegebenenfalls Rückgabewerte in Form von Formulardaten in der Parameterzeile, falls bei der Durchführung einer Funktion ein Fehler auftritt.

Folgende Rückgabewerte sind möglich:

Wert	Beschreibung
_ERROR	Fehlermeldung
_MISSING	Liste fehlender Parameter
_TAKEN	Log-In bereits vergeben
_UNKNOWN	unbekannter Besucher
_FAILED	Falsches Passwort

Tabelle 7.4. Mögliche Rückgabewerte

7.7 Beispiel einer personalisierten Website

Das folgende Beispiel zeigt grundlegende Techniken zur Personalisierung einer Website. Ein Besucher dieser Seite kann sich an das System anmelden und durch Angabe von Schlüsselwörtern den angezeigten Content bestimmen.

Das Beispiel enthält zwei Seiten:

- eine Log-In-Seite zum An- und Abmelden
- die personalisierte Seite

Auf der Log-In-Seite legt der User ein Profil durch Angabe seines Namens, eines Passworts und verschiedener Präferenzen an.

Die personalisierte Seite wird automatisch erzeugt und enthält nur den Content, der auf die vom User eingegebenen Präferenzen passt. Das Template für diese Seite enthält den Code, der für die Personalisierung verantwortlich ist.

Die Beschreibung des Beispiels beschränkt sich nur auf die relevanten Codeabschnitte und berücksichtigt keine Layout-Elemente.

Das Beispiel geht davon aus, dass eine Rubrik mit dem Namen `PersTest` angelegt wird. Des Weiteren wird vorausgesetzt, dass die Log-In-Seite `myLoginLogout.html` und die zu personalisierende Seite `myImperia. html` heißt. Wenn Sie andere Namen verwenden wollen, ersetzen Sie die hier genannten an den entsprechenden Stellen im Code.

Der Webserver für ein personalisiertes Zielsystem sollte aus Performance-Gründen für den Betrieb mit dem Modul **mod_perl** konfiguriert sein. Informationen zur Konfiguration für die Verwendung von **mod_perl** finden Sie in der Dokumentation Ihres Webservers. Für den Apache-Webserver finden Sie weitere Informationen unter `http://perl.apache.org/`. Für das Beispiel ist eine Webserver-Konfiguration ohne **mod_perl** ausreichend.

7.7.1 Template und Metadatei

Um Dokumente zu erzeugen, die in der personalisierten Site angezeigt werden können, müssen ein Template und eine Metadatei erzeugt werden.

In der Metadatei werden Dateiname und Verzeichnis der fertigen Dokumente sowie das Template bestimmt.

Das Template enthält Eingabefelder für den Content, der später personalisiert dargestellt werden soll. In unserem Beispiel verwenden wir die Felder `title`, `leadtext` und `contentblock`.

7.7.2 Seite zum An- und Abmelden

Diese Seite enthält einige versteckte Felder, mit deren Hilfe die folgenden Parameter übergeben werden:

- _MID
- _LOCATION
- _FAILURE

Das Feld _MID enthält die Session-ID des Users, sofern er schon im System bekannt ist. Diese Session-ID wird beim ersten Anmelden generiert. Mit _LOCATION wird die URL übergeben, an die der Besucher weitergeleitet wird, wenn die Anmeldung erfolgreich verlief. _FAILURE enthält die URL, an die der Besucher weitergeleitet wird, wenn die Anmeldung fehlgeschlagen ist. Sie können zum Beispiel folgenden Code hierfür verwenden:

```
<input type="hidden" name="_MID" value="<!--UU-MID-->">
<input type="hidden" name="_LOCATION" value="/PersTest/myImperia.htms">
<input type="hidden" name="_FAILURE" value="/PersTest/myImperia.htms">
```

Weiterhin sollen sich Besucher auf dieser Seite an das System an- oder abmelden können. Es muss festgestellt werden, ob der Besucher bereits angemeldet ist. Dazu wird innerhalb einer IF-Abfrage geprüft, ob die Variable <!--UU-UID--> leer ist oder nicht. Es wird der auszugebende HTML-Code für beide Zustände festgelegt:

```
#IF ("<!--UU-UID-->")
  Name:<!--UU-login--><br />
  Daten:<!--UU-data--><br />
  <input type="submit" value="Abmelden">
  <input type="hidden" name="_cmd" value="logout">
#ELSE
  <input type="text" name="login">
  <input type="password" name="-password">
  <input type="text" name="data" size=50>
  <input type="submit" value="Anmelden">
  <input type="hidden" name="_cmd" value="new">
#ENDIF
```

Ist die Variable <!--UU-UID--> nicht leer, werden der Name des Besuchers und seine gespeicherten Daten angezeigt. Weiterhin erscheint ein Button zum Abmelden. Das HIDDEN-Feld übergibt das entsprechende Kommando logout.

Ist die Variable <!--UU-UID--> leer, werden folgende Eingabefelder angezeigt, die der Besucher ausfüllen muss:

- Name (Feld login)
- Passwort (Feld -password)
- Besucherdaten (Feld data)

Anhand des in das Feld data eingegebenen Textes wird später der vorhandene Content personalisiert. Weiterhin wird ein Button zum Übertragen des Formulars angezeigt und das Kommando new wird über das HIDDEN-Feld übergeben.

7.7.3 Personalisierte Seite

Diese Seite enthält alle aktiven Inhalte, die vor der Auslieferung der Seite an den Besucher ausgeführt werden sollen.

Auch hier verwenden wir die Variable <!--UU-UID--> um zu prüfen, ob der Besucher angemeldet ist. Ist er angemeldet, wird der vorhandene Content anhand der Daten des Besuchers, die im Feld data gespeichert sind, aufbereitet und angezeigt. Ist der Besucher nicht angemeldet, wird der gesamte vorhandene Content angezeigt.

7.7.3.1 Besucher ist angemeldet

Ist ein Besucher bereits angemeldet, wird folgender Code ausgeführt:

```
#IF ("<!--UU-UID-->")
<IMPERIA lang=perl>
  filemask("index.html");
  ireaddir("/PersTest/");
  sort_latest_first();
  foreach my $file (@FILELIST) {
```

```
    my $rel_file = $FILE_META{$file}->{directory}.'/
                              '.$FILE_META{$file}->{filename};
    my $title    = $FILE_META{$file}->{title};
    my $leadtext = $FILE_META{$file}->{leadtext};
    my $text     = $FILE_META{$file}->{contentblock};
    my $found    = 0;

    foreach my $i (split('\s',$USER->{data})) {
      $found++ if ($title   =~s/($i)/<B>$1<\/B>/ig);
      $found++ if ($leadtext=~s/($i)/<B>$1<\/B>/ig);
      $found++ if ($text    =~s/($i)/<B>$1<\/B>/ig);
    }

    if ($found > 0) {
      print "$title \n";
      print "$leadtext \n";
      print "$text \n";
      print "<a href=$rel_file>$rel_file</a> \n";
    }
  }
</IMPERIA>
```

Achtung:

Teile des Codes wurden aus Platzgründen umgebrochen.

Im ersten Abschnitt des Codes wird zunächst bestimmt, dass innerhalb des IMPERIA-Blocks Perl verwendet wird. Danach bestimmen Sie über die Filemask, welche Verzeichnisse und Dokumente gelesen werden sollen und sortieren daraufhin die gefundenen Treffer nach ihrem Datum, so dass die jüngsten Treffer am Anfang der Liste stehen. Alle gefundenen Dokumente werden in der Liste `@FILELIST` gespeichert.

Im zweiten Code-Abschnitt werden verschiedene Variablen mit Werten gefüllt:

`$rel_file` enthält den Pfad. Für jedes Dokument in der Liste `@FILELIST` wird aus dem Hash `$FILE_META` das Verzeichnis (directory) und der Dateiname (filename) extrahiert. Beide Werte werden dann in der Variablen `$rel_file` gespeichert, wobei sie durch einen Slash getrennt werden.

`$title` enthält den Titel eines Dokuments. Für jedes Dokument in der Liste `@FILELIST` wird aus dem Hash `$FILE_META` der Titel (title) extrahiert und in der Variablen `$title` gespeichert.

Für die Variablen `$leadtext` und `$text` wird entsprechend verfahren.

Mit Hilfe des dritten Code-Abschnitts werden alle Wörter in den Treffern, die einem Wort aus dem Feld `data` entsprechen, fett formatiert:

Zunächst werden alle im Feld `data` gespeicherten Wörter anhand der Leerzeichen getrennt und in der Variablen `$i` gespeichert. Dann wird jeweils in den Variablen `$title`, `$leadtext` und `$text` global und ohne Beachtung der Groß- und Kleinschreibung nach Entsprechungen gesucht. Gefundene Wörter werden in der ausgegebenen HTML-Seite fett dargestellt.

Der vierte Code-Abschnitt enthält die eigentliche Ausgabe:

Hier wird der Inhalt der einzelnen Variablen ausgegeben. Zusätzlich wird um die Variable `$rel_file` ein Anker gesetzt, der auf das entsprechende Verzeichnis verweist.

7.7.3.2 Besucher ist nicht angemeldet

Ist ein Besucher noch nicht angemeldet, wird folgender Code ausgeführt:

```
<IMPERIA lang=perl>
  filemask("index.html");
  ireaddir("/PersTest/");
  sort_latest_first();
  foreach my $file (@FILELIST) {
    my $rel_file = $FILE_META{$file}->{directory}.'/'.
                              $FILE_META{$file}->{filename};
    my $title    = $FILE_META{$file}->{title};
```

```
    my $leadtext = $FILE_META{$file}->{leadtext};
    print "$leadtext \n";
    print "<a href=$rel_file>$rel_file</a> \n";
    print "<hr></hr>\n";
  }
</IMPERIA>
```

Achtung:

Teile des Codes wurden aus Platzgründen umgebrochen.

Im ersten Abschnitt des Codes wird zunächst bestimmt, dass innerhalb des IMPERIA-Blocks Perl verwendet wird. Danach bestimmen Sie über die Filemask, welches Verzeichnis und welche Dokumente gelesen werden sollen und sortieren daraufhin die gefundenen Treffer nach ihrem Datum, so dass die jüngsten Treffer am Anfang der Liste stehen. Alle gefundenen Dokumente werden in der Liste `@FILELIST` gespeichert.

Im zweiten Code-Abschnitt werden verschiedene Variablen mit Werten gefüllt:

`$rel_file` enthält den Pfad. Für jedes Dokument in der Liste `@FILELIST` wird aus dem Hash `$FILE_META` das Verzeichnis (directory) und der Dateiname (filename) extrahiert. Beide Werte werden dann in der Variablen `$rel_file` gespeichert, wobei diese durch einen Slash getrennt werden.

`$title` enthält den Titel eines Dokuments. Für jedes Dokument in der Liste `@FILELIST` wird aus dem Hash `$FILE_META` der Titel (`title`) extrahiert und in der Variablen `$title` gespeichert.

Für die Variablen `$leadtext` und `$text` wird entsprechend verfahren.

Der dritte Code-Abschnitt enthält die eigentliche Ausgabe:

Es wird der Inhalt der einzelnen Variablen ausgegeben. Zusätzlich wird um die Variable `$rel_file` ein Anker gesetzt, der auf das entsprechende Verzeichnis verweist.

Kapitel 8. IWE - Imperia-WYSIWYG-Editor

8.1 Aufruf

8.1.1 Aufruf im Template

Um den IWE zu nutzen, muss im Imperia-Template eine entsprechende Processing Instruction platziert werden. Diese sieht in der Minimalkonfiguration wie folgt aus:

```
<?imperia iwe
id: iwe1 ?>
```

Diese Processing Instruction beinhaltet die Konfigurationsoptionen, welche in jedem Aufruf des IWE im Imperia-Template mindestens enthalten sein müssen. Es wird eine Instanz des IWE mit der Standardkonfiguration aufgerufen. Weitere Konfigurationsmöglichkeiten entnehmen Sie bitte dem ensprechenden Kapitel in der Dokumentation.

Wichtig:

Bitte beachten Sie, dass der Wert der 'id' Option eindeutig sein muss, d.h. nicht mehrfach verwendet werden darf.

8.1.2 Aufruf im Flexmodul

Um den IWE in einem Flexmodul aufzurufen, muss man beachten, dass die 'id' des Editors in jeder Instanz eindeutig ist. Dies erreicht man, indem man die ID des Flexmoduls mit in die Processing Instruction schreibt, welche den IWE aufruft.

Beispiel:

```
#IF ("<!--XX-editmode-->") <br />
<?imperia iwe
id: iwe1_<!--FLEX_INDEX-->_<!--FLEX_ID-->?>
#ELSE
<div> <!--XX-FLEX-iwe1--> </div>
#ENDIF
```

8.2 Konfiguration des IWE

Der IWE bietet eine Vielzahl von Konfigurationsoptionen, mit welchen die Funktionalitäten, das Aussehen und das Verhalten beeinflusst werden.

Es gibt es zwei Möglichkeiten, den Editor zu konfigurieren:

- Konfiguration im Template

  Bei dieser Methode erfolgt die Konfiguration der jeweiligen Instanz des Editors über das Imperia-Template.

- Steuerung über eine Konfigurationsdatei

  Hierbei wird entweder für jede Instanz eine eigene Konfigurationsdatei im Imperia-Template spezifiziert, oder es wird eine globale Datei für alle Instanzen verwendet.

Wichtig:

Bitte beachten Sie die Hierarchie der Konfigurationsmöglichkeiten. Wenn Sie eine Konfigurationsoption im Template gesetzt haben, hat diese grundsätzlich Vorrang vor der gesetzten Option in der Konfigurationsdatei. Es gibt also insgesamt drei Hierachiestufen:

1. *Template*
2. *Benutzerdefinierte Konfigurationsdatei der Instanz*
3. *Globale Konfigurationsdatei*

8.2.1 Konfiguration im Imperia-Template

Neben den erforderlichen Konfigurationsoptionen gibt es noch eine Vielzahl weiterer Möglichkeiten, den IWE entsprechend anzupassen. Ebenso gibt es einige Optionen, welche nur im Template gesetzt werden können und nicht in einer Konfigurationsdatei. Eine Liste mit allen verfügbaren Konfigurationsoptionen finden Sie im Internet unter http://docs.cksource.com/FCKeditor_2.x/Developers_Guide/Configuration/Configuration_Options

Die Konfiguration im Template erfolgt immer in der folgenden Syntax, wobei 'Option' der Name der Konfigurationsoption ist und 'wert' der zugehörige Wert:

```
<?imperia iwe
id: iwe1
Option: wert
SourcePopup: true?>
```

8.2.2 Steuerung über eine Konfigurationsdatei

Der Editor bietet auch die Möglichkeit, eine oder mehrere Instanzen über eine Konfigurationsdatei zu steuern. Hierfür muss die Option 'CustomConfigurationsPath' im Template gesetzt werden. Als zugehöriger Wert ist eine Konfigurationsdatei mit absolutem Pfad, relativ zum DOCUMENT-ROOT anzugeben. Beispiel:

```
<?imperia iwe
id: iwe1
CustomConfigurationsPath:/foo.js ?>
```

Anschliessend besteht die Möglichkeit, die meisten gewünschten Konfigurationsoptionen in der Datei 'foo.js' abzulegen.

Wichtig:

Ein wichtiger Unterschied zur Konfiguration über das Template ist, dass das Prefix 'FCKConfig.' vor die verwendete Option angefügt werden muss und dass das Option-Wert-Paar nicht mit einem Doppelpunkt, sondern mit einem Gleichheitszeichen getrennt wird. Ebenfalls zu beachten ist das abschliessende Semikolon.

Anhand des vorher bereits verwendeten Beispiels 'SourcePopup' würde die Konfigurationsdatei wie folgt aussehen:

```
FCKConfig.SourcePopup = true;
```

Detailiertere Informationen zur Konfiguration finden Sie im Kapitel „IWE - Imperia-WYSIWYG-Editor“ im Administrationshandbuch.

Kapitel 9. Anhang

In diesem Kapitel finden Sie ergänzende Informationen zu verschiedenen Funktionen und Modulen sowie Beschreibungen von Themen, die nicht unbedingt zu Imperia gehören, zum Verständnis oder zum Betrieb von Imperia aber nützlich sind.

9.1 Imperia Plug-Ins für den Daemon Hermes

Unter einem Hermes-Plug-In ist ein Modul zu verstehen, das eine oder mehrere Funktionen beinhaltet, die der Hintergrund-Daemon nach definierten Regeln ausführt. Ein maßgeblicher Vorteil von Plug-Ins ist ihr leichtes Einbinden und Entfernen. Sie können selbst während der Daemon läuft Plug-Ins einbinden oder entfernen.

Imperia liefert standardmäßig zahlreiche Plug-Ins mit, die unter anderem folgende Funktionen übernehmen:

- automatisches Freischalten
- automatisches Löschen
- Generierung von Übersichtsseiten
- Import von Dokumenten

Dank der Modularität können Sie die Funktionen des Daemons durch Hinzufügen weiterer Plug-Ins leicht erweitern. Den Daemon selbst brauchen Sie dabei nicht zu modifizieren. Dies gilt insbesondere für jede Art von automatischem Content-Import.

Neben der reinen Funktionalität muss ein Plug-In eine Schnittstelle enthalten, mit deren Hilfe es mit dem Daemon kommunizieren kann. Zum einen muss das Plug-In sich beim Einbinden identifizieren und eine Kompatibilitätsprüfung durchführen, zum anderen muss es mitteilen, unter welchen Bedingungen der Daemon welche Funktionen ausführen soll.

Letztere Informationen legt der Daemon als Default-Werte in der Ausführungstabelle ab. Des Weiteren gibt der Daemon Plug-Ins die Möglichkeit, Informationen in die Log-Datei zu schreiben oder Daten an andere Plug-Ins weiterzugeben.

Plug-Ins implementieren Sie als Perl-Packages. Die Funktionalität eines Plug-Ins bilden Sie durch dessen Methoden ab.

9.1.1 Plug-In-Rumpf-Code

Damit ein Plug-In erfolgreich eingebunden werden kann, muss es zumindest die Schnittstelle zum Daemon realisieren. Diese Schnittstelle muss aus einem Konstruktor und einer Init-Methode bestehen.

Der Konstruktor erhält vom Daemon alle notwendigen Informationen und führt die Kompatibilitätsprüfung durch. Die Init-Methode liefert die Default-Konfiguration an den Daemon.

9.1.1.1 Konstruktor

Der typische Konstruktor eines Plug-Ins sieht wie folgt aus:

```
sub new {
    my ($class, %params) = @_;
    my $self = {};
    while (my ($key, $value) = each %params) {
            $self->{'__' . $key} = $value;
    }
    #The named arguments also fit for a Hermes::Logger.
    $self->{__logger} = Hermes::Logger->new(%params, prefix  =>
    'myHermesPlugin');
    #Plugin is not yet operational. This will be reset to 1
    #by reinit().
    $self->{__bad} = 0;
```

```
    bless ($self, $class);
}
```

9.1.1.2 Init-Methode

Die Init-Methode liefert die Default-Werte für die Ausführung der einzelnen Plug-In-Methoden. Eine Init-Methode mit zwei Default-Einträgen sieht wie folgt aus:

```
sub init {
   [    { 'FUNCTION' =>'method_one',
          'MAX_RUNTIME' => 10,
          'EXEC_TIME' => 0,
          'PRIORITY' => 10
        },
        { 'FUNCTION' =>'method_two',
          'MAX_RUNTIME' => 3600,
          'EXEC_TIME' => 'no',
          'PRIORITY' => 90
        },
   ]
}
```

Die aufgeführten Felder haben die in Kapitel **Plug-In-Tabelle** im Administrationshandbuch beschriebene Funktionsweise und werden vom Daemon beim initialen Start in die Ausführungstabelle übertragen.

9.1.1.3 Methoden

Der Daemon übergibt beim Aufruf einer Methode verschiedene Parameter. Eine typische Methodendefinition sieht wie folgt aus:

```
sub method_one {
  my ($self, $container) = @_;
  my $logger = $self->{__logger};
  $logger->fatal ("plugin is in bad state") if
          $self->{__bad};
  return 1;
}
```

Die Variable `$container` enthält eine Referenz auf einen Hash, der dazu verwendet wird, Informationen zwischen Plug-Ins auszutauschen. Jedes Plug-In kann den Hash auslesen oder Informationen hinzufügen. Neue Informationen werden nur an darauf folgende Plug-Ins weitergegeben. Zu Beginn eines Durchlaufs der Ausführungstabelle wird der Hash gelöscht.

Die Variable `$logger` enthält eine Referenz auf die Log-Funktion des Daemons. Jedem Plug-In wird somit ermöglicht, Informationen in die Log-Datei zu schreiben.

9.1.2 Fehlerbehandlung

Treten Fehler in einem Plug-In auf, werden diese an den Daemon weitergegeben. Dieser protokolliert sie in der Log-Datei. Daraufhin bricht er die Ausführung des Plug-Ins ab und startet den nächsten Aufruf in der Ausführungstabelle. Tritt ein Fehler im Plug-In auf, erfolgt die Ausführung beim nächsten Durchlauf der Ausführungstabelle trotzdem wieder.

Der Daemon protokolliert Laufzeitfehler ebenfalls.

9.2 Imperia-Metafelder

In diesem Abschnitt finden Sie eine Liste der wichtigsten Metafelder eines Dokuments in Imperia.

__imperia_node_id

Diese Variable enthält die Node-ID eines Dokuments. Die Node-ID besteht aus der numerische ID der Rubrik und der numerischen ID des Dokuments. Diese IDs werden unter anderem dazu verwendet, den Verzeichnisbaum der Datenhaltung im Dateisystem aufzubauen.

__imperia_modified

Diese Imperia-Variable enthält das Datum der letzten Änderung an einem Dokument.

__imperia_category_template

In dieser Variablen wird das Template gespeichert, das zur Bearbeitung der Meta-Informationen der Rubrik dient. Ist diese Variable nicht gesetzt, wird ein Fallback-Template verwendet.

__imperia_created

Diese Variable speichert das Erstellungsdatum eines Dokuments.

__imperia_mime_content_type

Diese Variable enthält den MIME-TYPE des binären Contents, der in der Variablen __imperia_mime_content enthalten ist.

__imperia_mime_content

Diese Variable speichert ein binäres Objekt.

__imperia_mime_original

Diese Variable enthält den Dateinamen der hochgeladenen Datei.

__imperia_children

In dieser Variablen werden die Node-IDs der aus diesem Dokument geklonten Dokumente abgelegt. Sie ist das Pendant zu `__imperia_parent`.

__imperia_uid

Diese Variable enthält die ID des Benutzers, der ein Dokument erstellt hat.

__imperia_last_uid

Diese Variable enthält die ID des Benutzers, der ein Dokument zuletzt bearbeitet hat.

__imperia_template_chain

Mit dieser Variablen können Sie auf Rubriken- bzw. Dokumentenebene eine alternative Template-Chain festlegen.

__imperia_is_copy

Copy-Seiten werden durch diese Variable kenntlich gemacht.

__imperia_meta_expand

Diese Variable beinhaltet ein Flag, ob das Dokument sich noch im Meta-Mode befindet.

__imperia_imported

Diese Variable beinhaltet den UNIX-Timestamp des Zeitpunktes, bei dem das Dokument vom Hintergrunddienst Hermes importiert wurde.

__imperia_category

In dieser Variablen ist die Rubrik eines Dokuments gespeichert.

__imperia_parent

Diese Variable beinhaltet die NodeID des Ursprungsdokuments eines geklonten Dokuments (siehe auch `__imperia_children`).

__imperia_parent_filename

Diese Variable beinhaltet den Dateinamen der Hauptseite.

__imperia__parent_directory

Diese Variable beinhaltet das Verzeichnis der Hauptseite.

directory

In dieser Variablen ist der Pfad des Dokuments gespeichert.

filename

Diese Variable enthält den Dateinamen, unter dem das Dokument abgespeichert ist.

linguas (optional, von Rubrikinformationen geerbt)

Diese Variable notieren Sie in den Metainformationen von Rubriken. Geben Sie die Sprachversionen, die für Dokumente der betreffenden Rubrik auswählbar sein sollen als Elemente eines Arrays in dieser Variablen an. Die Sprachen notieren Sie mit ihrem jeweiligen ISO 639 Sprach-Code (**de**, **en**, **fr** etc.).

template

In dieser Variablen ist das Template des Dokuments gespeichert.

imperiablock_count_<ID>

Diese Variable speichert die Anzahl der vorhandenen Blockinstanzen in einem Imperiablock-Element. Die ID gibt dabei an, um welchen Imperiablock es sich handelt.

FLEXHISTORY_<ID>

In der Flexhistory sind die vorhandenen Flexmodule und deren Position in einem Flexmodul-Element gespeichert. Die ID gibt dabei an, um welches Element es sich handelt.

9.3 XML-Export

Der Zieldateiname kann beliebige Metafelder enthalten, z.B.:

```
/xmlexportdir/<!--XX-__imperia_node_id-->/<!--XX-filename-->
```

9.4 Mehrsprachige Dokumente mit Imperia-Multilanguage verwalten

Beim Multi-Language-Konzept von Imperia liegen die einzelnen Sprachversionen eines Dokuments jeweils in einer eigenen Copy-Seite vor (siehe auch Abschnitt 9.5 **Copy-Seiten, Beispiel mehrsprachige Dokumente** auf Seite 174). Die einzelnen Kopien unterscheiden sich im Dateinamen durch Anhängen des ISO 639 Sprach-Codes an die Dateiendung (`index.html.de`, `index.html.en`, ...). Die neuen Multi-Language-Features vereinfachen die Erzeugung und Bearbeitung dieser Copy-Seiten. Die folgenden Abschnitte beschreiben, wie Sie das Multi-Language-Feature von Imperia nutzen

9.4.1 Workflow

Der Workflow für Dokumente mit mehrsprachigen Inhalten muss vor dem Bearbeiten-Schritt das Plug-In „**Multilang**“ enthalten. Das Plug-In sorgt für die Bereitstellung der Eingabeelemente für die Sprachversionen im Bearbeiten-Schritt des Workflows und die automatische Erzeugung der Copy-Seiten.

9.4.2 Rubrikeinstellungen

Mit der Rubrik-Metavariablen `linguas` steuern Sie den Ablauf beim Anlegen mehrsprachiger Dokumente. Diese Variable speichert ein Array von aus zwei Buchstaben bestehenden Ländercodes. Diese Liste gibt an, welche Sprachversionen in der betreffenden Rubrik verfügbar sind. Die erste Sprache in dieser Liste zählt dabei als „Hauptsprache“. Imperia baut die Sprachauswahl im Meta-Edit-Schritt anhand dieses Arrays auf. Bitte beachten Sie, dass alle Dokumente ebenfalls eine Variable „linguas“ enthalten, die eine Teilmenge der Liste aus den Rubrikeinstellungen darstellt. Der Inhalt der Variablen im Master-Dokument ergibt sich aus der Auswahl, die der Redakteur im Meta-Edit-Schritt trifft. In den einzelnen Copy-Seiten ist nur der Wert der jeweiligen Sprachversion gespeichert.

9.4.3 Metadatei

In der Metadatei aktivieren Sie die Sprachauswahl mit der Direktive *MULTILANG_SELECTION*. Während des Meta-Edit-Schritts bietet Imperia daraufhin eine Sprachauswahl an. Mit der Variablen `__ip_no_multi-lang` können Sie die Multi-Language-Features für ein Dokument unterdrücken.

9.4.4 Template

Schließen Sie die Template-Elemente, deren Inhalte Sie in allen Sprachversionen getrennt erfassen wollen in die Kommentare <*!--multilang_start--*> und <*!--multilang_end--*> ein. Ein Template kann mehrere solcher Bereiche enthalten, Sie dürfen diese aber nicht schachteln. Für jede Sprachversion erzeugt Imperia automatisch ein Duplikat dieser Bereiche, das in einem Div-Tag eingeschlossen im fertigen Dokument erscheint. Durch eine ebenfalls automatisch eingefügte Sprachauswahl am Seitenanfang lassen sich die Sprachversionen der Bereiche ein- und ausblenden.

Innerhalb eines solchen Bereichs nimmt der Template-Prozessor folgende Ersetzungen vor:

- Die Zeichenketten *_lingua* und <*!--XX-lingua--*> werden durch den gegenwärtigen Ländercode des betreffenden Abschnittes ersetzt. Die XX-Ersetzung (ausschließlich für lingua und ohne jegliche Escaping-Modes) findet bereits zu diesem Zeitpunkt statt, weil die jeweilige Information häufig schon benötigt wird.
- Die Zeichenkette *ML_INDEX=25* wird durch *INDEX=2025* ersetzt. Das ist allerdings nicht ganz exakt ausgedrückt. Wenn Sie beispielsweise drei Sprachen verwenden bleibt die Zahl bei der ersten unverändert. Bei der zweiten Sprache kommen 1000 hinzu, bei der dritten 2000 und so weiter. Sollte der Wert 1000 nicht passen, können Sie mit der Systemkonfigurations-Variablen `ML_RANGE` einen beliebigen Wert festlegen. Wenn Sie in Ihrem Template die Berechnung im Template benötigen, verwenden Sie die Syntax <*!--ML_RANGE[25]--*>. Der Template-Prozessor wandelt dies in die Werte 25, 1025, 2025 usw. um.

Fügen Sie allen Metavariablen innerhalb eines Multilang-Bereichs das Suffix *_lingua* hinzu. So gewährleistet die automatische Ersetzung durch den Template-Prozessor deren Eindeutigkeit für die Sprachversionen. Die Variable `foobar_lingua` erscheint im Bearbeitungsmodus des Templates als *foobar_de*, *foobar_en*, *foobar_bg*, etc.

Flexmodule können Sie ausschließlich mit einem festen Index einsetzen, den Sie mit *ML_INDEX=* statt dem sonst üblichen *INDEX=* definieren müssen. Imperiablöcke innerhalb von Flexmodulen können Sie unverändert verwenden. Imperiablöcke außerhalb von Flexmodulen müssen ebenfalls mit *ML_INDEX=* einen festen Index bekommen.

Bei Metavariablen in Flexmodulen und Slots brauchen Sie das Suffix *_lingua* nicht anzugeben, weil deren Eindeutigkeit bereits durch die automatisch vergebenen Indizes sichergestellt ist. Nebenbei gesagt wäre die Verwendung zu diesem Zeitpunkt ohnehin zu spät, da aufgrund der Modulreihenfolge in der Template-Chain die Ersetzung bereits stattgefunden hat. Alle Dokumenten-Modi arbeiten auf die gleiche Weise, mit der Ausnahme, dass die Duplizierung der Multilang-Bereiche nur im Editmode stattfindet.

9.5 Copy-Seiten, Beispiel mehrsprachige Dokumente

Copy-Seiten sind von Imperia automatisch angelegte, zusätzliche Kopien eines Dokuments. Aus dem Original leiten sich die durch die copy-Anweisung generierten Dokumente ab. Die Kopien können auf die im Ursprungsdokument definierten Variablen zurückgreifen. Der Inhalt der Kopie sowie dessen Präsentation können dabei auch vom Original, der so genannten Masterpage, abweichen.

Für Copy-Pages benötigen Sie speziell angelegte Templates, die dann mit dem jeweils relevanten Teil der Metainformationen des Originaldokuments gefüllt und automatisch als zusätzliche HTML-Dateien abgespeichert werden. Imperia erzeugt die entsprechenden Dateien in dem Augenblick, in dem das Dokument den Workflow verlässt. Der Redakteur sieht und bearbeitet im Workflow die Masterpage und deren Meta-Informationen. Für Kontrollzwecke besteht im Bearbeiten-Schritt die Möglichkeit, mit einem zusätzlich eingeblendeten Auswahlfeld eine der Kopien für die Erzeugung einer Vorschau zu bestimmen.

Die Verwendungsmöglichkeiten von Copy-Pages sind vielfältig. Beispielsweise können Sie so verschiedene Sprachversionen, weitere Kopien in anderen Unterverzeichnissen oder für den Ausdruck optimierte Versionen eines Dokuments erstellen. Das folgende Beispiel veranschaulicht die Vorgehensweise anhand dreier Sprachversionen eines Dokuments.

Die Kopierfunktion von Imperia aktivieren Sie durch das Setzen des speziellen Metafelds `copy`. Die notwendigen Einstellungen dieses Metafelds können Sie bereits mit einer HIDDEN-Anweisung in der Metadatei vornehmen.

Hinweis:

Da die eigentliche Aufteilung der Information und das Erzeugen der Dateien erst am Ende des Workflows geschehen, ist es nicht zwingend notwendig, das copy-Metafeld im Meta-Edit-Schritt zu setzen. Andere Workflowschritte, z.B. einen Metasetter oder auch einen Bearbeiten-Schritt können Sie hierfür ebensogut nutzen.

Beachten Sie aber, dass Sie dann keinen Zugriff auf die besonderen Funktionen des Meta-Edits wie beispielsweise count und copycount haben.

Ein Metafile für drei Sprachkopien könnte wie folgt aussehen:

```
    TITLE "Multilanguage Example"
    AUTHOR "Imperia 8"


❶ #IF ("<!--XX-METAMODE-->")
❷  HIDDEN "directory:<!--XX-directory-->/<!--count-->"

❸  PRINT "English:<!--XX-directory-->/<!--copycount-->/index.html.en"
        PRINT "Deutsch:<!--XX-directory-->/<!--copycount-->/index.html.de"
        PRINT "Français:<!--XX-directory-->/<!--copycount-->/index.html.fr"

        //Kopien beim ersten Durchlauf initialisieren
❹
        HIDDEN "copy:<!--XX-directory-->/<!--copycount-->/ \\
index.html.en:TEMPLATE=i18n_en"
        HIDDEN "copy:<!--XX-directory-->/<!--copycount-->/ \\
index.html.de:TEMPLATE=i18n_de"
        HIDDEN "copy:<!--XX-directory-->/<!--copycount-->/ \\
index.html.fr:TEMPLATE=i18n_fr"

    #ELSE
        HIDDEN "directory:<!--XX-directory-->
```

```
        PRINT "English:<!--XX-directory-->/index.html.en"
        PRINT "Deutsch:<!--XX-directory-->/index.html.de"
        PRINT "Français:<!--XX-directory-->/index.html.fr"

    #ENDIF
❺
    SELECTION "no_publish:Quelldokument veröffentlichen"
        OPTION "1:Nein"
        OPTION "0:Ja"
    ENDSEL
```

Hinweis:

Aus Darstellungsgründen wurde der obige Quelltext teilweise umbrochen.

❶ Die hier gezeigte IF-Abfrage wird in Abschnitt 2.5.1 **Zähler im Verzeichnisnamen** auf Seite 23 näher erläutert.

❷ Dieser Abschnitt der Metadatei bestimmt das Verzeichnis, in dem das Masterdokument und die fertigen Kopien abgelegt werden.

❸ Dieser Code-Abschnitt stellt die Pfade und Dateinamen der einzelnen Copy-Dokumente in der Eingabemaske des Meta-Edit-Schritts dar.

❹ Die drei HIDDEN-Anweisungen belegen das spezielle Metafeld `copy` mit den notwendigen Informationen für die Copy-Seiten. Hier sind nur die nötigsten Informationen, nämlich Pfad, Dateiname und anzuwendendes Template für die jeweilige Kopie angegeben. Weitere Einstellungsmöglichkeiten für Seitenkopien finden Sie in Abschnitt 2.4.1 **copy** auf Seite 19.

❺ Mit der Funktion *no_publish* legen Sie fest, ob neben den Copy-Dokumenten mit den einzelnen Sprachversionen auch das Quelldokument veröffentlicht werden soll.Anstatt eines HIDDEN-Felds verwendet das Beispiel eine Auswahlliste. Über diese Auswahlliste erhält die Variable den Wert `1` oder `0` (siehe auch Abschnitt 2.4.6 **no_publish** auf Seite 23).

Damit Imperia den jeweiligen Kopien die spezifischen Inhalte zuordnen kann, müssen im Template des Quelldokuments die Eingabeelemente für die einzelnen Sprachversionen gekennzeichnet werden, beispielsweise durch Anhängen des Länderkürzels (_de, _en, _fr). Bei der einfachsten Variante stellen Sie die Eingabefelder für alle Sprachen in einem Template gleichzeitig zur Verfügung:

```
   [...]
#IF(<!--XX-editmode-->)

   Deutsche &Uuml;berschrift:<br />
     <input name="IMPERIA:headline_de" type="text">
   <br />
   Deutscher Text:<br />
     <textarea name="IMPERIA:HTMLBR:text_de" cols="40" rows="20">
     </textarea>
   <br />
   English headline:<br />
     <input name="IMPERIA:headline_en" type="text">
   <br />
   English text:<br />
     <textarea name="IMPERIA:HTMLBR:text_en" cols="40" rows="20">
     </textarea>
   <br />
   Titre fran&ccedil;ais:<br />
     <input name="IMPERIA:headline_fr" type="text">
   <br />
   Texte fran&ccedil;ais:<br />
     <textarea name="IMPERIA:HTMLBR:text_fr" cols="40" rows="20">
     </textarea>
   <br />
```

```
#ELSE
   [...]
```

Hinweis:

Obenstehendes Listing stellt lediglich einen Ausschnitt aus dem Template dar.

Zusätzlich legen Sie für jede Sprachversion ein Ausgabetemplate an. Darin brauchen Sie allerdings lediglich auf die jeweiligen Metafelder zuzugreifen. Die Variablen in den Templates der Copy-Dokumente und die Eingabeelemente für die entsprechende Sprache im Template der Masterpage müssen den gleichen Namen haben.

Beispiel deutsches Ausgabetemplate:

```
<html>
  <head>
    <title>Dummy-Template</title>
   <!--IMPERIA:ONECLICKEDIT-->
  </head>

  <body>

<!--formstart-->
<!--template-description:Dummy-Template-->
                <h2><!--XX-headline_de--></h2>
                <p><!--XX-text_de--></p>
<!--formend-->
  </body>
</html>
```

Für die Erzeugung mehrsprachiger Inhalte gibt es eine Vielzahl von Ansätzen, von denen der hier gezeigte sicher die simpelste Variante ist. Beispielsweise ist es nicht zwingend notwendig Ein- und Ausgabe-Templates voneinander zu trennen. Auch muss die Ausgabe der einzelnen Sprachen nicht unbedingt jeweils durch ein eigenes Template geregelt werden. Vielmehr können Sie dies alles auch in einem einzigen Template abwickeln. In diesem Fall müssen Sie aber an anderer Stelle für die Aufteilung der Inhalte sorgen, z.B. durch Änderungen im Workflow.

9.6 Das Metatool 1

Hinweis:

Die folgenden Abschnitte beschreiben das Metatool 1, das aus Imperia 7 stammt und aus Kompatibilitätsgründen auch in Imperia 8 weiterhin unterstützt wird. In zukünftigen Imperia-Versionen wird dies voraussichtlich nicht mehr der Fall sein.

Wenn Sie ein neues Projekt entwickeln, empfehlen wir, direkt das verbesserte Metatool 2 zu verwenden. Lesen Sie mehr über die Verwendung des Metatools 2 ab Abschnitt 3.7.1 **Aufruf des Metatools 2 im Template** *auf Seite 80.*

9.6.1 Einbau des Metatools 1 in ein Template

Der Aufruf des Metatools im Template geschieht über einen Anker mit einem JavaScript als Ziel. In diesem Aufruf werden alle gewünschten Übergabe-Parameter eingefügt. Beim folgenden Aufruf handelt es sich um ein Beispiel, in dem aus Gründen der Übersicht die einzelnen Übergabe-Parameter untereinander dargestellt werden.

Im tatsächlichen Aufruf sollten die Parameter ohne Umbrüche hintereinander geschrieben und durch Doppelpunkte voneinander getrennt werden. Grundsätzlich stellt sich der Aufruf für das Metatool wie folgt dar:

```
<a href="javascript:var win=open('/cgi-bin/site_metatool.pl?
```

key1=value1:key2=value2:...keyN=valueN:

```
','','' tool-bar=
no,width=550,height=550,scrollbars=1,scrolling=1,menubar=no,
location=no,copyhistory=no')">
 <img src="/imperia/images/design/teaser_v.gif" border=0 width=28
 height=28></a>
```

Hinweis:

Eine Erläuterung der einzelnen Parameter finden Sie in Kapitel 9.6.4.

9.6.2 Aufruf in Imperia-Flexmodulen

Das Metatool kann aus Flexmodulen heraus aufgerufen werden.

```
MYURL=<!--XX-_imperia_node_id-->_<!--FLEX_INDEX-->_<!--FLEX_ID-->
```

Achtung:

Beachten Sie hierbei, dass als Wert von MYURL nicht der URI verwendet werden kann! Sonst würde beim Aufruf aus einer neuen Flexinstanz die schon getroffenen Auswahlen für alle vorhergehenden Flexinstanzen als vorgewähltes Feature 1 angezeigt. Diese müssten dann erst einzeln wieder deaktiviert werden.

9.6.3 Imperia-Blocks füllen

Mit Hilfe des Metatools können auch Imperia-Blocks gefüllt werden. Dadurch kann man beispielsweise beliebig viele Links mit Hilfe des Metatools erzeugen und in die Inputfelder eines Imperia-Blocks eintragen.

Analog zum Flexmodul kann das Metatool auch aus einem Imperiablock heraus aufgerufen werden.

```
MYURL=<!--XX-directory-->_<!--XX-filename-->
```

Achtung:

Das aufrufende Dokument wird ebenfalls in der Liste der zu verlinkenden Möglichkeiten angeboten. Es muss also darauf geachtet werden, dass dieses dabei nicht mit ausgewählt wird.

9.6.4 Übergabe-Parameter

Die Übergabe-Parameter bestimmen die Oberfläche und die Funktionen des Metatools beim Aufruf aus einem Template. Im Folgenden werden die Übergabe-Parameter aufgelistet:

MYURL = <!--XX-directory-->/<!--XX-filename-->

Dies ist die URL der Seite, von der aus das Metatool aufgerufen wurde. Die URL sollte durch die Meta-Variablen-Kombination <!--XX-directory-->/<!--XX-filename--> gebildet werden. Diese URL dient dazu, die Einstellungen des Metatools beim erneuten Editieren der Seite wiederherzustellen.

PICSYNC = *bool*

Mit diesem Parameter wird eingestellt, ob Bilder zwischen dem Quelldokument und dem Zieldokument synchronisiert werden sollen. Mögliche Werte sind:

- `0`, Bilder werden nicht synchronisiert
- `1`, Bilder werden synchronisiert

IMPERIABLOCK = *string*

Über diesen Parameter steuern Sie den Zugriff auf Metafelder im aktuellen Dokument. Der Parameter *string* enthält dabei den Index des Imperia-Blocks, den Sie füllen möchten und wird an den entsprechenden Metafeldnamen angehängt.

IMPERIASUFFIX = *string*

Über diesen Parameter steuern Sie den Zugriff auf Metafelder im aktuellen Dokument. Der Parameter *string* enthält dabei Index und ID des Flexmoduls, das Sie füllen möchten und wird an den entsprechenden Metafeldnamen angehängt. Die Verwendung dieses Parameters ermöglicht Ihnen die Verwendung des Metatools innerhalb von Flexmodulen.

ANZMAINS = *int*

Über diesen Parameter wird die Anzahl der Feature-Elemente im Zieldokument definiert. Eine Linkliste kann bis zu 25 Elemente enthalten, jedoch können nur so viele Feature-Felder verwendet werden, wie im Template vorgesehen.

SELSYNC_*Meta-Variable* =

Über diesen Parameter werden die Verbindungen zwischen den Meta-Variablen des Quelldokuments und den Meta-Variablen des Zieldokuments hergestellt. Der Platzhalter *Meta-Variable* wird in diesem Parameter durch den Namen der Meta-Variablen des Zieldokuments ersetzt. Hinter dem Gleichheitszeichen wird der Name der Meta-Variablen aus dem Quelldokument angegeben:

```
SELSYNC_headline = title
```

Mit dieser Zeile wird der Inhalt der Meta-Variablen `title` in die Meta-Variable `headline` des Zieldokuments kopiert.

FEATURE(\d+)_NAME =

Mit diesem Parameter werden FEATURE-Namen übergeben, die hinter dem Gleichheitszeichen definiert werden.

FEATURE(\d+)_ANZMAINS =

Mit diesem Parameter wird die Anzahl der neuen FEATURE-Elemente angegeben.

FEATURE(\d+)_SYNC_*Meta-Variable* =

Dieser Parameter gibt an, welche Meta-Variable in einem Feature-Set mit einer Meta-Variable aus dem Quelldokument synchronisiert werden soll. Der Platzhalter *Meta-Variable* wird durch den Namen der Meta-Variable im Zieldokument ersetzt. Hinter dem Gleichheitszeichen muss der Name der Meta-Variable im Quelldokument angegeben werden.

CATEGORY = *string*

Nutzen Sie diesen Parameter zur Einschränkung des Rubrikenbaums auf einen bestimmten Teilbaum. Hinter dem Gleichheitszeichen notieren Sie einen Rubrikennamen oder eine Node-ID.

CHARSET = *string*

Mit diesem Parameter können Sie den in Imperia eingestellten Standard-Zeichensatz umgehen.

Achtung:

Setzen Sie diesen Parameter mit Vorsicht ein, da es zu fehlerhafter Textdarstellung kommen kann.

DISPLAY_CONTENT(\d+) =

Dieser Parameter gibt an, von welchen Meta-Variablen der Inhalt im Fenster des Metatools angezeigt werden soll. Hinter dem Gleichheitszeichen wird der Name der entsprechenden Meta-Variablen aus dem Quelldokument angegeben.

DISPLAY_IMAGE =

Dieser Parameter stellt ein, welches Bild im Fenster des Metatools angezeigt werden soll. Hinter dem Gleichheitszeichen muss der Name des Bildes angegeben werden.

DISPLAY_LISTELEMS =

Mit diesem Parameter bestimmt man die Anzahl der angezeigten Elemente in der Linkliste des Metatools. Geben Sie hinter dem Gleichheitszeichen einen numerischen Wert ein.

DISPLAY_TITLE =

Dieser Parameter stellt ein, welcher Titel im Fenster des Metatools angezeigt werden soll. Hinter dem Gleichheitszeichen muss der Name der Meta-Variablen aus dem Quelldokument angegeben werden.

DISPLAY_PERPAGE =

Dieser Parameter stellt ein, wie viele Dokumente im Metatool angezeigt werden. Der Standardwert ist 20.

DOCSTATUS =

Mit Hilfe dieses Parameters können die Treffer gefiltert werden. Folgende Werte können verwendet werden: `W` (= nur Dokumente, die sich im Workflow befinden), `L` (= nur Dokumente vom Zielsystem) oder `P` (= nur Dokumente, die sich in der Freischaltliste befinden).

Es können gleichzeitig mehrere Parameter übergeben werden. Daraus ergibt sich eine Liste von Dokumenten, die jeweils mit den angegebenen Parametern referenziert werden.

Beispiel:

```
DOCSTATUS=WP
```

Mit diesem Parameter werden nur Dokumente als Treffer angezeigt, die sich im Workflow oder in der Freischaltliste befinden.

HIDE_TITLE =

Hiermit wird eingestellt, ob ein Titel angezeigt wird. Mögliche Werte sind: `1` (= Titel wird nicht angezeigt) oder `0` (= Titel wird angezeigt).

SORTBY =

Dieser Parameter dient der Sortierung der Anzeige im Metatool. Hinter dem Gleichheitszeichen wird eine Meta-Variable angegeben, nach der sortiert wird. Es wird, je nach Inhalt der Meta-Variablen, numerisch oder alphabetisch aufsteigend sortiert.

SELFILTER =

Mit der Verwendung dieses Parameters kann der User einen oder mehrere, vom Programmierer des Templates zur Einschränkung der angezeigten Dateien vorgesehene Parameter ändern. Wird dieser Parameter verwendet, erscheint auf der Metatool-Oberfläche ein Eingabefeld zur Änderung des Einschränkungsparameters sowie ein Such-Button, der die Anzeige entsprechend der Änderung aktualisiert. Aus Kompatibilitätsgründen werden nur kleingeschriebene Meta-Variablen-Werte verwaltet.

STOREFILTER =

Mit dieser Option kann die Änderung des Einschränkungsparameters für SELFILTER gespeichert werden, so dass der User bei erneutem Start des Metatools die geänderten Parameter wieder sieht. Mögliche Werte sind `1` (= die Änderungen, die der User vorgenommen hat, werden gespeichert) oder `0` (= die Änderungen, die der User vorgenommen hat, werden nicht gespeichert).

Index

Symbole

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z