ADDISON-WESLEY

PEARSON Education

open source library

inkl. 1,00 € Spende an TYPO3

Joscha Feth

Das TYPO3 Anwenderhandbuch

Websites erstellen, gestalten und verwalten mit Version 4

ADDISON-WESLEY

6 TypoScript

6.1 TypoScript

TypoScript ist die TYPO3-eigene Beschreibungssprache. Beschreibungssprache deshalb, weil TypoScript prinzipiell nur strukturiert – TypoScript selbst macht nichts, sondern wird nur verwendet, um Baumstrukturen (u.a. auch Konfigurationselemente) zu beschreiben.

Der Name *TypoScript* ist missverständlich, aber als dies auffiel, war es bereits zu spät, den Namen aus sämtlicher Literatur zu tilgen. TypoScript wird nicht ausgeführt, sondern geparst und in ein PHP-Array transformiert. Dieses PHP-Array wiederum wird vom TYPO3-Kern interpretiert und nach dessen Informationen dann die Ausgabe erstellt.

Beispiel:

Die TypoScript-Deklararionen:

```
myObject {
      myProperty = 100
      subObject.subProperty = Hallo
}
```

werden in ein Array mit folgender Struktur umgeformt:

```
$TS['myObject.']['myProperty'] = 100;
$TS['myObject.']['subObject.']['subProperty'] = 'Hallo';
```

Zu TypoScript selbst gibt es eine ausführliche Sprachreferenz[1] (TSRef), die Ihnen an vielen Stellen weiterhelfen wird – in diesem Buch werde ich nur die grundlegende Syntax und einige, häufig verwendete Objekte beschreiben.

6.1.1 Grundlegende Sprachelemente

Mehr zur Syntax finden Sie auch auf *typ3.org* in der TypoScript Referenz[2].

1 *http://typo3.org/documentation/document-library/references/doc_core_tsconfig/current/view/*
2 *http://typo3.org/documentation/document-library/doc_core_ts/TypoScript_Syntax/*

Achtung

- TypoScript ist case-sensitive, d.h. es wird zwischen Groß- und Kleinschreibung unterschieden.
- TypoScript wird sukzessive geparst, d.h. später deklarierte, gleichnamige Variablen überschreiben zuvor deklarierte.

Zuweisungen

```
a      = 111
a.b.c  = elfeins
```

Der Variablen links vom Gleichheitszeichen wird der Wert rechts vom Gleichheitszeichen zugewiesen.

Da mit TypoScript Baumstrukturen dargestellt werden, kann die Variable links aus einem (siehe Zeile 1) oder mehreren (siehe Zeile 2) Teilen bestehen.

Hinweis

Mit einfachen Zuweisungen sind nur einzeilige Werte möglich. Wie mehrzeilige Werte zu definieren sind, wird im Abschnitt »Mehrzeilige Werte«, auf Seite 493 gezeigt.

Konkatenation

(Operator existiert seit TYPO3 Version 4.0.)

```
meineListe = 1,2,3

meineListe := addToList(4)
```

Der kommaseparierten Liste `meineListe` wird der Wert `4` angehängt. Der Wert der Variable lautet am Ende `1,2,3,4`.

Erinnern Sie sich an diese »Funktion« bei der Definition von Konfigurationseinstellungen, die dynamisch erzeugt werden (beispielsweise abhängig von verschiedenen Bedingungen).

Hinweis

Das nötige Komma wird automatisch eingefügt und muss nicht explizit angegeben werden.

Kommentare

```
01 / Das ist ein einzeiliger Kommentar
02 # Dieser Kommentar ist ebenfalls einzeilig
03 /* Dieser Kommentar
04      ist mehrzeilig */
```

Einzeilige Kommentare können durch / oder # eingeleitet werden. Mehrzeilige Kommentare müssen von /* */ umschlossen sein.

Kopien

```
a < b
```

Mit dem <-Operator kann eine Instanz eines Objekts kopiert werden. Im Beispiel wird der Variablen a eine Kopie des Objekts b zugewiesen.

Hinweis

Es wird eine tatsächliche Kopie des Objekts zum Zuweisungszeitpunkt erstellt. Dies bedeutet, dass der Wert der Variable a nicht geändert wird, wenn Sie den Wert der Variablen b nach der Zuweisung verändern.

Beispiel:

```
01 a = TEXT
02 a.value = Hallo Du!
03
04 b < a
05 a.value = Hallo Welt!
```

Das Objekt a hat am Ende den Wert Hallo Welt!, während das Objekt b immer noch den Wert Hallo Du! hat.

Referenzen

```
a =< b
```

Folgt nach dem Zuweisungsoperator = ein <-Operator, wird der Inhalt des links stehenden Objekts auf den Pfad rechts vom <-Operator gesetzt.

Referenzen unterscheiden sich von Kopien darin, dass eine spätere Änderung des Inhalts des als Referenz angegebenen Objektpfades sich auch nach der Referenzierung auswirkt.

Beispiel:

```
01 a = TEXT
02 a.value = Hallo Du!
03
04 b =< a
05 a.value = Hallo Welt!
```

Sowohl das Objekt `a` als auch das Objekt `b` haben am Ende den Wert `Hallo Welt!`.

Löschen von Objekten

```
a >
```

Mit dem >-Operator können Variablen aus dem Namensraum entfernt werden. Im Beispiel wird die Variable `a` gelöscht (aus dem Namensraum entfernt).

Beispiel:

```
01 a = TEXT
02 a.wrap = <strong>|</strong>
03 a.value = Hallo Welt!
04 a.wrap >
```

Das Objekt `a` hat am Ende den Wert `Hallo Welt!`. Die in Zeile zwei definierten umgebenen `<strong>`-Tags wurden in Zeile vier wieder entfernt.

Konstanten (Constants)

```
a = {$Konstante}
```

Konstanten können in TypoScript verwendet werden, sofern sie zuvor im Constants-Feld definiert wurden.

Hinweis

Konstanten sind einfache Zeichenketten. Um diese aber von TS unterscheiden zu können, werden sie besonders markiert. Aus einer im Constants-Feld definierten Konstanten `xyz` wird so im TypoScript (Setup-Feld etc.) eine Konstante `{$xyz}`.

Bedingungen (Conditions)

Mit `[Bedingung]` können Bedingungen deklariert werden. Das erste Zeichen der Zeile muss hierbei eine eckige Klammer `[` sein. Die zu vergleichenden Werte werden vor dem Vergleich getrimmt (Whitespaces entfernt). Es gibt einige vorgefertigte Parameter, die in Bedingungen verwendet werden können. Diese finden Sie in der TSRef[3].

```
01 [system=win][browser=msie]
02     # System ist Windows oder der Browser ist Internet Explorer
03 [else]
04     # System ist weder Windows noch ist der Browser Internet Explorer
05 [end]
```

Bedingungen können verwendet werden, um bestimmte TypoScript-Deklarationen nur zu aktivieren, wenn die Bedingungen erfüllt sind. Mehrere Bedingungen einer Zeile werden mit »oder« (Vorsicht: nicht mit »und«) verknüpft.

Der Teil hinter `[else]` wird aktiviert, wenn alle Bedingungen nicht erfüllt sind.

Möchten Sie bei einer Bedingung mehrere Werte zulassen, können Sie die erlaubten Werte durch Kommata trennen:

```
01 [browser = netscape, opera]
02     # Der Browser ist Netscape oder Opera
03 [end]
```

Um eine »und«-Verknüpfung zu schaffen, können die Bedingungen geschachtelt werden.

```
01 [system=win]
02     # Das System ist Windows
03     [browser=msie]
04     # Das System ist Windows und der Browser ist Internet Explorer
05     [end]
06 [end]
```

Hinweis

Bedingungen werden in Beispielen auch häufig mit `[global]` anstelle von `[end]` beendet. Diese beiden Schlüsselwörter werden gleich behandelt und können daher synonym eingesetzt werden.

Mehrzeilige Werte

Bei einer normalen Zuweisung können einer Variablen keine mehrzeiligen Werte zugewiesen werden. Steuerungszeichen wie `\n` werden als ganz normaler Text interpretiert und Zeilenumbrüche verursachen einen Fehler. Um mehrzeilige Werte dennoch möglich zu machen, muss folgende Syntax verwendet werden:

3 *http://typo3.org/documentation/document-library/references/doc_core_tsref/current/view/4/1/*

```
01 a = TEXT
02 a.value (
03     Dieser Wert
04     ist
05     mehrzeilig
06 )
```

Hinweis

Whitespaces werden nicht ignoriert, d.h. in unserem Fall hätten wir links vom Text in jeder Zeile einen Tabulator.

Möchten Sie Whitespaces verhindern, so müssen Sie die einzelnen Zeilen immer an den Zeilenanfang schreiben.

Achtung

Die Zuweisung geschieht ohne Gleichheitszeichen (=). In der Zeile, wo sich öffnende und schließende Klammer befinden, darf kein Text eingefügt werden.

Verschachtelung

Im Normalfall werden Zuweisungen so geschrieben:

```
01 a         = A
02 a.b       = B
03 a.b.c     = C
04 a.b.d     < a.b.c
```

Hat man nun aber auf einer Ebene mehrere Zuweisungen (hier die letzten zwei Zeilen), kann es sinnvoll sein, diese so zu schreiben:

```
01 a         = A
02 a.b = B   /* Hier lohnt es sich noch nicht zu schachteln */
03 a.b {           /* Hier schon. Auf dieser Ebene sind zwei Aktionen */
04     c = C
05     d < .c
06 }
```

In unserem Beispiel sind die Variablennamen noch relativ kurz, und wir befinden uns erst in der 2. Ebene, daher ist der geschriebene Code etwa gleich lang. Haben Sie aber einmal mehr Ebenen und längere Variablennamen, steigt die Übersichtlichkeit extrem, wenn Sie Abschnitte einsetzen.

Hinweis

Diese Methode ist auch sehr sinnvoll, wenn Sie Kopien von Objekten benötigen (im Beispiel soll `a.b.d` eine Kopie von `a.b.c` sein). Hier muss innerhalb einer Ebene nicht mehr der volle Pfad angegeben werden, sondern dem Variablennamen lediglich ein Punkt (`.`) vorangestellt werden.

TypoScript aus externer Datei laden

TypoScript-Deklarationen können aus externen Dateien geladen werden. Dies funktioniert beispielsweise in TypoScript-Templates, der Seiten-TSconfig und der User-TSconfig.

Hierzu müssen Sie in das entsprechende Feld folgende Anweisung schreiben:

```
<INCLUDE_TYPOSCRIPT: source="FILE:[Pfad zur Datei]">
```

Möchten Sie also beispielsweise TypoScript aus der Datei `meinTS.txt` im `fileadmin/`-Verzeichnis laden, würden Sie folgende Deklaration in das Feld einfügen:

```
<INCLUDE_TYPOSCRIPT: source="FILE:fileadmin/meinTS.txt">
```

Hinweis

Die Anweisung muss in einer eigenen Zeile stehen – in derselben Zeile dürfen keine anderen Deklarationen existieren.

Achtung

Aus extern geladenen Dateien können nicht wiederum externe Dateien geladen werden. D.h. eine `<INCLUDE_TYPOSCRIPT:…>`-Deklaration in einer externen Datei wird ignoriert.

6.1.2 TypoScript auslagern

Häufig haben Webseiten nur wenige oder gar ein einziges Template.

Im Setup-Feld häuft sich daher meist eine Menge TypoScript.

Auch wenn man bei TypoScript nicht von Funktionen sprechen kann, so lässt sich die Konfiguration dennoch oft in einzelne, zusammengehörige Objekte aufteilen.

Wenn man diese Objekte nun in andere Templates auslagert (und möglicherweise noch als Objekte im `temp.`-Namensraum anlegt), so erhält man wieder verwendbaren und übersichtlichen Code. Im Haupt-Template könnten dann beispielsweise nur noch die Konfiguration des `PAGE`-Objekts und die `config`-Variablen stehen. So zumindest der Gedanke.

Beipiel:

```
01 page = PAGE
02 page.10          = TEXT
03 page.10.data     = page:title
04 page.20          = TEXT
05 page.20.data     = page:title
06 page.20.case     = upper
```

lässt sich schreiben als:

```
01 temp.title       = TEXT
02 temp.title.data  = page:title
03
04 page = PAGE
05 page.10 < temp.title
06 page.20 < temp.title
07 page.20.case = upper
```

Hierbei könnte das Objekt `temp.title` ohne Probleme in ein anderes Template ausgelagert werden.

Schritt 1: Bringen Sie den Inhalt Ihres Templates, wenn möglich, auf eine Form, die sich in einzelne Objekte untergliedert. Diese einzelnen Objekte lassen sich dann ohne weiteres in anderen Templates unterbringen.

Schritt 2: Fügen Sie den Teil des Codes in die Zwischenablage ein, den Sie in ein anderes Template auslagern möchten.

Schritt 3: Wählen Sie die Seite (dies kann z.B. ein `SysFolder` sein), in der Sie ein Template erstellen möchten. Existiert in der Seite bereits ein Template, so wechseln Sie in die Listen-Ansicht, andernfalls in die Template-Ansicht.

Befinden Sie sich in der Listen-Ansicht, klicken Sie im Abschnitt TEMPLATE auf das Icon ganz rechts, um ein neues Template zu erstellen (siehe Abbildung 6.1).

Abbildung 6.1: Per Klick auf das Icon rechts wird ein neues Template in der Seite erstellt

Im folgenden Dialog gibt es nur eine Pflichtangabe, das ist der Template-Titel. Prinzipiell können Sie eingeben, was Sie möchten, allerdings ist es sinnvoll, das Template als Extension Template zu kennzeichnen. Hierzu hat sich die Voranstellung eines + vor den eigentlichen Titel bewährt (siehe Abbildung 6.2).

Abbildung 6.2: Durch ein + vor dem Titel wird das Template als Extension Template gekennzeichnet

Befinden Sie sich in der Template-Ansicht und existiert noch kein Template in der aktuellen Seite, so können Sie durch einen Klick auf CLICK HERE TO CREATE AN EXTENSION TEMPLATE ein neues Extension Template erstellen (siehe Abbildung 6.3). Bestätigen Sie die auftauchende Sicherheitsabfrage mit Klick auf OK, um das Template zu erstellen.

Hinweis

Bei dieser Methode wird das Template automatisch *+ext* genannt. Sie können den Namen jederzeit ändern.

Abbildung 6.3: Per Klick auf den Button wird ein neues Template in der aktuellen Seite erstellt

Schritt 4: Öffnen Sie das Template zur Bearbeitung und fügen Sie in das `Setup`-Feld den zuvor ausgeschnittenen Code ein.

Sie können das Template entweder über die Template-Ansicht zur Bearbeitung öffnen – dort auch nur das `Setup`-Feld allein – oder über die List-Ansicht per Kontextmenü.

Speichern und schließen Sie das Template danach.

Schritt 5: Wechseln Sie wieder in Ihr Haupt-Template und wählen Sie dort im Feld INCLUDE BASIS TEMPLATE über den TYPO3 Element Browser (siehe Kapitel 4.8.1, »Der TYPO3 Element Browser«, auf Seite 340) das soeben angelegte Extension Template.

Über diese Einstellung werden die gewählten Templates (und somit auch der Inhalt des SETUP-Feldes) geladen und die definierten Objekte stehen im Haupt-Template zur Verfügung, als ob sie im Template-eigenen Setup-Feld stehen würden.

Speichern und schließen Sie das Haupt-Template.

Hinweis

Die Ladereihenfolge der eingebundenen Extension Templates entspricht der Reihenfolge im `Include basis template`-Feld. Das Haupt-Template wird als letztes geladen und hat damit die höchste Priorität.

Sie können die Ladereihenfolge über den Template Analyzer (siehe Abschnitt »Der Template Analyzer«, auf Seite 398) einsehen.

Abbildung 6.4: Per Klick auf das Ordnersymbol rechts vom Auswahlfeld wird der TYPO3 Element Browser aufgerufen, mit dem Sie ein Extension Template wählen können

Schritt 6: Leeren Sie den Frontend-Cache, um die Einstellungen wirksam zu machen.

6.1.3 Konfiguration (CONFIG-Objekt)

E-Mail-Adressen schützen

Kodierung

Um die verlinkten E-Mail-Adressen zu verschlüsseln, damit kein Bot sie erkennen kann, steht die TypoScript-Option CONFIG.SPAMPROTECTEMAILADDRESSES zur Verfügung.

Mit dieser Option können Sie die Kodierung von E-Mail-Adressen in Typolinks bei der Ausgabe beeinflussen:

```
config.spamProtectEmailAdresses = -3
```

Bei dieser Einstellung würde der ASCII-Wert aller Zeichen einer E-Mail-Adresse um 3 (nach links) verschoben. Damit der Besucher der Seite dennoch die korrekte E-Mail-Adresse sieht und auch beim Anklicken des Links entsprechend antwortet, wird ein kleines JavaScript benötigt, das bei Aktivierung dieser Option automatisch im Header der Seite eingebunden wird.

Möchten Sie kein JavaScript verwenden, können Sie die den Wert der Option auf ASCII setzen:

```
config.spamProtectEmailAdresses = ascii
```

Damit würden die einzelnen Buchstaben der E-Mail-Adresse in ihre Unicode-Entities umgewandelt. Der Vorteil dieser Methode ist, dass kein JavaScript benötigt wird.

@ ersetzen

Über eine weitere Einstellung können Sie das @-Zeichen ersetzen, um die E-Mail-Adressen weiter unkenntlich zu machen:

```
config.spamProtectEmailAddresses_atSubst = [at]
```

ersetzt in allen E-Mail-Links das `@` durch `[at]`.

Aus `a.b@c.de` würde dann `a.b[at]c.de`.

Standardmäßig steht der Wert dieser Einstellung auf `(at)`.

Punkt ersetzen

Über die Einstellung `spamProtectEmailAddresses_lastDotSubst` kann der letzte Punkt in einer E-Mail-Adresse ersetzt werden:

```
config.spamProtectEmailAddresses_lastDotSubst = [dot]
```

Aus `a.b@c.de` würde dann `a.b@c[dot]de`.

Standardmäßig steht der Wert dieser Einstellung auf `.` (Punkt), d.h. keine Ersetzung.

Standardsprache ändern

Die Standardsprache von TYPO3 ist Englisch. Deshalb sind die Spracheinstellungen standardmäßig auch auf Englisch eingestellt.

Möchten Sie nun eine Seite in einer anderen Sprache betreiben und benötigen kein Englisch, ist es sinnvoll, diese Konfiguration zu ändern.

Dies können Sie über mehrere Einstellungen des CONFIG-Objektes erreichen.

```
01 config {
02    # Sprache festlegen, wichtig für die korrekte Anzeige von Labels
03    language             = de
04    # Locale festlegen - wichtig für Formatierung von Datum/Uhrzeit
05    locale_all           = german
06    # Wichtig für korrekte lang-Attribute
07    htmlTag_langKey      = de-DE
08 }
```

Durch diese drei Einstellungen wird Ihre Seite vollständig in der gewählten Sprache (hier Deutsche Sprache in Deutschland) lokalisiert.

Mehr zur Lokalisierung finden Sie auch in Kapitel 9.1, »Mehrsprachigkeit«, auf Seite 623. Dort sind auch die einzelnen Einstellungen näher beschrieben. Lesen Sie unbedingt das Kapitel, da sich einige Einstellungen (beispielsweise die `locales` – siehe Zeile 5) je nach Betriebssystem unterscheiden.

Medien-Basis einstellen

Manchmal ist es hilfreich, eine Medienbasis festzulegen (bei Einsatz von Templates, RealURL etc.), um Links zu externen Medien (Bildern, Stylesheets, Skripten etc.) konsistent zu halten. So können Sie mit relativen Pfaden arbeiten und die Basis später anpassen, sollte sich die Seite einmal verschieben.

Dies können Sie mit folgender Konfigurationseinstellung erreichen:

```
config.baseURL = http://www.domain.de/
```

Mit dieser Einstellung wird ein `<base>`-Tag in jede Seite eingefügt, das dafür sorgt, dass die Basis aller relativen Links die angegebene URL ist.

Achtung

In der TypoScript-Referenz wird die Möglichkeit angegeben, der Einstellung `config.baseURL` einen booleschen Wert zuzuweisen, um die URL der aktuellen Seite als Basis einzufügen. Dies wurde in TYPO3 Version 3.8.1 aus Sicherheitsgründen deaktiviert. Sie sollten daher immer eine volle URL als Basis angeben. Benötigen Sie dennoch die Angabe verschiedener Basen, können Sie dies mit Bedingungen erreichen:

```
01 config.baseURL = http://www.domain.de/
02 [hostname = abc.domain.de]
03     config.baseURL = http://abc.domain.de/
04 [hostname = xyz.domain.de]
05     config.baseURL = http://xyz.domain.de/
06 [global]
```

XHTML-Konformität

Standardmäßig erzeugt TYPO3 keinen XHTML-konformen Quellcode. Es gibt aber mehrere Optionen, wie man die Ausgabe von TYPO3 XHTML-konform machen kann.

XHTML-Ausgabe aktivieren

Um überhaupt XHTML ausgeben zu können, müssen Sie den entsprechenden Doctype definieren:

```
config.doctype = xhtml_trans
```

Mit dieser Konfiguration würde ein `XHTML 1.0 Transitional` Doctype ausgegeben.

Die Einstellung hat folgende mögliche Werte:

- `xhtml_trans`
  für XHTML 1.0 Transitional
- `xhtml_frames`
  für XHTML 1.0 Frameset
- `xhtml_strict`
  für XHTML 1.0 Strict
- `xhtml_11`
  für XHTML 1.1
- `xhtml_2`
  für XHTML 2
- `none`
  für überhaupt keinen Doctype

Hinweis

Der TYPO3 Standard-Doctype ist HTML 4.0 Transitional.

Code säubern

Mit der folgenden Konfigurationseinstellung können Sie den von TYPO3 ausgegebenen HTML-Quelltext säubern:

```
config.xhtml_cleaning = all
```

Diese Einstellung ist noch nicht vollständig und konsistent implementiert. Momentan werden durch diese Einstellung nur folgende Dinge XHTML-konform gemacht:

- Alle inhaltsleeren Tags (`<img>`,`<br>`,`<hr>`) werden mit `/>` abgeschlossen.
- HTML-Elemente und -Attribute werden klein geschrieben.
- Alle Attribute werden in Anführungszeichen eingeschlossen.
- Allen `<img>`-Tags wird – sofern noch nicht vorhanden – das `alt`-Attribut hinzugefügt.

Andere Säuberungsoptionen werden im Laufe der Zeit noch implementiert. Nichtsdestotrotz können Sie die Option schon jetzt aktivieren.

Die Einstellung hat drei mögliche Werte:

- `all`

  Die Inhalte werden immer verarbeitet, bevor sie möglicherweise zwischengespeichert werden.

- `cached`

  Die Inhalte werden nur verarbeitet, wenn die Seite zwischengespeichert wird.

- `output`

  Die Inhalte werden nur kurz vor der Ausgabe verarbeitet.

Sonderzeichen bei textbasierten Menüs (TMENU-Objekten) umwandeln

Standardmäßig werden Sonderzeichen bei textbasierten Menüs nicht umgewandelt – haben Sie nun aber beispielsweise ein Kaufmanns-Und (&) im Titel einer Seite, so wird dieses direkt in den HTML-Quelltext geschrieben. Dies ist bei XHTML aber nicht zulässig – Sonderzeichen müssen escaped werden.

Dies können Sie mit der `stdWrap`-Eigenschaft `htmlSpecialChars` erreichen. Über die Aktivierung dieser Eigenschaft wird der Inhalt des jeweiligen Elements mit der gleichnamigen PHP-Funktion `htmlspecialchars()`[4] HTML-tauglich gemacht.

Beispiel:

```
01 temp.menu = HMENU
02 temp.menu.1 = TMENU
03 temp.menu.1 {
04    NO = 1
05    NO.stdWrap.htmlSpecialChars = 1
06 }
```

6.1.4 Seiten-Konfiguration (PAGE-Objekt)

Das PAGE-Objekt ist ein sehr grundlegendes Element von TypoScript.

Jede Seite, die im Browser dargestellt werden soll, benötigt ein Template record. Ob dieser Template record der Seite nun direkt zugeordnet ist oder von einer höheren Seite vererbt wird, spielt keine Rolle – wichtig ist, dass mindestens eines der (geerbten) Template records ein PAGE-Objekt enthält. Dieses PAGE-Objekt repräsentiert den Inhalt der entsprechenden Seite im Frontend.

Alle anderen Elemente, die auf der Seite angezeigt werden sollen, werden dem PAGE-Objekt als Kindelemente hinzugefügt.

4 *http://de.php.net/htmlspecialchars*

Hinweis

Warum wurde ein PAGE-Objekt eingeführt, wenn sowieso jede Seite mindestens ein PAGE-Objekt benötigt, hätte man ja auch standardmäßig ein PAGE-Objekt definieren können?!

Dafür gibt es mehrere Gründe: Zum einen gibt es Templates, die vielleicht nur einen Teil einer Konfiguration beinhalten und so kein PAGE-Objekt benötigen und zum anderen gibt es auch Seiten (z. B. SysFolder), die kein PAGE-Objekt benötigen, da sie nicht direkt im Frontend dargestellt werden müssen. Weiterhin ist es manchmal nötig, mehrere PAGE-Objekte zu definieren – beispielsweise bei Framesets.

Beispiel:

```
01 page = PAGE
02 page.10          = TEXT
03 page.10.value    = Hallo Welt!
```

Wird diese Seite aufgerufen (`index.php?id=[ID]`), so wird `Hallo Welt!` ausgegeben.

Sie sehen: Jede Seite wird durch ihre Identifikationsnummer (`id`) eindeutig identifiziert. Was aber, wenn man ein- und dieselbe Seite mehrmals mit nur kleinen Änderungen benötigt? Hier kommt die Typnummer (`type`) ins Spiel. Mit der Identifikationsnummer wird der Seitendatensatz eindeutig bestimmt und mit der Typnummer das PAGE-Objekt.

Eine Seite kann mehrere PAGE-Objekte definieren, hinter denen sich unterschiedlicher Inhalt verbirgt. Dies wird zum Beispiel bei Framesets oder Druckversionen einer Seite eingesetzt.

Beispiel:

```
01 page = PAGE
02 page.10          = TEXT
03 page.10.value    = Hallo Welt!
04
05 page2 < page
06 page2.typeNum    = 1
07 page2.10.value   = Hallo Du!
```

In diesem Beispiel enthält die Seite zwei PAGE-Objekte. Mit dem Aufruf von (`index.php?id=[ID]`) wird das erste PAGE-Objekt aktiviert und Sie erhalten die Ausgabe wie gehabt.

Mit dem Aufruf von (`index.php?id=[ID]&type=1`) wird das zweite PAGE-Objekt angesprochen. Sie bekommen beim zweiten Aufruf also die Ausgabe `Hallo Du!`.

Achtung

Die Typnummer eines PAGE-Objekts ist standardmäßig `0`, sobald Sie aber mehrere PAGE-Objekte definieren, müssen Sie zumindest in allen außer dem Standard-Element die Typnummer korrekt setzen.

Es ist nicht wichtig, welche Typnummern Sie für was verwenden und wie Ihr PAGE-Objekt heißt, aber es haben sich einige Quasi-Standards herausgebildet:

- Das PAGE-Objekt wird `page` genannt.
- Die Typnummer der Standardseite des Framesets ist `1`.

Wenn Sie dies befolgen oder zumindest im Hinterkopf behalten, wird es einfacher, fremde Konfigurationen zu lesen oder anderen Personen etwas zu erklären.

Über das PAGE-Objekt können seitenspezifische Änderungen gemacht werden. Ich werde hier nur einige anführen, eine vollständige Übersicht erhalten Sie in der TypoScript-Referenz.

bodyTag

Über diese Einstellung lässt sich das `<body>`-Tag des PAGE-Objekts beeinflussen. Standardmäßig lautet das Tag folgendermaßen: `<body bgcolor="#FFFFFF">`.

Die Anpassung des Tags kann hilfreich sein, wenn Sie

- zusätzliche Attribute hinzufügen möchten,
- die Seite `XHTML 1.0 Strict` konform halten möchten – das Attribut `bgcolor` ist dann nicht mehr erlaubt.

Beispiel:

```
page = PAGE
page.bodyTag = <body onload="meinScript();">
```

Hier hätten Sie den `<body>` Tag durch eine eigene Version ersetzt, die nach dem vollständigen Laden der Seite die Javascript-Funktion `meinScript()` ausführt.

shortcutIcon

Über diese Option können Sie für Ihre Seite ein Shortcut-Icon (auch: Favicon) definieren. Laden Sie dazu eine `.ico`-Datei in das `fileadmin/`-Verzeichnis.

Hinweis

Einen Freeware-Icon-Editor (LiquidIconXP) finden Sie bei X2 Studios[5].

Beispiel:

Sie können dann einem PAGE-Objekt mit folgender Deklaration das Icon zuweisen:

```
page = PAGE
page.shortcutIcon = fileadmin/pfad/zu/ihrem/icon.ico
```

stylesheet

Über diese Option können Sie in Ihre Seite Stil-Informationen über ein externes Stylesheet einfügen. Laden Sie dazu eine CSS-Datei in das `fileadmin/`-Verzeichnis.

Beispiel:

```
page = PAGE
page.stylesheet = fileadmin/pfad/zu/ihrer/stylesheet.css
```

Im Beispiel würde dann im `<head>`-Tag Ihrer Seite folgender Link eingefügt:

```
<link rel="stylesheet" href="fileadmin/pfad/zu/ihrer/stylesheet.css">
```

Hinweis

Sie können die CSS-Datei, anstatt sie hochzuladen auch zu Ihren Mediadaten hinzufügen und über

```
page = PAGE
page.stylesheet = stylesheet*.css
```

hinzufügen.

meta

Über diese Option können Sie der Seite Meta-Tags zuordnen. Dies ist sinnvoll, damit Ihre Seite von Suchmaschinen besser gefunden werden kann.

Mit folgender Deklaration können Sie die Beschreibung der Webseite festlegen:

```
# page.meta.[Name des Meta-Tags] = [Wert des Meta-Tags]
page.meta.description = Hallo, ich bin die Beschreibung.
```

5 *http://www.x2studios.com*

Diese Information würde als <meta>-Tag in den <head>-Bereich der Seite eingefügt:

```
<meta name="description" content="Hallo, ich bin die Beschreibung." />
```

Sie sehen, dass der Name des Meta-Tags frei wählbar ist – anstelle von `description` hätten Sie auch `refresh`, `keywords` oder andere wählen können.

Meta-Tags können Sie auch mit der Erweiterung `metatags` (siehe Kapitel 7.1.9, »Meta-Tags einfach verwalten (metatags)«, auf Seite 546) verwalten. Diese Erweiterung bietet ein einfaches Interface, mit dem auch Ihr Kunde Meta-Tags einpflegen kann.

6.1.5 Bilder (GIFBUILDER-Objekt)

Das GIFBUILDER-Objekt stellt ein Interface bereit, um Bilder wie Buttons oder Menüelemente zu erzeugen. Die beiden Menüformen GMENU und IMGMENU verwenden ebenfalls das GIFBUILDER-Objekt.

In diesem Kapitel wird nur die Basis eines GIFBUILDER-Objekts erklärt, eine vollständige Übersicht aller Teil-Objekte und deren Eigenschaften finden Sie in der TypoScript-Referenz[6].

Hinweis

Anders als der Name vermuten lässt, lassen sich mit diesem Objekt auch Bilder im JPEG- oder PNG-Format erzeugen. Dazu müssen Sie die `format`-Eigenschaft des GIFBUILDER-Objekts auf `jpeg` ändern.

```
01 # GIFBUILDER-Objekt erstellen
02 gfx = GIFBUILDER
03 gfx {
04     # Größe der Bilddatei festlegen
05     XY = [10.w]+4,[10.h]+2
06     # Offset festlegen
07     offset = 0,[10.h]-4
08     # Neues TEXT-Objekt erstellen
09     10 = TEXT
10     10 {
11           # Untertitel oder falls nicht vorhanden Seitentitel anzeigen
12           text.field = subtitle // title
13           # Schriftartendatei festlegen
14           fontFile = fileadmin/Bluehigh.ttf
15           # Schriftgröße festlegen
16           fontSize = 24
17           # Schriftfarbe festlegen
18           fontColor = gray
```

6 *http://typo3.org/documentation/document-library/doc_core_tsref/GIFBUILDER/*

```
19          # Antialiasing aktivieren, benötigt ImageMagick/GraphicsMagick
20          niceText = 1
21      }
22  }
23
24  # PAGE-Objekt erstellen
25  page = PAGE
26  # IMAGE-Objekt mit GIFBUILDER als Quelle erstellen
27  page.10 = IMAGE
28  # IMAGE-Objekt auf Seite einblenden
29  page.10.file < gfx
```

- Zuerst muss ein neues GIFBUILDER-Objekt erzeugt werden (Zeile 2).
  - In Zeile 5 wird die Bildgröße des GIFBUILDER-Objekts über die Eigenschaft `XY` festgelegt. Diese Eigenschaft erwartet einen Wert im Format `Breite,Höhe`.

    Hierbei können außer fixen Größenangaben in Pixel auch Eigenschaften von untergeordneten TEXT-Objekten und Berechnungen verwendet werden.

    So bezieht sich die Angabe `[10.w]` auf die Breite (width) des untergeordneten TEXT-Objekts `10` (Zeile 9) – respektive `[10.h]` auf die Höhe (height) des Objekts.

    Zur Breite werden noch vier Pixel und zur Höhe zwei Pixel addiert.
  - In Zeile 7 wird der Versatz aller Inhalte des GIFBUILDER-Objekts definiert. Dies wird über die Eigenschaft `offset` vorgenommen, die einen Wert im Format `X-Versatz,Y-Versatz` erwartet.

    Der horizontale Versatz ist hier Null, während der vertikale Versatz die Höhe des TEXT-Objekts `10` minus vier Pixel beträgt.

Hinweis

Die Angabe des Versatzes ist wichtig, da sonst Inhalte möglicherweise außerhalb des sichtbaren Bildbereiches gerendert werden. Wenn Sie einmal ein Bild angezeigt bekommen, das zwar die richtige Größe, aber keinen Inhalt hat, stimmt höchstwahrscheinlich der Versatz nicht.

  - Nun wird ein TEXT-Objekt benötigt, das den (textuellen) Inhalt des GIFBUILDER-Objekts definiert (Zeile 10).
    - In Zeile 12 wird der Inhalt des TEXT-Objekts festgelegt. In unserem Fall soll im Bild der Untertitel der Webseite und (falls nicht vorhanden) der Titel angezeigt werden.

Achtung

Das TEXT-Objekt an dieser Stelle ist kein »normales« TEXT-Objekt, sondern ein speziell auf die Verwendung innerhalb eines GIFBUILDER-Objekts angepasstes Objekt. Deshalb wird auf die `stdWrap`-Eigenschaften auch nicht direkt über das Objekt, sondern über die Eigenschaft `text` zugegriffen.

- In Zeile 14 wird die Schriftart des Textes festgelegt.

  Sie haben hier die Möglichkeit, sowohl TrueType- als auch Postscript-Schriftarten zu verwenden (standardmäßig versteht die FreeType2 eine ganze Menge Schriftartformate[7], allerdings muss die Unterstützung bei der Kompilierung aktiviert worden sein).

  Bei Postscript-Schriftarten müssen Sie als `fontFile` die .PFB-Datei angeben – die `.PFM`-Datei muss dabei aber im selben Verzeichnis liegen.

Achtung

Verwenden Sie Schriften, die urheberrechtlich geschützt sind, sollten Sie das Verzeichnis, in dem die Schriftart-Dateien liegen, unbedingt vor externen Zugriffen schützen, da Besucher sonst möglicherweise die Schriften herunterladen könnten. TYPO3 selbst muss nur über das Dateisystem auf die Schriften zugreifen. Denken Sie beim Ablegen von Schriften auf einem Server auch daran, dass die meisten Lizenzen für Schriftarten von der Anzahl der CPUs abhängig sind.

- In Zeile 16 wird die Größe der Schrift angegeben.
- In Zeile 18 wird die Schriftfarbe auf Grau gesetzt. Hier können entweder Hexadezimal-Farbwerte oder benannte Farben angegeben werden. Hier ist es wichtig, Hexadezimal-Farbwerte mit einer Raute (#) zu beginnen. Wird eine benannte Farbe nicht erkannt, so wird der Text schwarz gerendert.
- In Zeile 20 wird die NICETEXT-Option aktiviert. NICETEXT ist ein Konzept, das die Weichzeichnung der gerenderten Schrift verbessert.

7 *http://freetype.sourceforge.net/freetype2/index.html#features*

niceText

Dies geschieht, indem der Text in doppelter Größe auf eine Schwarz-Weiß-Maske gezeichnet wird. Diese Maske wird dann heruntergerechnet und die Schrift über die Maske auf das Bild gelegt.

Zusätzlich zum internen Rendern der Schrift mit FreeType werden beim Einsatz von `niceText` die ImageMagick-Funktionen `combine` (bzw. `composite`) und `convert` benötigt.

Aufgrund dessen entsteht beim Rendern ein erheblicher Mehraufwand, was sich vor allem auf schwachen Servern mit viel gerenderten Menüs bzw. Text beträchtlich auf die Performanz auswirken kann.

Abbildung 6.5: Beispieltext ohne Aktivierung von niceText – leicht pixelig

Abbildung 6.6: Beispieltext mit niceText aktiviert – schön geglättet

Für Interessierte das TypoScript zur Erzeugung der Beispiele in Abbildung 6.5 bzw. Abbildung 6.6:

```
01 temp.niceTest = IMAGE
02 temp.niceTest {
03    file = GIFBUILDER
04    file {
05       XY = [10.w]+10,[10.h]+10
06       10 = TEXT
07       10 {
08          fontSize = 72
09          fontFile = fileadmin/Bluehigh.ttf
10          offset = 0,50
11          text.data = page:title
12          niceText = 1 # Hier an-/ausschalten für Vergleich
13       }
14    }
15 }
```

- In Zeile 25 wird das PAGE-Objekt erstellt.
- In Zeile 26 wird ein neues IMAGE-Objekt innerhalb des PAGE-Objekts erzeugt.

  IMAGE-Objekte repräsentieren quasi das HTML `<img>`-Tag, während GIFBUILDER-Objekte die Bilddatei an sich repräsentieren.

  Deshalb wird, um ein GIFBUILDER-Objekt in einer HTML-Seite anzeigen zu können ein IMAGE-Objekt benötigt.
- Das IMAGE-Objekt benötigt einen korrekten Pfad zu einer Bilddatei, um das `<img>`-Tag mit korrektem `href`-Attribut bilden zu können. Deshalb wird dem erzeugten IMAGE-Objekt in Zeile 29 als `file`-Eigenschaft das GIFBUILDER-Objekt übergeben.

6.1.6 Menüs

Menüs sind in TYPO3 unglaublich leicht zu erstellen – die Anpassung dieser Menüs an bestimmte Vorgaben ist dabei schon komplizierter.

Es ist wichtig, dass Sie lernen, mit den xMENU-Objekten umzugehen, da Sie höchstwahrscheinlich selbst beim Einsatz von TemplaVoilà eigene Menüstrukturen definieren werden.

Hierarchische Menüs (HMENU-Objekt)

Der Grundstein eines jeden Menüs ist das HMENU-Objekt.

Aufgrund der Seitenstrukturen in TYPO3 sind hierarchische – die Baumstruktur widerspiegelnde – Menüs die am häufigsten eingesetzten. Der Grundstein dieser Menüs ist daher das HMENU-Objekt.

Über das HMENU-Objekt lassen sich die einzelnen Menüebenen konfigurieren und mit Menüformen (Text-, Grafik-, Bild- oder JavaScript-Menü) versehen. Auch wenn es theoretisch möglich ist, alle Menüformen in einem hierarchischen Menü zu vereinen, spielt dies in der Praxis kaum eine Rolle. Die meisten Menüs bestehen aus einer, höchstens zwei Menüformen innerhalb eines HMENU-Objekts.

Über das HMENU-Objekt lassen sich grundsätzliche Dinge des Menüs steuern – in welcher Ebene beginnt das Menü, gibt es eine minimale/maximale Anzahl Menüeinträge, werden bestimmten Seitentypen ausgeblendet, wird die Baumstruktur abgebildet oder nur spezielle Seiten etc.

Hinweis

Zur Verkürzung der Listings habe ich die Konfiguration eines Standard-Menüobjekts immer mit … angedeutet.

Wann immer Sie ein `temp.menu.1 = …` finden, können Sie anstelle der angedeuteten Zeile, folgenden Code anhängen, um ein (zugegebenermaßen minimalistisches, aber) funktionierendes Menü zu erhalten:

```
01 temp.menu.1 = TMENU
02 temp.menu.1 {
03     NO = 1
04     NO.after = <br />
05 }
```

Menüobjekte definieren

```
01 temp.menu = HMENU
02 temp.menu.1 = TMENU
03 temp.menu.2 = GMENU
04 temp.menu.3 = JSMENU
05 temp.menu.4 = …
```

Das HMENU-Objekt kann beliebig viele untergeordnete Menüobjekte enthalten. Dabei stellen die Zahlen die jeweilige Menüebene dar. So wird die erste Ebene im Beispiel als textbasiertes Menü dargestellt (Zeile 2), die zweite Ebene als grafikbasiertes Menü (Zeile 3), die dritte Ebene als JavaScript-basiertes Menü (Zeile 4) etc.

Achtung

Jedes HMENU-Objekt muss mindestens ein untergeordnetes Menüobjekt (für Ebene eins) enthalten. Außerdem ist es nicht möglich, Ebenen komplett auszulassen – möchten Sie eine Ebene nicht anzeigen, so müssen Sie das Menüelement für diese Ebene dennoch anlegen und dann die Ausgabe bzw. die Sichtbarkeit beeinflussen.

Startebene festlegen

```
01 temp.menu = HMENU
02 temp.menu.entryLevel = 1
03 temp.menu.1 = …
```

Mit der Eigenschaft `entryLevel` (Zeile 2) können Sie festlegen, ab welcher Ebene (also welcher Seite in Ihrer Baumstruktur) das Menü beginnt. Im Beispiel wird die erste Ebene (Ebene 0) ausgelassen und erst bei Ebene 1 begonnen.

Die Zählung der Ebenen beginnt immer mit Null auf der Ebene, wo zuletzt im Template die `rootlevel`-Eigenschaft (siehe Abschnitt »Übersicht der Optionen eines Template records«, auf Seite 390, Abbildung 4.176, Punkt ⑤) gesetzt wurde.

Negative Zahlen sind ebenfalls erlaubt, dann beginnt die Zählung außen (-1 wäre also die aktuelle Ebene, -2 eine Ebene höher etc.).

Beispiel:

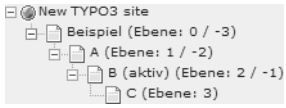

Abbildung 6.7: Seitenstruktur – wir befinden uns momentan auf Seite B

Angenommen, es existiert eine Seitenstruktur wie in Abbildung 6.7 gezeigt. Die Seite `Beispiel` hat die `Rootlevel`-Eigenschaft gesetzt. Wir befinden uns im Frontend auf Seite `B`.

Die positiven Zahlen sind verständlich: Die Seite mit der gesetzten `Rootlevel`-Eigenschaft ist Ebene `0`, die darunter liegende Ebene `1`, die darunter Ebene `2` etc.

Interessanter sind die negativen Angaben. Diese gelten immer ab der aktuellen (aktiven) Seite (also in unserem Fall Seite `B`). Die aktuelle Ebene ist dabei Ebene `-1`, die nächst höhere Ebene `-2` etc.

Wichtig ist hierbei, dass negative Angaben relativ sind, d.h. sobald Sie z.B. auf Seite `C` wechseln, wäre Seite `C` Ebene `-1` und Seite `B` Ebene `-2` etc.

Tipp

Wie viele Ebenen nachher im Menü eingeblendet werden, hängt natürlich nicht nur davon ab auf welcher Ebene Sie beginnen, sondern auch wie viele Ebenen (Menüobjekte) definiert sind (siehe Abschnitt »Menüobjekte definieren«, auf Seite 511).

Spezielle Menüs

Das HMENU-Objekt stellt standardmäßig Menüs für die im Backend festgelegte Baumstruktur bereit, allerdings kann es manchmal nötig sein, Seiten nicht in der angelegten Baumstruktur darzustellen, sondern Seiten mit bestimmten Eigenschaften etc. in einem Menü bereitzustellen. Dazu gibt es die `special`-Eigenschaft.

Diese Eigenschaft kann verschiedene Werte annehmen:

- `directory`

  Über diese Option können Sie alle Seiten in einem Menü darstellen, die eine bestimmte Elternseite haben. Dies erlaubt es Ihnen beispielsweise, an unterschiedlichen Stellen der Webseite ein kleines Menü mit Seiten eines bestimmten Themas einzublenden.

Beispiel:

```
01 temp.menu = HMENU
02 temp.menu.special = directory
03 # IDs der Seiten, deren Kindelemente angezeigt werden sollen
04 temp.menu.special.value = 11,22
05 temp.menu.1 = …
```

- `list`

  Über diese Option können Sie bestimmte Seiten, die ansonsten keinen Zusammenhang (gleiche Elternseite, o. Ä.) haben, in einem Menü darstellen.

Beispiel:

```
01 temp.menu = HMENU
02 temp.menu.special = list
03 # IDs der Seiten, die im Menü angezeigt werden sollen
04 temp.menu.special.value = 11,22,33
05 temp.menu.1 = …
```

- `updated`

  Über diese Option können Sie zuletzt bearbeitete/veränderte Seiten in einem Menü anzeigen.
  Über weitere Eigenschaften können Sie die Anzahl der Seiten, die Sortierung, den Zeitrahmen und ähnliche Eigenschaften zur Einschränkung der Treffermenge beeinflussen.

Tipp

Achten Sie bei der Verwendung dieses speziellen Menüs auf einen für Ihre Seite sinnvollen Zeitraum und eine sinnvolle (Ebenen-)Tiefe. Sie können diese Eigenschaften mittels Extension Templates durch Anpassung des Menüobjekts auch für einzelne Teilbäume festlegen.

Beispiel:

```
01 temp.menu = HMENU
02 temp.menu.special = updated
03 temp.menu.1 = …
```

- `rootline`

  Über diese Option können Sie alle Seiten von der Wurzel (Template mit gesetztem `rootlevel`-Flag) bis zur aktuellen Seite in einem Menü anzeigen. Über weitere Eigenschaften können Sie u.a. die Start- und Ziel-Ebene genauer definieren.

Beispiel der Rootline:

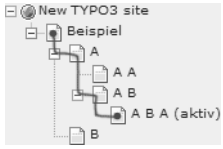

Abbildung 6.8: Rootline zwischen Beispiel und A B A

In der Seite `Beispiel` ist die rootlevel-Eigenschaft gesetzt. Wir befinden uns im Frontend auf Seite `A B A`. Die Rootline umfasst daher die Seiten `A B A` → `A B` → `A` → `Beispiel` (in Abbildung 6.8 eingezeichnet).
Code-Beispiel:

```
01 temp.menu = HMENU
02 temp.menu.special = rootline
03 temp.menu.1 = …
```

- `browse`

  Über diese Option können Sie ein Menü erzeugen, das eine Dateibrowser-ähnliche Navigation über den Seitenbaum erlaubt.
  Es stehen unterschiedliche Elemente (vorherige Seite, folgende Seite, eine Ebene höher etc.) zur Verfügung, die im Menü eingeblendet werden können. Weiterhin lassen sich die Sortierung und andere Eigenschaften des Menüs über zusätzliche Schalter beeinflussen.

Beispiel:

```
01 temp.menu = HMENU
02 temp.menu.special = browse
03 # Elemente, die im Menü zur Verfügung stehen sollen
04 temp.menu.special.items = prev|next
05 temp.menu.1 = …
```

Standardmäßig werden als Link die Titel der verlinkten Seiten verwendet. Möchten Sie die Links nach Ihrer Funktion benennen, können Sie folgende Deklarationen verwenden:

```
01 temp.menu.special {
02  prev.fields.title = vorherige Seite
03  next.fields.title = nächste Seite
04 }
```

Dies funktioniert auch mit allen anderen reservierten Elementnamen.

- `keywords`

  Über diese Option können Sie ein Menü erzeugen, das Seiten mit gleichen Schlüsselwörtern enthält. Dies ist sehr praktisch, um Verknüpfungen von Seiten eines ähnlichen Themengebietes zu schaffen, die nicht im selben Teilbaum liegen.

  Außer der Verwendung von Schlüsselwörtern der aktuellen Seite steht Ihnen auch die Möglichkeit offen, Schlüsselwörter explizit zu definieren oder Verknüpfungen über andere Felder (mittels Anpassung der Eigenschaft `.keywordsField` bzw. `.keywordsField.sourceField`) zu erzeugen.

Beispiel:

```
01 temp.menu = HMENU
02 temp.menu.special = keywords
03 temp.menu.1 = …
```

- `language`

  Über diese Option können Sie ein Menü erzeugen, das die Auswahl der Sprache bzw. lokalisierter Seiten in anderen Sprachen erlaubt.

  Ein ausführliches Beispiel zu dieser Option finden Sie im Abschnitt »Sprachauswahl«, auf Seite 631.

Beispiel:

```
01 temp.menu = HMENU
02 temp.menu.special = language
03 # IDs der Sprachen, deren lokalisierte Seiten verlinkt werden sollen
04 temp.menu.special.value = 0,1,2
05 temp.menu.1 = …
```

- `userdefined`

  Diese Option macht es möglich, Menüelemente durch ein eigenes PHP-Skript bereitzustellen. Dies ist äußerst praktisch, wenn Sie externe Menüs in die eigene Seite einbinden und diese dennoch über TypoScript konfigurierbar haben möchten.

  Einige spezielle Schlüsselwörter lassen die Konfiguration von Zuständen und Link-Zielen bereits im Skript zu.

Beispiel:

```
01 temp.menu = HMENU
02 temp.menu.special = userdefined
03 temp.menu.special.file = fileadmin/my_menu.inc.php
04 temp.menu.1 = …
```

Inhalt von `fileadmin/my_menu.inc.php`:

```
01 <?php
02 $menuItemsArray[] = array('title' => 'Alles',   'uid' => 1);
03 $menuItemsArray[] = array('title' => 'Lüge',   'uid' => 3);
04 ?>
```

Achtung

Jeder Array-Eintrag im Array `$menuItemsArray` (dieses muss so benannt werden) stellt eine Tabellenzeile der `pages`-Tabelle dar. Das Minimum an Information, um ein Menü zu erstellen, ist der Titel des Elements (`title`) und die Seiten-ID (`uid`), zu der verlinkt werden soll.

Beispiel (Fortsetzung):

Möchten Sie mehrere Ebenen erzeugen, so können Sie dies durch den Eintrag `_SUB_MENU` erreichen:

```
01 <?php
02 $menuItemsArray[] = array(
03                     'title' => 'Alles',
04                     'uid'   => 1);
05 $menuItemsArray[] = array(
06                     'title' => 'nur',
07                     'uid'   => 2,
08                     '_SUB_MENU' => array(
09                                        array(
10                                               'title' => 'Lüge',
11                                               'uid'   => 3)));
12 ?>
```

Achtung

Sie müssen beim Erzeugen mehrerer Ebenen im HMENU-Element selbstverständlich auch für jede Ebene ein Menüobjekt definieren.

- `userfunction`

Mit dieser Option ist es möglich, das Menü mit dem Rückgabewert einer Funktion zu füllen. Die Funktion muss ein Array zurückgeben, das dem Aufbau der definierten Menüstruktur bei der Verwendung der `userdefined`-Option entspricht.

Beispiel:

```
01 # Datei mit Klasse/Funktion laden
02 includeLibs.user_myMenu = fileadmin/user_myMenu.inc.php
03
04 temp.menu = HMENU
05 temp.menu.special = userfunction
06 temp.menu.special.userFunc = user_myMenu
07 temp.menu.1 = …
```

Inhalt von `fileadmin/user_myMenu.inc.php`:

```
01 <?php
02 function user_myMenu($content,$conf) {
03  return array( array(
04                        'title' => 'Alles',
05                        'uid'   => 1),
06                        array(
07                               'title' => 'Lüge',
08                               'uid'   => 2));
09 }
10 ?>
```

Achtung

Benutzerdefinierte Funktionen müssen immer mit `user_` beginnen, ansonsten kann es evtl. zu Namensraum-Überschneidungen mit Erweiterungen etc. kommen.

Für jede dieser Werte gibt es wiederum spezielle Optionen, die die Ausgabe beeinflussen. Eine genauere Dokumentation finden Sie in der TSRef[8].

Menüzustände (TMENU, GMENU und IMGMENU)

Für jedes Menüobjekt (nur TMENU, GMENU und IMGMENU) gibt es mehrere Zustände.

Das heißt abstrahiert sieht die Konfiguration eines Menüs immer so aus:

```
Menü (HMENU-Objekt)
       |-> Menüobjekt (xMENU-Objekt)
             |-> Normaler  Zustand (NO)
             |-> Aktiver   Zustand (ACT)
             |-> …         Zustand (…)
```

Welche Zustände es für welche Menüobjekte gibt, hängt sowohl von den Eigenschaften des HMENU-Objekts als auch von den Menüobjekten selbst ab.

Verschiedene Zustände (bei Zuständen mit dem Suffix RO ist es der Zustand mit den gleichen Voraussetzungen, jedoch nur aktiv, wenn der Besucher zusätzlich mit seiner Maus über das Menüelement fährt):

- `NO`

  Dieser Zustand ist der Normalzustand eines Menüelements und standardmäßig aktiviert.

Hinweis

Wenn Sie diesen Menüzustand später kopieren möchten – beispielsweise für andere Zustände, so ist es dennoch sinnvoll, ihn mit `NO = 1` zu initialisieren, da es hin und wieder Probleme beim Kopieren des Zustandes gibt, sofern keine explizite Initialisierung stattfand.

- `RO`

  Dieser Zustand ist aktiv, wenn ein Benutzer mit der Maus über ein Menüelement fährt. Dieser Zustand steht beispielsweise bei einem GMENU-Objekt zur Verfügung.

- `IFSUB/IFSUBRO`

  Dieser Zustand wird aktiviert, wenn ein Menüelement Kindelemente hat.

- `ACT/ACTRO`

  Dieser Zustand wird aktiviert, wenn ein Menüelement in der `rootline` liegt, d.h. auf dem »Weg« von der Seitenwurzel bis zur aktuellen Seite.

8 *http://typo3.org/documentation/document-library/references/doc_core_tsref/current/view/8/11/*

- `ACTIFSUB/ACTIFSUBRO`

  Dieser Zustand wird aktiviert, wenn ein Menüelement in der `rootline` liegt und Kindelemente hat.

- `CUR/CURRO`

  Dieser Zustand wird aktiviert, wenn das Menüelement das letzte in der `rootline` ist, d.h. der Benutzer sich momentan auf der dem Menüelement entsprechenden Seite befindet.

- `CURIFSUB/CURIFSUBRO`

  Dieser Zustand wird aktiviert, wenn das Menüelement das letzte in der `rootline` ist und zusätzlich noch Kindelemente hat.

- `USR/USRRO`

  Dieser Zustand wird aktiviert, wenn das Menüelement Seiten repräsentiert, die zugangsgeschützt sind und zu denen der aktuelle Besucher Zugang hat.

- `SPC`

  Dieser Zustand wird aktiviert, wenn das Menüelement eine Seite vom Typ `Spacer` (Trennelement) repräsentiert.

Hinweis

Verwenden Sie das GMENU-Menüobjekt, funktioniert dieser Zustand nicht, wenn Sie Rollovers (RO) einsetzen.

- `USERDEF1/USERDEF1RO`

  Erster benutzerdefinierter Zustand. Dieser Zustand wird abhängig von einem Skript beim Eintreten bestimmter Voraussetzungen aktiviert.

  Das spezielle HMENU-Objekt für ein Sprachenmenü (`.special = language`) setzt beispielsweise diesen Zustand, wenn eine Seite in einer Sprache nicht lokalisiert wurde. So lassen sich abhängig von den Gegebenheiten weitere Zustände aktivieren.

- `USERDEF2/USERDEF2RO`

  Zweiter benutzerdefinierter Zustand. Beschreibung siehe USERDEF1.

Text-basiertes Menü (TMENU-Objekt)

Textbasierte Menüs sind die wohl am häufigsten genutzten – dies hat einerseits mit dem unaufhaltsamen Vormarsch von CSS und andererseits mit der höheren Besucherfreundlichkeit (Accessibility) zu tun.

Ich rate Ihnen, wann immer es Ihnen möglich ist, auf textbasierte Menüs zurückzugreifen und diese mit Stil-Informationen in die benötigte Form zu bringen.

<ul>-Menü

Das folgende Menü erzeugt eine (unsortierte) Liste mit derselben Baumstruktur wie im Backend. Über CSS können Sie das Aussehen dieser Liste einfach anpassen.

```
01 # HMENU-Objekt erstellen
02 temp.menu = HMENU
03 temp.menu {
04     # TMENU-Objekt erstellen
05     1 = TMENU
06     1 {
07           # Jedes TMENU mit <ul> umschließen
08           wrap = <ul>|</ul>
09           # Normalen Zustand definieren
10           NO = 1
11           NO {
12                 # jedes Element und Sub-Elemente mit <li> umschließen
13                 wrapItemAndSub = <li>|</li>
14                 # Das title-Attribut der Links einstellen
15                 ATagTitle.field = subtitle // title
16           }
17           # Aktiven Zustand definieren
18           ACT = 1
19           # Aktiven von normalem Zustand kopieren
20           ACT < .NO
21           # Aktive Elemente kennzeichnen
22           ACT.ATagParams = style="color: red;"
23     }
24     # Konfiguration von erster Ebene für zweite Ebene kopieren
25     2 < .1
26     # Konfiguration von erster Ebene für dritte Ebene kopieren
27     3 < .1
28     # Hier für beliebig viele weitere Ebenen kopieren (z.B. 4 < .1)
29 }
```

- Zuerst muss ein neues HMENU-Objekt erzeugt werden (Zeile 2).
  - Dieses benötigt mindestens ein Menüobjekt – in diesem Fall ein TMENU-Objekt (Zeile 5). Da die gesamte Navigation gleichförmig und damit rekursiv definierbar ist, werden wir dieses Menüobjekt nur einmal definieren und danach kopieren.
    - Jedes Menüobjekt soll von `<ul>` umschlossen werden. Dies geschieht in Zeile 8 und nur, wenn das Menü Elemente enthält. So kann sichergestellt werden, dass kein leeres `<ul>`-Tag produziert wird, sollte das Menü keine Elemente enthalten. Diese Behandlung ist auch wichtig für die weiteren Ebenen, die ansonsten fehlerhaft eingeblendet werden.

- Nun muss der normale Zustand eines jeden Menüelements definiert werden. Wir möchten, dass jedes Element von `<li>` umschlossen wird (Zeile 13). Außerdem sollen die Links `title`-Attribute bekommen, die den Untertitel oder falls nicht vorhanden den Titel der Seite enthalten (Zeile 15).
- Da wir dem Besucher zeigen möchten, auf welcher Seite und in welchem Untermenü er sich gerade befindet, müssen wir den aktiven Menüzustand jedes Elements ebenfalls definieren (Zeile 18).
- Wir kopieren die Einstellungen des normalen Zustandes für den aktiven Zustand (Zeile 20), um uns die zusätzlichen Deklarationen zu ersparen.
- Dann fügen wir jedem Link ein `style`-Attribut hinzu (Zeile 22), mit dem wir den Link rot färben.

  Sie können hier selbstverständlich jedes zulässige Attribut hinzufügen – es bietet sich ein `class`-Attribut an, dann können Sie das Menü wiederverwenden und müssen nur das CSS anpassen.

– In Zeile 25 kopieren wir das erste TMENU-Objekt, um die zweite Ebene unseres Menüs zu definieren.
– In Zeile 27 kopieren wir das erste TMENU-Objekt, um die dritte Ebene unseres Menüs zu definieren. Bei diesem Menü können Sie den Kopiervorgang beliebig (für ein beliebig tiefes Menü) wiederholen.

expAll

Diese Eigenschaft des TMENU-Objekts wird häufig eingesetzt/benötigt, deshalb möchte ich diese hier gesondert beschreiben.

Sie müssten diese Eigenschaft im Beispiel dieses Kapitels zwischen Zeile 6 und 23 (am besten in Zeile 7 oder 9) einfügen:

```
expAll = 1
```

Standardeinstellung bei Menüs ist, dass nur die aktuelle Ebene (also die, auf der man sich gerade befindet) ausgeklappt wird.

Die nächste Unterebene wird erst dann ausgeklappt, wenn der Menüpunkt, der die Unterebene enthält, aktiviert wird. Um Menüs zu erzeugen, bei denen standardmäßig alle Untermenüs ausgeklappt sind, wird die expAll-Eigenschaft benötigt.

Beispiel:

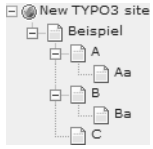

Abbildung 6.9: Seitenstruktur mit Unterseiten.

Sie haben folgenden Menücode eingebunden:

```
01 temp.myNavigation = HMENU
02 temp.myNavigation {
```

```
03      1 = TMENU
04      1 {
05            NO = 1
06            NO.before = |->
07            NO.after  = <br />
08            expAll    = 1
09      }
10      2 < .1
11      2.NO.before = |     |->
12  }
```

Wir befinden uns im Frontend auf der Seite `Beispiel`. Das erzeugte Menü sieht folgendermaßen aus:

```
|->A
|  |->Aa
|->B
|  |->Ba
|->C
```

Abbildung 6.10: Unterseiten von Seite A und Seite B werden eingeblendet

Sie sehen, dass nicht nur die Unterseiten der aktuellen Seite im Menü eingeblendet werden, sondern alle. Dies wurde durch die Eigenschaft expAll (in Zeile 8) definiert. Würden wir Zeile acht entfernen, so werden Unterseiten nicht immer angezeigt, sondern erst wenn man die entsprechende Elterseite öffnet. Siehe Abbildung 6.11.

```
|->A
|->B
|->C
```

Abbildung 6.11: Menü mit ausgeblendeten Unterseiten

Grafik-basiertes Menü (GMENU-Objekt)

Über das GMENU-Objekt lassen sich Menüs erzeugen, deren Menüelemente einzelne GIF-Dateien sind.

```
01 # HMENU-Objekt erzeugen
02 temp.menu = HMENU
03 # GMENU-Menüobjekt hinzufügen
04 temp.menu.1 = GMENU
05 temp.menu.1 {
06     # Normalen Zustand definieren
07     NO = 1
08     NO {
09           # Größe des GIF-Bildes einstellen
10           XY = [10.w]+2,[10.h]+4
11           # Versatz für Inhalt festlegen
12           offset = 0,10
```

```
13          # TEXT-Objekt innerhalb des GIFBUILDER-Objekts definieren
14          10 = TEXT
15          10 {
16                # Textinhalt festlegen
17                text.field = subtitle // title
18                # Inhalte zentrieren
19                align = center
20          }
21    }
22    # Rollover Zustand definieren
23    RO = 1
24    # Einstellungen vom Normalzustand kopieren
25    RO < .NO
26    RO {
27          # Hintergrund bei Mouseover rot einfärben
28           backColor = red
29    }
30 }
```

- Zuerst muss ein HMENU-Objekt erzeugt werden (Zeile 2).
  - Dann muss ein Menüobjekt hinzugefügt werden (Zeile 4).
    - Für das GMENU-Objekt wird der normale Zustand definiert (Zeile 7). Jeder Zustand eines GMENU-Objekts repräsentiert ein GIFBUILDER-Objekt.
    - In Zeile 10 wird die Größe des GIF-Bildes (des GIFBUILDER-Objekts) eingestellt.
    - In Zeile 12 wird der Versatz festgelegt, um den (Text-)Inhalt des GIFBUILDER-Objekts richtig auszurichten.
    - Damit der Menütext eingeblendet wird, muss ein neues TEXT-Objekt innerhalb des GIFBUILDER-Objekts angelegt werden (Zeile 14).
    - Der Textinhalt soll dem Untertitel der Seite, oder falls nicht vorhanden, dem Seitentitel entsprechen (Zeile 17). Achten Sie an dieser Stelle auf die verwendete Eigenschaft `text`: Diese existiert nur bei TEXT-Objekten innerhalb von GIFBUILDER-Objekten.
    - Damit der Textinhalt gleichmäßig ausgerichtet wird, muss er zentriert werden (Zeile 19).
    - Wir möchten, dass das Menüelement eine andere Hintergrundfarbe bekommt, wenn man mit der Maus darüber fährt. Deshalb muss der Zustand RO (RollOver) gesetzt werden (Zeile 23).
    - Der Inhalt des GIF-Bildes soll ansonsten aber gleich bleiben, deshalb wird der Normalzustand kopiert (Zeile 25).
    - In Zeile 28 wird im RO-Zustand der standardmäßig weiße Hintergrund durch einen roten ersetzt.