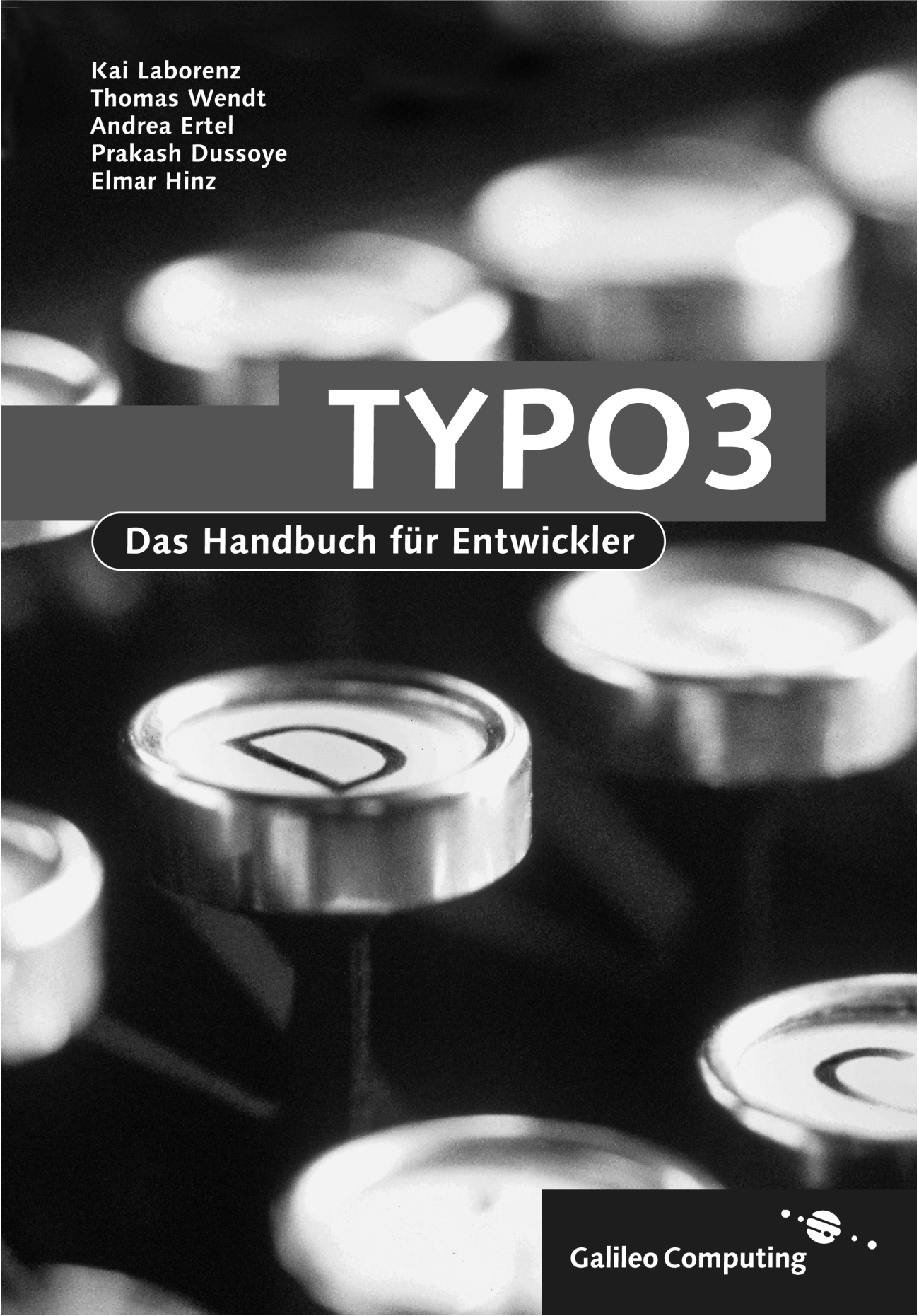
Kai Laborenz
Thomas Wendt
Andrea Ertel
Prakash Dussoye
Elmar Hinz

TYPO3

Das Handbuch für Entwickler



Galileo Computing

Auf einen Blick

Inhalt

2 Einführung

TYPO3 hat sich in den Jahren seiner Existenz stetig weiterentwickelt und verbessert. Einige der wichtigsten Änderungen sind in den letzten drei Versionssprüngen (ab Version 3.6) entstanden, und für die nahe Zukunft sind neue Meilensteine angekündigt.

2.1 Neu in Version 3.8

Der Umstieg von TYPO3 3.7 auf 3.8 hat eine Reihe von Veränderungen und Verbesserungen gebracht:

- Statt ImageMagick kann nun GraphicsMagick (**http://www.graphicsmagick.org/ (Linkcode 002)**) zur Bildmanipulation verwendet werden – es soll schneller sein, weniger Ressourcen verbrauchen und bessere Bildqualität liefern (optional im Install-Tool wählbar).
- Spracheinstellungen als Extensions
- Verwendung von Cache-Control-Headern zur Performance-Steigerung (siehe Abschnitt 4.3.1, *Performance-Steigerungen*)
- Neuer Ordner Systemextensions (**typo3/sysext/**) für grundlegende Extensions
- Verbesserungen in der Backend-Bedienung (u.a. funktioniert das Kontextmenü auch in Opera, Backend-Module können zur besseren Übersicht eingeklappt werden)
- Verbesserungen im Frontend (Frontend-Rendering kann nun auf Teile des Seitenbaums beschränkt werden, Frontend-Loginfelder sind »case-sensitiv«, und Cookies für Frontend-Benutzer können mit einer Lebensdauer versehen werden; die Spam-Verschlüsselung für E-Mails funktioniert jetzt ohne Javascript.)
- Verbesserungen der Extension »Indexed Search« (Indizierung von Excel, Powerpoint und OpenOffice, Statistiken über die Suchwörter; leider noch keine Templates oder tabellenfreier Code für die Ausgabe)
- Umfangreiche Verbesserungen der »Import/Export«-Extension
- Auch an der Datenbankabstraktion (Verwendung anderer Datenbanken als MySQL) wurde gearbeitet (siehe dazu Abschnitt 13.3.2, *Datenbankzugriffe mit dem DBAL*).

Allerdings sind einige sehnlichst erwartete Verbesserungen noch nicht offiziell dabei:

- TemplaVoila
- Digital Asset Management (DAM)
- Workflow-Management

Diese und andere Funktionsblöcke sollen jedoch mit der kommenden Version TYPO3 4.0 integriert werden – lesen Sie dazu das nächste Kapitel.

2.2 Pläne für die Zukunft

Die TYPO3-Community hat einige Pläne für die Zukunft – laut der aktuellen Roadmap können wir folgende zukünftige Entwicklungen erwarten:

2.2.1 TYPO3 4.0 (in der Entwicklung, Veröffentlichung geplant für Ende 2005)

- PHP 4- und PHP 5-Kompatibilität
- Workflow-Management
- Versionsverwaltung
- Workspaces 1.0
- Digital Asset Management 1.0
- Weiterentwicklung des Extension Managers
- Neuer Standard-RTE
- TemplaVoila 1.0
- Datenbankabstraktionsmodell (DBAL 1.0)
- Indexed Search 2.0 (mit Templates)
- Methoden zur Prüfung und Verbesserung der Datenintegrität
- Verbesserung der Bedienungsfreundlichkeit

2.2.2 TYPO3 4.5 (Veröffentlichung geplant für Frühjahr 2006)

- Redesign des Backends
- Weitere Verbesserung der Bedienungsfreundlichkeit

2.2.3 TYPO3 5.0 (Veröffentlichung geplant für Herbst 2006)

- PHP5-Optimierungen
- Verbesserungen der Systemarchitektur

- Portalfunktionen
- Änderungen im TYPO3-Kern zur Verbesserung der Bedienungsfreundlichkeit
- Integration von SOAP
- WebDAV

12 Barrierefreie und standardkonforme Websites mit TYPO3

In den letzten Jahren hat die Bedeutung standardkonformer und barrierefreier Websites enorm zugenommen. Der Grund dafür ist in Deutschland unter anderem die »Verordnung zur Schaffung barrierefreier Informationstechnik nach dem Behindertengleichstellungsgesetz« (BITV), die barrierefreie Webauftritte für Bundesbehörden und deren Körperschaften bis spätestens Ende 2005 vorschreibt.

12.1 Warum Sie barrierefrei entwickeln sollten

Aber auch das gestiegene Bewusstsein, dass das WWW in erster Linie ein Informationsmedium ist, und die zunehmende Anzahl von mobilen Internet-Clients lässt immer mehr Kunden nach Barrierefreiheit verlangen.

Wir verwenden hier der Verständlichkeit halber den eingeführten Begriff »Barrierefreiheit«, obwohl genau genommen eher von »Barrierearmut« gesprochen werden muss. Aufgrund der vielen möglichen unterschiedlichen Einschränkungen ist es kaum möglich, eine Website so zu erstellen, dass sie wirklich keine Barrieren für niemanden mehr enthält.

Die Vorteile einer barrierefreien, standardkompatiblen Website sind:

- zugänglich für alle Menschen
- gut indizierbar für Suchmaschinen
- leicht zu warten
- geringe Dateigrößen – schnell zu laden – geringe Traffic-Kosten
- auch auf mobilen Geräten darstellbar
- zukunftssicher

12.2 Wie sieht eine barrierefreie Website aus?

Was ist nun notwendig, um eine barrierefreie Website zu erreichen? Sehen wir uns die Anforderungen zunächst allgemeiner an, um dann auf die konkreten technischen Kriterien zu kommen.

12.2.1 Inhalt und Präsentation trennen

Der HTML-Code enthält die Dokumentenstruktur – das Stylesheet die Layout- und Designinformationen. Nur so können Geräte, die mit der optischen Darstellung nichts anfangen können, die Struktur zur Präsentation verwenden.

12.2.2 Logische Strukturen schaffen

Verwenden Sie die HTML-Tags in ihrer Funktion als semantische Gliederungsbefehle: `<h1>` markiert die wichtigste Überschriftenebene eines Dokuments, und `<table>` leitet eine tabellarische Darstellung von Daten ein – nicht ein mehrspaltiges Dokument.

Ein nicht unerheblicher Aufwand entsteht durch die Forderung, Abkürzungen und Sprachwechsel im Dokument zu markieren.

Um so schlimmer ist es, dass der Internet Explorer gerade mit dem Element `<abbr title="..."> ... </abbr>` (für Abkürzungen) nichts anfangen kann. Weder zeigt er den Titel (also die ausgeschriebene Form) an noch wendet er per CSS zugewiesene Stile an. Eine Lösung hat der tschechische Entwickler Marek Prokop auf seiner Website **http://www.sovavsiti.cz/css/abbr.html (Linkcode 045)** entwickelt (keine Angst, der Artikel ist in englischer Sprache verfasst). In der nächsten Version von XHTML wird es im Übrigen auch nur noch das Element `<abbr>` geben – hoffen wir, dass auch der Interent Explorer es bis dahin gelernt hat.

12.2.3 Nicht für bestimmte Geräte arbeiten

Denken Sie bei der Planung der Seiten nicht nur an grafische Browser – und vor allem nicht an einen bestimmten Browser, ein bestimmtes Betriebssystem oder eine bestimmte Bildschirmauflösung. Sie können letztendlich nicht wissen, wer wo mit welchem Gerät Ihre Website besucht. Verzichten Sie auf proprietäre Befehle, auch wenn sie noch so verführerisch erscheinen.

Verwenden Sie möglichst relative Größenangaben statt absoluter, damit sehschwache Nutzer die Schrift vergrößern können.

12.2.4 Abwärtskompatible Seiten erstellen

Wenn Sie dennoch besondere Features nutzen (müssen), die nur auf neuen oder bestimmten Geräten funktionieren, sorgen Sie dafür, dass die Seiten auch ohne diese Features funktionieren.

Wenn Sie eine Navigationsleiste mit einem Java-Applet oder durch eine Flash-Animation realisieren, zeigt ein Browser ohne Java oder Flash-Plugin *gar nichts* an. Würde die Navigationsleiste aus Grafiken bestehen, die mit einem Mouse-Over-Effekt versehen sind, so könnten selbst Personen, die ihre Grafiken abgeschaltet haben, noch die alternativen Texte sehen (natürlich nur, sofern Sie diese angegeben haben).

12.2.5 Alternativen bereitstellen

Wenn Sie Inhalte oder Funktionen verwenden, die in Hinsicht auf die Zugänglichkeit problematisch sind, stellen Sie Alternativen bereit.

Grafiken können abgeschaltet werden bzw. sind Blinden nicht zugänglich: Verwenden Sie immer das `alt`-Attribut des `<img>`-Tags.

Bieten Sie zu einer auf JavaScript oder Flash basierenden Navigation eine alternative Text-Navigation an. Stellen Sie für Videos, Animationen oder gesprochene Texte eine Beschreibung bzw. ein Transskript zur Verfügung. Ergänzen Sie mausspezifische Ereignisse um entsprechende Tatstatur-Ereignisse: Zu jedem `:hover` gehört ein `:focus`.

12.2.6 Standards beachten

Beachten Sie die Standards des W3C, und erstellen Sie Webseiten in HTML 4 oder XHTML 1. Testen Sie Ihre Dokumente mit den vom W3C bereitgestellten Validatoren.

Detailliertere Informationen, was zum Erreichen einer barrierefreien Website erforderlich ist, finden Sie in der BITV, den Kriterien des BIK-Kurztests oder im Kriterienkatalog des BIENE-Wettbewerbs der Aktion Mensch (**http://www.biene-award.de/ (Linkcode 046)**).

Sie finden diese Unterlagen auf der dem Buch beiliegenden CD-ROM.

12.3 Standardkonforme Seiten mit TYPO3 erstellen

Das Erstellen standardkonformer Seiten mit TYPO3 ist in den Varianten *HTML 4.01* und *XHTML 1.0 Transitional* vergleichsweise einfach möglich.

Um die TYPO3-Standardeinstellung von *HTML 4.01* und *XHTML 1.0 Transitional* umzustellen, ist lediglich folgende Zeile im Root-Template erforderlich:

```
page.config.doctype = xhtml_trans
```

Generell sind folgende Werte möglich:

Wert für page.config.doctype	Ausgegebener Doctype
xhtml_trans	XHTML 1.0 Transitional
xhtml_frames	XHTML 1.0 Frameset
xhtml_strict	XHTML 1.0 Strict

Tabelle 12.1 Doctypes und TypoScript-Konfiguration

Wert für page.config.doctype	Ausgegebener Doctype
xhtml_11	XHTML 1.1
xhtml_20	XHTML 2
none	Gar kein Doctype

Tabelle 12.1 Doctypes und TypoScript-Konfiguration

Achtung Dies steuert nur die Ausgabe des Doctypes – es führt nicht zu einer entsprechenden Ausgabe des HTML-Codes!

Zudem gibt es noch die Möglichkeit, den berüchtigten XML-Prolog zu steuern:

```
page.config.xmlprologue = none
```

schaltet die Ausgabe des Prologs ab – das ist notwendig, damit der Internet Explorer in den Standard-Modus geschaltet wird (zum Thema Rendering-Modi lesen Sie **http://www.fabrice-pascal.de/artikel/dtd (Linkcode 047)**).

12.3.1 XHTML-Cleaning

Die Konfigurationsoption:

```
config.xhtml_cleaning = all
```

schaltet eine Art XHTML-Säuberung ein, die allerdings noch nicht wirklich vollständig ist. Das Aktivieren der Option sorgt dafür, dass

- Element- und Attributnamen kleingeschrieben werden,
- Einzelelemente (oder
) geschlossen werden,
- Anführungszeichen um alle Attribute gesetzt werden,
- ALT-Attribute in IMG-Elemente gesetzt werden, wenn diese fehlen.

Weitergehende Erfordernisse von XHTML, wie die korrekte Verschachtelung oder die CDATA-Umschließung für Skripte, werden allerdings noch nicht berücksichtigt.

Als Option kann gesteuert werden, wann der Code durch den »Cleaner« geschickt wird:

- all – immer
- cached – wenn die Seite in den Cache geschrieben wird
- output – bevor die Seite ausgegeben wird

Des Weiteren existiert eine Extension »Advanced HTML Cleaner« (**http://typo3.org/extensions/repository/search/qcom_htmlcleaner/ (Linkcode 048)**), die weitere Verbesserungen des Codes ermöglicht. So werden je nach Doctype unerlaubte Elemente und Attribute entfernt oder verändert. Durch Anpassen des Quellcodes der Datei **class.ux_t3lib_parsehtml.php** (**/typo3conf/ext/qcom_htmlcleaner/**) können hier auch eigene Umschreibungen definiert werden. Die Verwendung erfordert das Aktivieren von `config.xhtml_cleaning`.

Ein Problem bei der Realisierung valider Websites mit TYPO3 sind Extensions, die oft nur mit erheblichen Änderungen im PHP-Code zu standardkonformer Ausgabe gebracht werden können. Bei restriktiveren Doctypes wie *XHTML 1.0 strict* und *XHTML 1.1* verschärfen sich diese Probleme noch.

12.3.2 Die Extension »Page Validator«

Die Extension »Page Validator« (*sf_validator*) hilft, die Validität von Seiten zu überprüfen. Mit einem Klick zeigt sie die Ergebnisse einer Prüfung mit dem Validator des W3C.

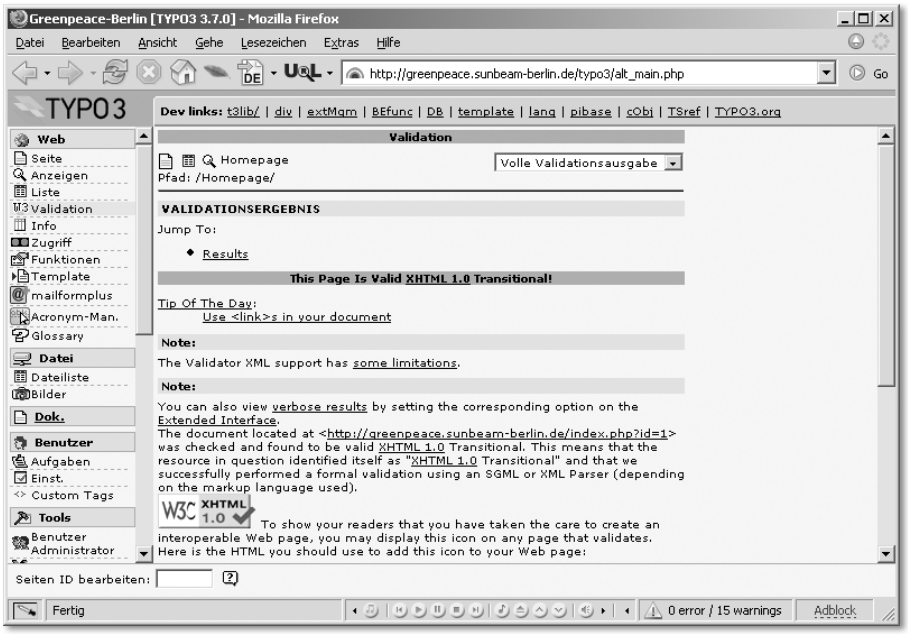


Abbildung 12.1 Gratulation vom W3C – diese Seite ist valide!

Installation und Bedienung sind einfach: Laden Sie die Extension aus dem Online-Repository herunter (Extension Key: *sf_validator*), und installieren Sie sie. Danach steht im Modul **Web** (nach dem Neuladen der Seite) das Submodul

Validation zur Verfügung. Zum Validieren muss die betreffende Seite nur noch angewählt werden. Über das Funktionsmenü können noch einige wenige Einstellungen vorgenommen werden – z.B. wie ausführlich die Kommentare des Validators angezeigt werden.

12.4 Barrierefreie Websites mit TYPO3

Die Probleme beim Erstellen barrierefreier Seiten liegen bei TYPO3 vor allem in folgenden Bereichen:

1. Es fehlt die Möglichkeit, typische Anforderungen an Barrierefreiheit über das Backend zu realisieren. So gibt es im »Text-Bild«-Objekt standardmäßig keine Möglichkeit, einen alternativen Bildtext (`alt`-Attribut) oder eine Langbeschreibung (`longdesc`) einzugeben. Das Objekt »Tabelle« lässt so wichtige Dinge wie die `<caption>` vermissen oder die Möglichkeit, Tabellenkopfzeilen als solche zu markieren.
2. Die Inhaltselemente werden weitgehend über komplizierte Tabellenkonstruktionen positioniert. Neuere Extensions wie »CSS Styled Content« beginnen zwar mit einem mehr CSS-orientierten Layout, dies ist aber bei weitem noch nicht vollständig.
3. Einige häufig verwendete Extensions (z.B. »Indexed Search«) sind ebenfalls noch auf Basis von Tabellenlayouts realisiert.

Zusätzlich animieren die ausgezeichneten Grafikfunktionen von TYPO3 viele Designer dazu, rein grafische Menüs zu entwerfen – auch kein Plus für barrierefreie Sites, da grafische Texte nicht vergrößert werden können.

12.4.1 Vorhandene Hilfsmittel

Neben den Einschränkungen gibt es für TYPO3 auch einige Werkzeuge, die beim Erstellen barrierefreier Seiten helfen.

Der erste Ansatz, Gestaltung über CSS zu realisieren, war das statische Template »cSet stylesheet«. Dieses ist inzwischen aber durch »CSS Styled Content« überholt und sollte nicht mehr verwendet werden.

Inzwischen existiert eine Reihe von Extensions, die sich der Aufgabe widmen, TYPO3 mehr Barrierefreiheit beizubringen.

CSS Styled Content (css_styled_content)

Die wichtigste verfügbare Erweiterung für barrierearmen Code ist »CSS Styled Content« (*css_styled_content*), mit der ein tabellenfreier Code erzeugt wird – zumindest in einigen Bereichen.

»CSS Styled Content« setzt im Wesentlichen TypoScript-Vorgaben für die Inhaltselemente ein, die auf Font-Elemente zur Gestaltung verzichten (im Gegensatz zu »Content (default)«), und schreibt die Funktionen für die Inhaltselemente

- Bulletlist (wird als ungeordnete HTML-Liste ausgegeben)
- Table
- Dateiliste (wird allerdings immer noch als Tabelle ausgegeben)

in **/tslib/tslib_pibase** neu.

Leider versetzt uns »CSS Styled Content« allein noch nicht in die Lage, barrierefreie Websites zu erstellen. Es berücksichtigt nicht alle Inhaltselemente, insbesondere nicht das wichtige »Text mit Bild«-Objekt.

Außerdem wird aus Gründen der Verallgemeinerbarkeit ein wahrer »Klassen-Overkill« erzeugt, der viele Elemente mit für die meisten Anwendungsbereiche unnötig vielen Klassen oder nicht sinnvollen Attributen ausstattet.

Trotzdem ist es sinnvoll, sich mit »CSS Styled Content« ausführlicher zu befassen. Die dort eingesetzten Methoden lassen sich für eigene Projekte abwandeln oder zeigen, wie eigene Accessibility-Ergänzungen entwickelt werden können.

Daher haben wir im Anhang B.3 die Erweiterung im Detail unter die Lupe genommen und den Quellcode Zeile für Zeile kommentiert.

CSS Styled Imagetext (css_styled_imgtext)

Eine große Lücke in »CSS Styled Content« soll die Erweiterung »CSS Styled Imagetext« (CSI) stopfen: die Realisierung eines »Text mit Bild«-Objekts ohne Layouttabellen.

Die Erweiterung befindet sich allerdings noch im Stadium »Experimental«.

Dazu gehört dann auch gleich die Extension »CSS styled IMGTEXT with alt and title attributes« (*sl_css_imgtext*), die `alt`-Texte und `title` nachrüstet.

Accessible XHTML Template (gov_accessibility)

Dieses Template ist eine in weiten Teilen anpassbare, fertige barrierefreie Site – eine Art lebende »Best-Practice-Studie«.

Nach Installation der Erweiterung steht ein komplettes Website-Layout mit Features wie Schriftgrößen- und Layoutauswahl zur Verfügung. Durch Verwendung des Objektbrowsers können Farben, Bilder und weitere Eigenschaf-

ten angepasst werden. Die Extension nutzt weitere Extensions wie »CSS Styled Content« und »CSS Styled Imagetext«.

http://typo3.org/documentation/document-library/gov_accessibility/ (Linkcode 049)

Accessible Tables (accessible tables)

Diese neue Erweiterung befasst sich mit dem Tabellenelement und ergänzt es um Barrierefreiheitsfunktionen – mehr dazu finden Sie im folgenden Abschnitt bei den Inhaltselementen.

Gov Textmenu und gov_accesskey

Diese beiden Extensions rüsten das TMENU um die Attribute `acceskey` und `tabindex` nach.

Allerdings wird die Verwendung dieser Attribute von Accessibility-Spezialisten inzwischen nicht mehr empfohlen (siehe **http://www.einfach-fuer-alle.de/artikel/tabindex/ (Linkcode 050)** und **http://www.einfach-fuer-alle.de/artikel/formulare/tag4/ (Linkcode 051)**). Daher kann auf den Einsatz dieser Extensions verzichtet werden.

Akronymmanager

Der »Akronymmanager« (*sb_akronymmanager)* ist ein Hilfsmittel, um die für Barrierefreiheit erforderlichen Auszeichnungen von Abkürzungen (`<abbr>`), Akronymen (das sind Abkürzungen, die als zusammenhängendes Wort gesprochen werden, wie z.B. UNO – als `<acronym>` zu kennzeichnen) und Sprachwechseln zu erleichtern.

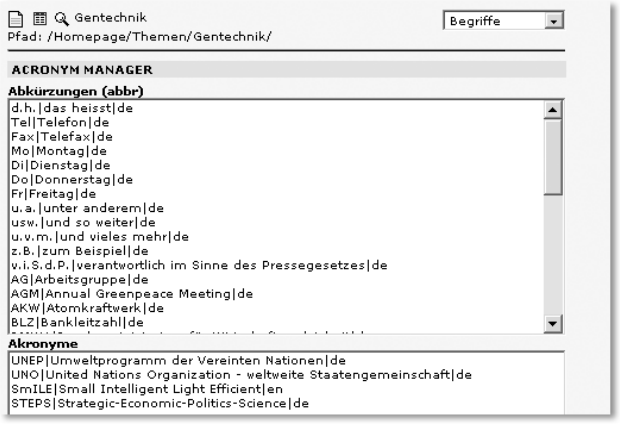


Abbildung 12.2 Der Acronymmanager im Einsatz

Über ein neues Backend-Modul können in drei Feldern Begriffe eingegeben werden, die dann bei der Ausgabe vollautomatisch mit den passenden HTML-Tags umschlossen werden.

Sie finden den Acronymmanager auf der CD-ROM, im Online-Repository oder unter **http://typo3.sunbeam-berlin.de/acronymmanager/ (Linkcode 052)**.

Die Extension befindet sich im Stadium »alpha« und ist noch nicht voll funktionsfähig. So werden z.B. keine Elemente zusammengefasst (wenn ein fremdsprachiges Wort allein verlinkt ist, sollten die `lang`-Attribute direkt in das Linkelement geschrieben werden). Die Extension wird jedoch ständig weiterentwickelt.

12.5 TYPO3-Inhaltselemente barrierefrei gestalten

12.5.1 Überschrift und Text

Verhältnismäßig unproblematisch sind die Elemente »Überschrift« und »Text«. Hier sind vor allem Anpassungen am TypoScript und eine sinnvolle Konfiguration des RTEs erforderlich.

Zusätzlich müssen noch die **spacer.gif**-Dateien entfernt werden, die beim Eintragen von Abständen nach oben oder unten (1) erzeugt werden. Diese könnten z.B. durch ein `<hr>`-Element mit per Style zugewiesener Höhe ersetzt werden.

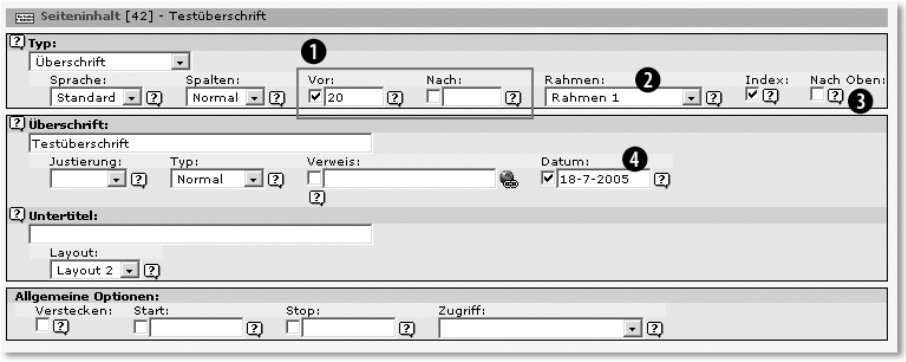


Abbildung 12.3 Überschrift- und Textelemente sind verhältnismäßig problemlos.

Funktionen wie »Rahmen« (2), »Nach oben«-Link (3) und »Datum« (4) werden bereits durch »CSS Styled Content« angepasst (wenn auch etwas überschwänglich mit Klassen versehen).

Die Konfiguration des (bzw. eines) Editors (RTE) ist etwas kniffliger, und letztlich hängt die Konfiguration von den konkreten Einsatzbedingungen ab.

1. Alle Bedienbuttons, die problematische Elemente realisieren, sollten ausgeblendet werden, z.B. die Buttons zur Schriftformatierung per FONT-Tag.
2. Zusätzlich können passende Klassen und benutzerdefinierte Tags definiert werden (z.B. `<cite>` für Zitate).

Der Konfiguration des RTE haben wir ein eigenes Kapitel gewidmet – lesen Sie dazu in den Abschnitten 7.3, *Rich Text Editor*, und 7.4.3, *Eine Beispiel Konfiguration für den RTE*, weiter.

12.5.2 Text-Bild und Bilder

Ein größeres Problem ist der Umgang mit Bildern und Text-Bild-Kombinationen. Hier setzt TYPO3 standardmäßig auf komplexe Tabellenkonstruktionen, die auch in »CSS Styled Content« nicht barrierefrei sind. Zudem fehlen im Backend Eingabemöglichkeiten für ALT-Texte oder LONGDESC-Beschreibungen.

Ein Versuch, die Formatierung auf modernes CSS-Layout umzustellen, ist die Erweiterung »CSS styled IMGTEXT« (*css_styled_imgtext*) von Ingmar Schlecht. Hier werden Bilder konsequent ohne Tabellen positioniert. Allerdings ist die Erweiterung noch im Stadium »Experimental« und kann nur eingeschränkt eingesetzt werden.

Für die notwendigen ALT-Texte gibt es ebenfalls Extensions – »Alttext for Images« (*dmc_image_alttext*) und »CSS styled IMGTEXT with alt and title attributes« (*sl_css_imgtext*) bei Verwendung von »CSS styled IMGTEXT«.

Eine andere Methode ist die Verwendung einer eigenen Erweiterung, um die Ausgabe von Bild-Text-Elementen anzupassen.

Auf der CD-ROM finden Sie die von uns erstellte Erweiterung »Accessible Content«, die auf »CSS Styled Content« basiert, aber die verschiedene Inhaltselemente anders aufbaut. Bild-Text-Elemente werden hier je nach Einsatz als einfaches IMG-Objekt, DIVs oder Definitionslisten realisiert (Definitionslisten sind eine interessante Methode, um Bild und Bildunterschrift semantisch korrekt abzubilden).

12.5.3 Aufzählung (Punktliste)

Für Aufzählungen sieht HTML die Elemente `<ul>` oder `<ol>` vor. In »CSS Styled Content« wird die Punktliste semantisch korrekt und barrierefrei ausgegeben.

12.5.4 Tabelle

Für barrierefreie Tabellen sind eine Reihe von Hilfen vorgesehen: So müssen Tabellenköpfe als `<th>` gekennzeichnet sein, und es muss eine Zusammenfassung verfügbar sein. Tabelle 12.2 zeigt erforderliche Eigenschaften von barrierefreien Tabellen.

HTML-Element/Attribut	Funktion
`<caption>`	Überschrift der Tabelle
Verwendung von `<th>`	Markiert eine Zelle als Kopfzelle.
`headers="ID"`	Attribut für `<td>` Ordnet eine Datenzelle der Kopfzelle ID zu.
`scope="col", scope="row"`	Attribut für `<th>` Ordnet einer Kopfzelle eine Spalte oder Zeile von Datenzellen zu.
`scope="colgroup", scope="rowgroup"`	Attribute für `<th>` Ordnet eine Überschrift mehreren untergeordneten Überschriften zu.
`summary`	Attribut für `<table>` Zusammenfassung der Tabelleninformation
`<thead>`, `<tfoot>`, `<tbody>`	Unterteilen die Tabelle in Kopf, Fußzeile und Rumpf; `<thead>` oder `<tfoot>` können auch wegfallen; `<tbody>` ist obligatorisch, wenn `<thead>` oder `<tfoot>` verwendet werden. Die Bereiche müssen in der nebenstehenden Reihenfolge notiert werden.

Tabelle 12.2 Elemente barrierefreier Tabellen

Eine barrierefreie Tabelle kann z.B. so aussehen:

Barrierefreie Tabelle

	Spalte 1	Spalte 2	
		Subspalte 2	Subspalte 3
Reihe 1			
Reihe 2			

Abbildung 12.4 Barrierefreie Tabelle in normalem Browser

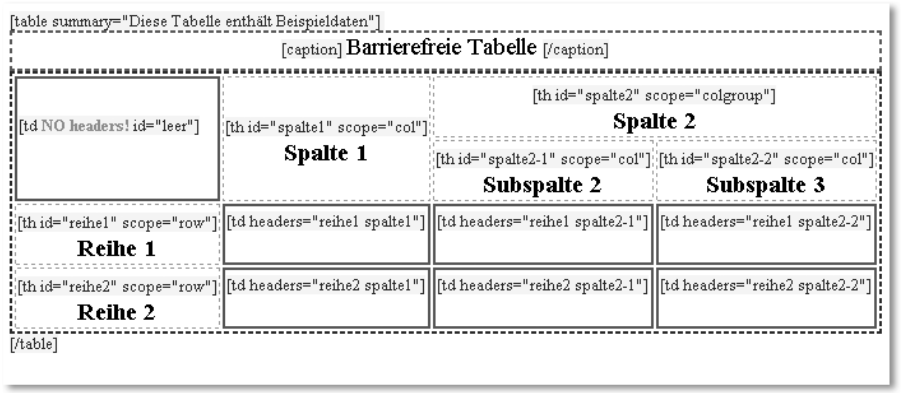


Abbildung 12.5 Besondere Accessibility-Funktionen sichtbar gemacht

Die gezeigte Tabelle wird mit folgendem HTML-Code erzeugt:

```
1:  <table border="1" summary="Diese Tabelle enthält
    Beispieldaten">
2:     <caption>Barrierefreie Tabelle</caption>
3:     <thead>
4:        <tr>
5:           <td rowspan="2" id="leer"> </td>
6:           <th rowspan="2" id="spalte1" scope="col">Spalte 1
             </th>
7:           <th colspan="2" id="spalte2" scope="colgroup">Spalte 2
             </th>
8:        </tr>
9:        <tr>
10:          <th scope="col" id="spalte2-1">Subspalte 2</th>
11:          <th scope="col" id="spalte2-2">Subspalte 3</th>
12:       </tr>
13:    </thead>
14:    <tbody>
15:       <tr>
16:          <th id="reihe1" scope="row">Reihe 1</th>
17:          <td headers="reihe1 spalte1"> </td>
18:          <td headers="reihe1 spalte2-1"> </td>
19:          <td headers="reihe1 spalte2-2"> </td>
20:       </tr>
21:       <tr>
22:          <th id="reihe2" scope="row">Reihe 2</th>
23:          <td headers="reihe2 spalte1"> </td>
```

```
24:          <td headers="reihe2 spalte2-1"> </td>
25:          <td headers="reihe2 spalte2-2"> </td>
26:       </tr>
27:    </tbody>
28: </table>
```

Listing 12.1 Quelltext einer barrierefreien Tabelle

Das Standard-Element für Tabellen bietet keine Möglichkeiten, diese Angaben im Backend vorzusehen.

Um Tabellen zu realisieren, stehen in TYPO3 neben dem Standard-Tabellenmodul verschiedene Extensions zur Verfügung. Einige davon eignen sich auch, um barrierefreie Tabellen herzustellen.

Extended Table Backend (mit Extended Table Service)

Ein komfortables Interface für die Konstruktion auch komplexer Tabellen bietet das »Extended Table Backend« (*th_exttable*). Hier kann über einen komplett neu geschriebenen Wizard eine Tabelle auch mit spalten- oder zeilenübergreifenden Zellen gestaltet werden. Die Bedienung ist zwar etwas ungewohnt und stellenweise ein wenig umständlich, aber dafür gibt es reichhaltige Konfigurationsmöglichkeiten für einzelne Zellen oder die gesamte Tabelle.

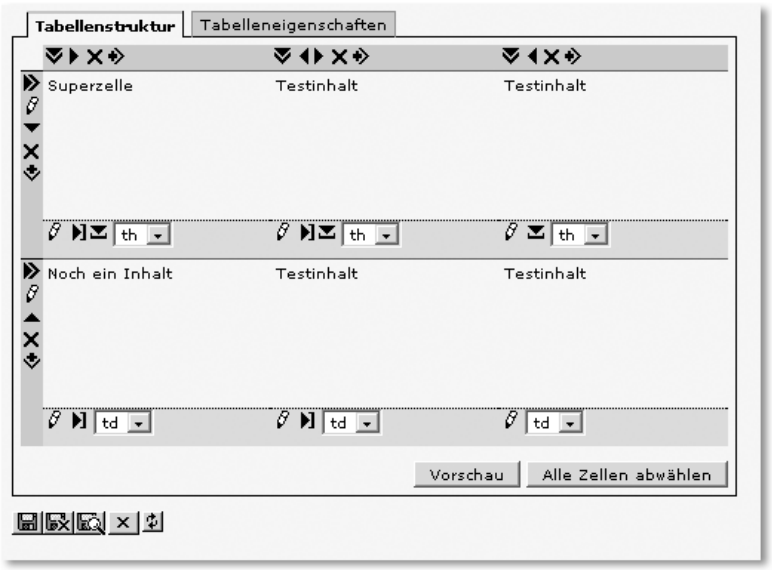


Abbildung 12.6 Ein komfortabler Editor macht auch komplexe Tabellen möglich.

Leider fehlen die meisten Funktionen, die eine barrierefreie Tabelle benötigt: `scope` und `headers` wird bislang nicht unterstützt, und auch `caption` ist noch nicht möglich (steht aber laut Autor auf der To-Do-Liste).

Da der Editor den kompletten Tabellencode in das Inhaltselement schreibt, lassen sich die fehlenden Attribute dort immerhin »per Hand« nachrüsten.

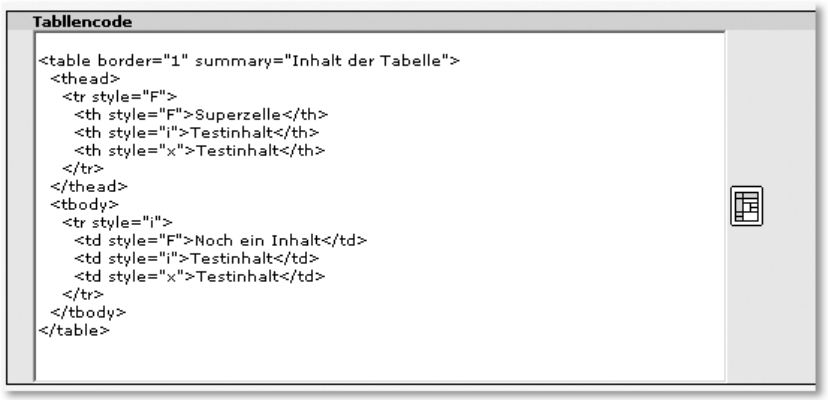


Abbildung 12.7 Der fertige Quelltext kann per Hand nacheditiert werden.

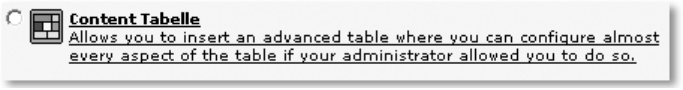


Abbildung 12.8 Ein neues Inhaltselement ermöglicht das Einfügen komplexer Tabellen.

Die Erweiterung ist als »beta« gekennzeichnet.

KB Content Table

Die Erweiterung »KB Content Table« (*kb_conttable*) bietet einen auf Flexforms basierenden Tabelleneditor, der alle nur denkbaren Optionen zur Formatierung der Tabelle und einzelner Zellen bietet (siehe Abbildung 12.9).

Sie können dabei die Editierrechte detailliert für Redakteure festlegen und einschränken. Mit dieser Erweiterung sind auch barrierefreie Tabellen möglich – jedoch ist vorher einiges an Konfigurationsarbeit nötig –, allerdings liefert Bernhard Kraft eine ausführliche Dokumentation mit, die Sie unter **http://typo3.org/documentation/document-library/kb_conttable/ (Linkcode 053)** finden.

Die Erweiterung ist als »alpha« spezifiziert und erfordert TYPO3 3.8.

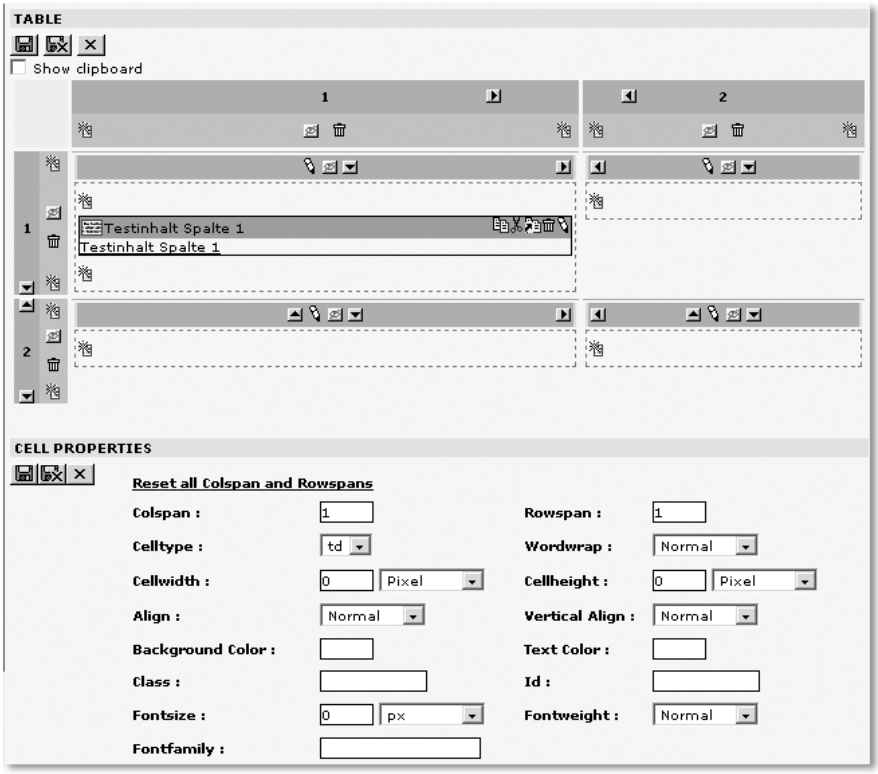


Abbildung 12.9 Konfigurationsoptionen für Tabellen mit KB Content Table

Accessible Tables

»Accessible Tables« erweitert das normale Inhaltselement »Tabelle« um einige Barierefreiheitsfunktionen:

- `<thead>` `<tbody>` und `<tfoot>` zur Gruppierung von Tabellenbereichen
- `<caption>` zur Tabellenbeschreibung
- Tabellenkopfzeilen (`<th>`)
- Attribut `scop` in den Kopfzeilen
- Attribut `summary`

Außerdem gibt es die Möglichkeit, alle üblicherweise von TYPO3 angebrachten Klassen zu entfernen (bei einfachen Tabellen zu empfehlen) und `<p>` in den Tabellenzellen zu entfernen.

Diese Erweiterung ist eine gute Möglichkeit, einfache Tabellen barrierefrei zu gestalten – allerdings setzt sie auf die Verwendung von `scope` statt `headers`, was auch korrekt ist, aber in der Praxis kaum unterstützt wird.

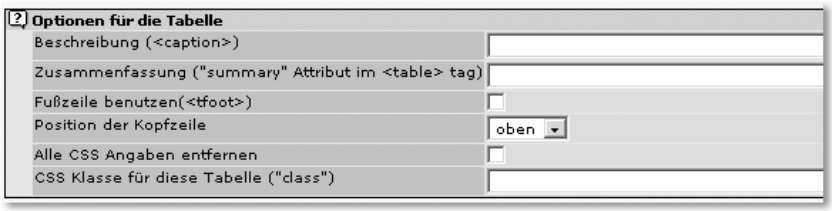


Abbildung 12.10 Zusätzliche Tabellenoptionen mit Accessible Tables

Sie können die Erweiterung unter dem Key *accessible_tables* finden – sie ist als »beta« eingestuft.

12.5.5 Dateiliste

Das Inhaltselement »Dateiliste« bietet verschiedene Layouts zur Darstellung von Dateilisten an – alle werden über Tabellen realisiert.

Nun kann man argumentieren, dass Dateilisten mit Dateinamen, Beschreibung, Icons und Dateigröße durchaus tabellarische Daten seien und daher die Verwendung einer Tabelle rechtens wäre. Allerdings fehlen der resultierenden Tabelle all die oben besprochenen Barrierefreiheitseigenschaften, so dass sie in keinem Fall barrierefrei ist.

Wie genau eine Dateiliste zu realisieren ist, hängt auch von deren Komplexität ab:

Einfache Listen, die nur aus einem verlinkten Dateinamen bestehen, lassen sich hervorragend mit dem HTML-Element `<ul>` oder `<ol>` erstellen.

Auch das Element »Definitionsliste« (`<dl>`) eignet sich zur Darstellung einer Download-Liste – z.B. so:

```
<dl>
   <dt><a href="linkurl">Dateiname</a></dt><dd>Dateibeschreibung
   (PDF, 45kb)</dd>
</dl>
```

Für komplexe Listen schließlich ist eine Tabelle geeignet – dann allerdings mit Tabellenkopf und für Barrierefreiheit optimiert.

Der momentan einzige Weg dorthin führt über die Neu-Definition der Funktion **render_uploads**, die in **class.tx_cssstyledcontent_pi1.php** definiert ist (»CSS styled content«).

Dies kann z.B. über eine User-Funktion realisiert werden, die über TypoScript (**tt_content.uploads.20**) angesprochen wird.

12.5.6 Multimedia

Bei der Verwendung von Multimedia-Elementen ist sicher die Einbindung das kleinste Problem in Sachen Barrierefreiheit ...

Die BITV verlangt, zu Multimedia-Inhalten entsprechende Alternativen bereitzustellen. Bei Videos können dies Untertitel oder Audiodeskriptionen sein. Das W3C befürwortet den Einsatz der »Syncronized Multimedia Integration Language (SMIL)« – damit können Multimedia-Inhalte mit zusätzlichen Informationen, z.B. Untertiteln, ausgestattet werden.

Das Thema hier weiter zu vertiefen würde allerdings den Rahmen dieses Buches sprengen. Weitere Informationen zur Gestaltung von barrierefreien Multimedia-Inhalten finden Sie bei »Einfach für Alle« (**http://www.einfach-fuer-alle.de (Linkcode 054)** – füttern Sie die interne Suche mit »multimedia«) oder im Buch »Barrierefreies Webdesign« von Jan Eric Hellbusch.

Ein nützliches Software-Werkzeug zur Untertitelung von Multimedia-Daten (und mehr) ist MACpie (**http://ncam.wgbh.org/webaccess/magpie/ (Linkcode 055)**).

TYPO3 setzt standardmäßig das Element `<embed>` für die Einbettung von Multimedia-Elementen ein. Leider ist `<embed>` kein offizielles HTML-Element, sondern eine proprietäre Erfindung von Netscape.

Zur Einbettung von Objekten sieht (X)HTML das Element `<object>` vor. Leider birgt die Einbettung mittels `<object>` einige Risiken (für Safari-Browser funktioniert sie leider nicht).

Geoff Stearns empfiehlt daher die Verwendung von Javascript zur Einbettung von Flash: **http://blog.deconcept.com/2005/03/31/proper-flash-embedding-flashobject-best-practices/ (Linkcode 056)**.

Für TYPO3 bedeutet dies, dass Multimedia-Objekte derzeit barrierefrei und standardkonform über das Inhaltselement »HTML« eingebunden werden (müssen).

12.5.7 Sitemap/Menü

Die Sitemap wird in der Standardeinstellung von TYPO3 ebenfalls über Tabellen realisiert. Es lässt sich jedoch ein barrierefreies Ergebnis über TypoScript-Änderungen bewerkstelligen. Hierfür müssen lediglich die diversen `<p>`-Tags, die um die einzelnen Navigationspunkte gewrappt werden, durch Listenauszeichnungen ersetzt werden.

Das TypoScript für eine Sitemap könnte dann folgendermaßen aussehen:

```
1:  ### Barrierefreie Sitemap-Konfiguration ###
2:  tt_content.menu.20.2 = HMENU
3:  tt_content.menu.20.2.stdWrap.wrap =
4:  tt_content.menu.20.2 {
5:     1 = TMENU
6:     1.noBlur = 1
7:     1.expAll = 1
8:     1.target = _self
9:     1.wrap = <ul class="sitemap">|</ul>
10:    1.NO.allWrap =
11:    1.NO.linkWrap =
12:    1.NO.allWrap = |<span class="usb">. </span>
13:    1.NO.wrapItemAndSub = <li class="sm1">|</li>
14:    1.NO.ATagBeforeWrap = 0
15:    2 < .1
16:    2.wrap = <ul>|</ul>
17:    2.NO.wrapItemAndSub = <li class="sm2">|</li>
18:    3 < .2
19:    3.NO.wrapItemAndSub = <li class="sm3">|</li>
20:    4 < .2
21:    4.NO.wrapItemAndSub = <li class="sm4">|</li>
22: }
```

Listing 12.2 Barrierefreie Sitemap durch TypoScript

Ausgegeben wird für das Sitemap-Menü eine verschachtelte ungeordnete Liste. Für die richtige Betonung beim Vorlesen wird ein `<span class="usb">. </span>` hinter jedem Menüpunkt gesetzt.

Dies ist notwendig, damit Screenreader beim Vorlesen nach jedem Menüpunkt die Stimme senken und eine kleine Pause machen und nicht alles in einem Rutsch »ohnepunktundkomma« vorlesen.

12.5.8 HTML

Das »HTML«-Element birgt von sich aus wenig Probleme – viel passiert dort ja auch nicht.

12.5.9 Mailformular

Formulare erfordern besondere Aufmerksamkeit, um keine unnötigen Barrieren aufzubauen. Dazu stehen dem Entwickler verschiedene Mittel zur Verfügung:

HTML-Element	Funktion
`<fieldset>`	Fasst logisch zueinander gehörende Felder zusammen, z.B. die Felder »Name«, »Vorname« und »E-Mail« als persönliche Daten.
`<legend>`	Zu jedem Fieldset gehört eine Beschreibung der Gruppe (also eben »Persönliche Daten«).
`<label>`	Das Label kennzeichnet die Bezeichnung des Formularfeldes (beispielsweise »Name:«).

Tabelle 12.3 Bestandteile barrierefreier Formulare

Zusätzlich fordert die Barrierefreiheit, dass erforderliche Felder gekennzeichnet sind (üblicherweise durch ein *), dass aktive Felder markiert sind (per CSS sehr elegant möglich), und verlangt verständliche Fehlermeldungen bei der Validierung des Formulars. So sollte ein gutes Formular ohnehin aufgebaut sein.

Ferner muss die logische Struktur des Formulars durch die Reihenfolge der Felder gewahrt sein (nach dem Feld für den Vornamen sollte das Feld für den Nachnamen folgen). Wenn dies nicht durch die HTML-Struktur erreicht ist, kann dazu das Attribut `tabindex` eingesetzt werden. Es ist jedoch nicht unbedingt nötig, *tabindices* zu verwenden, wenn das Formular auch ohne diese logisch korrekt durchlaufen werden kann.

Prinzipiell nicht mehr nötig ist es, alle Felder generell vorzubelegen, wie es noch in der BITV gefordert ist. Diese Forderung ist von der technischen Entwicklung überholt. (In der BITV steht sie allerdings noch.)

Eine ausführliche Anleitung zur Erstellung barrierefreier Formulare hat »Einfach für Alle« zusammengestellt: **http://www.einfach-fuer-alle.de/artikel/formulare/ (Linkcode 057)**.

Um solche Formulare herzustellen, reicht das Inhaltselement »Mailform« nicht aus. Neben dem Tabellenlayout verhindern auch die fehlenden Elemente `<fieldset>`, `<legend>` und `<label>` den Einsatz.

Stattdessen kann die Erweiterung »th_mailformplus« (*th_mailformplus*) zur Erstellung barrierefreier Formulare verwendet werden. Hier kann ein HTML-Template zur Definition des Formulars eingesetzt werden.

Natürlich ist es nicht die ideale Lösung, auf HTML-Templates angewiesen zu sein – schließlich können die Redakteure nun keine eigenen Formulare mehr entwerfen. Wir haben daher in der schon angesprochenen Erweiterung »Accessible Content« den Formularwizard derart überarbeitet, dass er `<fieldset>`, `<legend>` und `<label>` verwendet und es somit auch dem Redakteur ermöglicht, einfache Formulare zu erstellen.

12.5.10 Suchformular

Das Standardformular für die Suche erzeugt ein per Tabelle positioniertes, nicht barrierefreies Formular. Es ist prinzipiell möglich, das Suchformular per TypoScript umzugestalten.

Abbildung 12.11 TypoScript-Optionen der Suche

Das Erscheinungsbild der Suche kann über den TypoScript-Objektbrowser (siehe auch Abschnitt 9.5.3, *Template Tool: TypoScript Object Browser*) konfiguriert werden. Unter *tt_content.search.20* befinden sich die meisten Parameter, die das Erscheinungsbild des Formulars festlegen. Außerdem existiert eine TypoScript-Variable `accessibility`. Wenn diese auf »1« gesetzt wird, werden alle `<label>` mit dem Attribut `for="..."` und der Bezeichnung des `input`-Feldes belegt.

Allerdings wird es ziemlich schwierig, tatsächlich auch `<fieldset>`-Elemente unterzubringen, da es keine offensichtliche Möglichkeit gibt, um Gruppen von Formularfeldern nebst zugehörigen `<label>` noch weitere Elemente zu »wrappen«.

Eine Alternative ist die Verwendung der – ohnehin wesentlich leistungsfähigeren – Extension »Indexed Search«. Dort lässt sich ein Formular als HTML einfügen und recht einfach konfigurieren (zusätzlich steht die Erweiterung »Searchbox for Indexed Search« (*macina_searchbox*) zur Verfügung, mit der ein eigenes HTML-Template für das Suchformular verwendet werden kann. Das neue Suchformular ist einfach als Inhaltselement auswählbar).

Leider ist die Ausgabe von »Indexed Search« nicht barrierefrei, so dass dann doch wieder der Blick in den Quellcode erforderlich ist. Der Aufbau der Ergebnisliste geschieht in der Datei **pi1/class.tx_indexedsearch.php** im Verzeichnis

der Extension. Wir zeigen in der Extension »Accessible Content« (siehe auch Kapitel 12.9, *Accessible Content*), wie eine solche umgestaltete Ergebnisseite aussehen kann.

12.5.11 Anmeldeformular

Auch das hauseigene Anmeldeformular führt nicht zu barrierefreiem Code. Und auch hier gibt es Alternativen – die Erweiterung »New front end login box« (*newloginbox*) bietet ohnehin mehr Möglichkeiten als das klassische Login. Darüber hinaus erlaubt es eine weitere Erweiterung »Loginbox with template for newloginbox plugin« (*arotea_loginbox*), eigene HTML-Templates einzusetzen, die barrierefrei gestaltet sein können.

12.5.12 Plugins

Bei den Plugins ist die Situation recht unterschiedlich. Einige sehr wichtige Extensions, wie z.B. »News« (*tt_news*) sind von Haus aus auf Barrierefreiheit getrimmt oder bieten die Möglichkeit, über Templates die Ausgabe zu beeinflussen.

Leider erzeugt die wichtige »Indexed Search« immer noch ein weitgehend unzugängliches Tabellengerüst – das soll sich erst mit TYPO3 4.0 ändern. Zumindest sollen dann Templates möglich sein.

12.5.13 Sprungmarken à la XHTML

Während in HTML 4 noch das Attribut »name« sowohl in Formularen als auch in benannten Ankern (`<a name="87">`) üblich war (und von TYPO3 standardmäßig eingesetzt wird), ist es in XHTML unerwünscht (»deprecated«) bzw. für FORM und IMG in der Variante »strict« und in XHTML 1.1 nicht mehr erlaubt.

Um standardkonforme Dokumente zu erstellen, müssen wir also sowohl die Anker selbst umbauen als auch dafür sorgen, dass automatisch erzeugte Links von TYPO3 entsprechend angepasst werden.

Da IDs auch nicht mit einer Zahl beginnen dürfen, reicht ein einfaches Umkopieren nicht aus. Stattdessen muss mit:

```
tt_content.stdWrap.dataWrap = <a name="anker{field:uid}"
id="anker{field:uid}"></a>
```

noch ein Präfix vorangestellt werden. Für das Erzeugen der Links muss eine neue Funktion her (*userfunction*):

```
includeLibs.xhtmlanchor = fileadmin/user_typolink.inc
```

```
tt_content.text.20.parseFunc.tags.link.typolink.parameter.
postUserFunc = user_xhtmlAnchor
```

Die zugehörige PHP-Funktion lautet (als **user_typolink.inc** in **fileadmin** speichern bzw. den Pfad oben entsprechend anpassen):

```
<?php
function user_xhtmlAnchor($content,$conf) {
   $uriparts = explode('#',$content);
   if (isset($uriparts['1'])) {
     $content = $uriparts['0'].'#anker'.$uriparts['1'];
   }
   return $content;
}
?>
```

Stellen, die automatisch Links auf Anker erzeugen, müssen auch angepasst werden. Das Menü **Abschnittsübersicht** (*sektion menu*) wird beispielsweise mit:

```
tt_content.menu.20.3.renderObj.typolink.section.wrap = anker|
```

angepasst. Diese Lösung stammt von Andreas Schwarzkopf und wurde in der TYPO3-Newsgroup **typo3.projects.content-rendering** veröffentlicht.

12.6 Barrierefreie Menüs mit TYPO3

12.6.1 Textmenüs

Die beste Art, eine Navigation zugänglich zu machen, ist die Realisierung als Textmenü mittels des HTML-Elements `<ul>`.

Das ist zugänglich, semantisch korrekt (denn was ist ein Menü anderes als eine Liste von Links?) und lässt sich auch sehr gut per CSS stylen.

Wir gehen hier nicht weiter auf die Vorteile und Möglichkeiten von Listen als Menü ein – wenn Sie mehr zum Thema erfahren wollen, bieten diese Websites einen guten Einstieg:

- »Einfach für Alle«: Barrierefreie Navigationsmenüs (**http://www.einfach-fuer-alle.de/artikel/menues/tag1/ (Linkcode 058)**)
- »Listamatic« (Listen per CSS gestalten): **http://css.maxdesign.com.au/ (Linkcode 059)**

Für die TypoScript-Realisierung hat eine Liste den Vorteil, dass sie recht einfach anzulegen ist. Im simpelsten Fall genügt Folgendes für ein dreifach verschachteltes Menü:

```
1:  hmenue = HMENUE
2:  hmenue.1 = TMENU
3:     hmenue.1 {
4:        expAll = 0
5:        noBlur = 1
6:        wrap = <ul>|</ul>
7:        NO.wrapItemAndSub = <li>|</li>
8:     hmenue.2 = TMENU
9:     hmenue.2 < hmenue.1
10: hmenue.3 = TMENU
11: hmenue.3 < hmenue.2
```

Listing 12.3 TypoScript-Anweisungen für ein Menü als HTML-Liste

Um die Zugänglichkeit zu verbessern, ergänzen wir noch zwei Dinge:

1. eine versteckte Nummerierung für Screenreader
2. ».« zur Verbesserung der Aussprache

Gerade bei langen Menüs geht die Übersicht beim Vorlesen schnell verloren. Daher ist es ratsam, eine hierarchische Nummerierung vorzusehen (mehr dazu finden Sie auch im oben erwähnten »Einfach-für-Alle«-Artikel). Dazu nutzen wir das Definition-Element `<dfn>` und eine von Jan Wischnat geschriebene Benutzerfunktion.

Im TypoScript ergänzen wir:

```
hmenue.1.IProcFunc = user_IProc_dfn
```

Vorher müssen wir die User-Funktion einbinden:

```
page.includeLibs.dfn = fileadmin/inc/dfn_iproc_tmenu.inc
```

Sie finden die Funktion auf der beiliegenden CD-ROM.

Leider haben Screenreader die unangenehme Eigenschaft, aufeinander folgende Links direkt hintereinander vorzulesen, was die Übersicht nicht gerade verbessert. Daher müssen wir noch einen »Stopper« am Ende des Links einbauen, damit der Screenreader eine kleine Pause macht – wie am Ende eines Satzes.

```
<ul>
   <li><dfn>1:</dfn>
<a href="...">Link 1</a><span class="usb">. </span></li>
   …
</ul>
```

Im TypoScript ist das einfach einzufügen:

```
NO.wrapItemAndSub = <li>|<span class="usb">. </span></li>
```

Da wir natürlich weder die Definition noch den Punkt sehen wollen, blenden wir sie per CSS aus und notieren im Stylesheet:

```
li dfn, .usb {
   display: block;
   position: absolute;
   left: -3000px;
   height: 0px;
   width: 0px;
}
```

Einfacher wäre `display:none`, aber auf diese Weise versteckte Elemente werden leider auch von den verbreitetsten Screenreadern nicht gelesen ...

Fertig ist das barrierefreie Textmenü – Sie können und sollten es nach Ihrem Bedarf mit zusätzlichen Optionen für TMENU ergänzen.

Ein Title-Attribut können Sie mit:

```
NO.ATagTitle.field = abstract // description
```

einem Menüstatus hinzufügen – hier dem normalen Zustand. Der Text für den TITLE wird dem Feld *abstract* entnommen – und wenn dieses leer ist, der *description* (die Verwendung des verlinkten Textes selbst als TITLE wäre allerdings eine sinnlose Verdoppelung und ist daher überflüssig).

Sinnvoll ist, die aktuelle Seite nicht zu verlinken.

Mit der Anweisung:

```
CUR < .NO
CUR.doNotLinkIt = 1
```

schalten Sie die Verlinkung für die aktuelle Seite ab.

Von der Verwendung von *Accesskeys* und *Tabindex-Markierungen* raten wir ab, da die meisten Buchstaben bereits durch andere Software vorbelegt sind und eigene Definitionen hier nur stören (lesen Sie dazu **http://2bweb.de/accesskey/ (Linkcode 060)**). Für gut strukturierte Dokumente sind auch Tabindizes nicht erforderlich – bei der Verwendung von Sprungmenüs können sie sogar stören (auch hierzu gibt es eine weiterführende Quelle: **http://www.einfach-fuer-alle.de/artikel/tabindex/ (Linkcode 061)**).

12.6.2 Grafische Menüs

Grafische Menüs der herkömmlichen Art – realisiert mit Tabellen und ohne Accessibility-Hilfen – stellen erhebliche Barrieren für behinderte Nutzer dar. Das fängt damit an, dass als Grafik realisierte Schrift nicht vergrößert werden kann. Wenn der Entwickler nicht wenigstens `alt`-Attribute vergeben hat, lesen Screenreader den Dateinamen vor, was meist wenig weiterhilft.

Daher ist aus Sicht der Barrierefreiheit generell von grafischen Menüs abzuraten – im Übrigen sind sie auch zur Suchmaschinenoptimierung und für die Download-Zeit eher schädlich.

Modernes Webdesign nutzt daher – wenn es denn Grafiken sein sollen – eine Technik, die »Image Replacement« genannt wird. Dabei wird ein normaler Text per CSS ausgeblendet und an seiner Stelle eine Hintergrundgrafik angezeigt. Es gibt inzwischen eine ganze Reihe verschiedener IR-Techniken (eine Diskussion der Vor- und Nachteile finden Sie unter **http://www.meiert.com/de/releases/20050513/ (Linkcode 062)** von Jens Meiert).

12.7 Ein barrierefreier RTE

Für eine barrierefreie Ausgabe empfiehlt es sich, den RTE möglichst restriktiv zu konfigurieren. »Schädliche« Elemente wie `<font>` können durch das Deaktivieren der Buttons »Textfarbe«, »Schriftart« und »Schriftgröße« unterbunden werden. Zusätzlich werden sie aus der Liste erlaubter Tags entfernt.

Wir haben der Konfiguration des RTE ein eigenes Kapitel gewidmet, so dass wir hier nur kurz auf die erforderlichen Maßnahmen eingehen – Details zur Konfiguration des RTE erfahren Sie in Abschnitt 7.3, *Rich Text Editor*.

Manche Funktionen sind mit dem Standard-RTE nicht realisierbar, so lassen sich `title`-Attribute erst mit der Version 0.5.1 des alternativen RTE »htmlArea RTE« zuweisen.

Ziel	Grund	Maßnahme
Keines der folgenden Elemente soll verwendet werden: <font>, <b>, <i>, <u>	Diese Elemente sind veraltet oder führen zu rein visuellem Code (z.B. sollte statt <b> verwendet werden, das wird dann auch vom Screenreader entsprechend interpretiert).	Ausblenden der Buttons »Schriftart«, »Schriftgröße«, »Schriftfarbe«, »Emoticons«
Verwendung der TYPO3-Inhaltselemente, wenn möglich	Die Verwendung des Inhaltselements »Tabelle« gibt uns mehr Kontrolle über die Realisierung und verhindert doppelten Aufwand bei der Anpassung der Funktionen.	Ausblenden der Buttons zur Tabellendarstellung
Verwendung semantischer Elemente wie <cite> und <q> für Zitate	Semantisch korrekte Elemente können auch von Screenreadern und anderen Ausgabegeräten passend interpretiert werden.	Verwendung des Buttons »Benutzerdefinierte Elemente«
Verwendung der vorgegebenen CSS-Eigenschaften zur Gestaltung	Redakteure sollten keine eigenen Formatierungen einsetzen, sondern das verwenden, was vom Designer festgelegt wurde. Das ist nicht nur für die Barrierefreiheit sinnvoll.	Ausblenden des Buttons »Hintergrundfarbe«; verwenden von Klassen zur Gestaltung über die Buttons »Absatzart« und »Zeichenart« und Definition dieser Klassen im TypoScript.

Tabelle 12.4 Konfigurierung eines barrierefreien RTE

12.8 Einbindung von Tidy

Tidy ist ein sehr nützliches Werkzeug, um (X)HTML-Code zu prüfen und »aufzuräumen«. Ursprünglich von Dave Raggett entwickelt, wird das Programm inzwischen von einer Entwicklergemeinde als Sourceforge-Projekt gepflegt (**http://tidy.sourceforge.net/ (Linkcode 063)**). Tidy existiert als Kommandozeilenversion für nahezu alle Betriebssysteme – es gibt sogar eine Online-Version (**http://www.thedumbterminal.co.uk/services/tidy.shtml (Linkcode 064)**).

Tidy prüft (X)HTML-Code auf Schreibfehler und falsch verschachtelte Tags, formatiert ihn mit Einrückungen übersichtlich und ersetzt gestalterische Anweisungen (<font>) durch entsprechende CSS-Eigenschaften. Auch zur Umwandlung von HTML in XHTML lässt sich Tidy verwenden.

Tidy ist als Kommandozeilen-Tool erhältlich und kann somit auch als nachgeordneter Parser für TYPO3-Code eingesetzt werden.

Tidy installieren und konfigieren

Tidy liegt sowohl als kompilierte Version für viele Systeme als auch im Quellcode vor. Wie die Installation im Detail funktioniert, richtet sich (natürlich) nach Ihrem Betriebssystem und ist in der Regel unkompliziert. Selbst für Windows liegt eine einfache ausführbare Datei vor, die keine spezielle Installation erfordert.

TYPO3 sieht bereits eine Verwendung von Tidy vor – im Install-Tool können Sie im Bereich »All Configuration – FE« (bzw. in `$TYPO3_CONF_VARS["FE"]`) Tidy einschalten (`[FE][tidy] = 1`) und konfigurieren (`[FE][tidy_option] = …`). Außerdem müssen Sie dort den Pfad auf dem Server angeben (`[FE][tidy_path] = ...`).

Die im Install-Tool vorbelegten Parameter haben folgende Bedeutung:

-i	Rückt den Quellcode ein.
-quiet true	Es werden keine Meldungen durch Tidy ausgegeben.
-tidy-mark true	Es wird ein Metaelement eingesetzt, das auf die Reinigung durch Tidy hinweist.
-wrap 0	Tidy bricht zu lange Zeilen nach einer hier anzugeben Anzahl Zeichen um. Mit »0« wird dies abgeschaltet.
-raw	Gibt Zeichen oberhalb von 127 aus ohne sie in Entities umzuwandeln.

Tabelle 12.5 Parameter im Install-Tool

Eine detaillierte Liste aller Tidy-Optionen finden Sie auf der CD-ROM oder unter **http://tidy.sourceforge.net/docs/quickref.html (Linkcode 065)**. Wenn Sie Tidy installiert haben, können Sie mit `tidy -h` und `tidy -help-config` alle verfügbaren Konfigurationsparameter einsehen.

12.9 Accessible Content

Die vorhanden Rendering-Funktionen von TYPO3 sind (noch) nicht für barrierefreie Ausgabe optimiert. Obwohl prinzipiell eine barrierefreier Quellcode möglich ist, bedeutet es doch an vielen Stellen erhebliche Mühe und Detailarbeit. Wir haben daher eine Extension geschrieben, die uns und Ihnen diese Arbeit erleichtern soll.

»Accessible Content« bietet neue Rendering-Funktionen für die meistverwendeten Inhaltstypen und Content-Objekt-Typen. Durch den Einsatz dieser Extension und des mitgelieferten statischen Templates ist es möglich, bestehende TYPO3-Projekte zu barrierefreien Webseiten umzubauen und

neue, barrierefreie Webseiten aufzubauen. Jedoch ist diese Extension dabei nur eine Unterstützung – Barrierefreiheit lässt sich nicht auf rein technische Hilfen reduzieren!

»Accessible Content« (Key: *sb_accessiblecontent*) ist von der Systemextension »CSS Styled Content« abgewandelt und soll diese für barrierefreie Webseiten ersetzen. Sie stellt ein eigenes *static template* zur Verfügung.

12.9.1 Exkurs: Rendering von Content-Elementen

Das Rendering von Inhaltselementen wird im wesentlichen von der Klasse `tslib_cObj` vorgenommen. Der Administrator hat über das TypoScript-Template umfangreiche Möglichkeiten, die Frontendausgabe zu verändern. Diese Möglichkeiten sind jedoch begrenzt. Soll beispielsweise der Inhaltstyp »Text mit Bild« statt mit Tabellen CSS-basiert ausgegeben werden, so ist dies nicht über das Template einstellbar.

Inhaltstelemente werden in der Tabelle `tt_content` abgelegt. Die Spalte `CType` gibt an, um welchen Inhaltstyp es sich handelt. Die eigentliche Zusammenstellung der Typen findet im TypoScript-Setup unter `tt_content` statt. Quelle dieser Definition ist meist das verwendete *static template*.

Ein Inhaltstyp wird aus mindestens einem Content-Objekt, in den meisten Fällen einem COA (*content object array*), zusammengesetzt. Nachfolgend ist dies am Beispiel des Typs `textpic` aus dem *static template CSS Styled Content* beschrieben:

```
1:  tt_content.textpic = COA
2:  tt_content.textpic {
3:     10 = COA
4:     10.if.value = 25
5:     10.if.isLessThan.field = imageorient
6:     10.10 = < lib.stdheader
7:     20  = < tt_content.image.20
8:     20.text.10 = COA
9:     20.text.10 {
10:       if.value = 24
11:       if.isGreaterThan.field = imageorient
12:       10 = < lib.stdheader
13:       10.stdWrap.dataWrap = <div class="csc-textpicHeader
          csc-textpicHeader-{field:imageorient}">|</div>
14:    }
```

```
15:     20.text.20 = < tt_content.text.20
16: }
```

Listing 12.4 Rendering einesElementes »Bild mit Text« (textpic)

Die Funktionsweise im einzelnen:

- In `tt_content.textpic.10` und `tt_content.textpic.20.10` wird das Feld `imageorient` aus `tt_content` abgefragt. Je nachdem, ob es kleiner oder größer gleich 25 ist, wird der Standard-Header in den einen oder anderen Container gelegt. Dieses Feld wird in der Eingabemaske für das Inhaltselement im Feld »Position« gesetzt. Die Verknüpfung zwischen dem Zahlenwert und der wählbaren Position ist im `$TCA` eingestellt.
- In `tt.content.20` wird das Bild abgelegt, es wird dafür der Eintrag aus `tt_content.image.20` kopiert.
- Im Bereich `text` von `tt_content.20` wird dann der Text abgelegt, auch hier handelt es sich um eine Kopie eines Teils eines anderen Inhaltstypen, nämlich `text`. In diesem Inhaltstypen wird auf den Content-Objekt-Typen `TEXT` zurückgegriffen.

Für jedes Content Objekt gibt es in der Klasse `tslib_cObj` eine eigene Rendering-Funktion. Im Falle von `HRULER` sieht die Funktion beispielsweise so aus:

```
1:  function HRULER ($conf) {
2:      $lineThickness = t3lib_div::intInRange($this->stdWrap
        ($conf['lineThickness'],$conf['lineThickness.']),1,50);
3:      $lineColor=$conf['lineColor']?$conf['lineColor']:'black';
4:      $spaceBefore = intval($conf['spaceLeft']);
5:      $spaceAfter = intval($conf['spaceRight']);
6:      $tableWidth = $conf['tableWidth'] ? $conf['tableWidth'] :
        '99%';
7:      $content='';
8:      $content.='<table border="0" cellspacing="0" cellpadding="0"
        width="'.htmlspecialchars($tableWidth).'"><tr>';
9:      if ($spaceBefore) {$content.='<td width="1"><img src="'.
        $GLOBALS['TSFE']->absRefPrefix.'clear.gif" width=
        "'.$spaceBefore.'" height="1" alt="" title="" /></td>'; }
10:     $content.='<td bgcolor="'.$lineColor.'"><img src="'.
        $GLOBALS['TSFE']->absRefPrefix.'clear.gif" width="1"
        height="'.$lineThickness.'" alt="" title="" /></td>';
```

```
11:    if ($spaceAfter)  {$content.='<td width="1"><img src="'.
       $GLOBALS['TSFE']->absRefPrefix.'clear.gif" width=
       "'.$spaceAfter.'" height="1" alt="" title="" /></td>'; }
12:    $content.='</tr></table>';
13:    $content = $this->stdWrap($content, $conf['stdWrap.']);
14: return $content;
15: }
```

Listing 12.5 Rendering-Funktion für HRULER

In `$conf` werden das TypoScript für das aktuelle Element übergeben, hier sind es `width`, `height`, `wrap` und `stdWrap`. Ohne auf die Details dieser Funktion einzugehen, so ist doch klar zu erkennen, dass sie nicht – wie man es von einem `HRULER`, also einer horizontalen Linie, erwarten würde, mit einem `<hr>`-Tag arbeitet, sondern stattdessen auf Tabellenkonstrukte zurückgreift.

12.9.2 Neues Rendering erstellen

Im vorangegangenen Abschnitt wurde das Rendering einzelner Inhaltselemente erklärt. Das Beispiel der Funktion `HRULER()` macht deutlich, dass viele der HTML-Codes, die in der Ausgabe erscheinen, hartcodiert in der Klasse `tslib_cObj` vorhanden und daher über TypoScript nicht verändert werden können.

Deshalb ist es notwendig, diese Funktionen durch eigene auszutauschen. Das kann auf zwei Wegen erfolgen:

- Über eine XCLASS von `tslib_cObj`. Leider der zurzeit einzige Weg, die Rendering-Funktionen direkt auszutauschen, da ein Hook fehlt. (Zu XCLASS finden Sie mehr Informationen im Abschnitt 13.10.4).
- Über die Umgestaltung des TypoScript-Setups für Inhaltstypen – die Standard-Rendering-Funktionen werden nicht verwendet, stattdessen werden User-Funktionen (Content-Objekt-Typ `USER`) verwendet (dazu mehr im Abschnitt 13.6.4).

In »Accessible Content« werden vorerst beide Varianten benutzt. Dies hat historische Gründe: da »Accessible Content« aus »CSS Styled Content« hervorgegangen ist und dort einige Inhaltstypen durch User-Funktionen ausgetauscht wurden, sollten diese vorerst beibehalten werden. Für unsererseits hinzugefügte Rendering-Funktionen verwendeten wir aber lieber das XCLASS-Konzept, da wir direkt die Content-Objekte ändern wollten.

Letztendlich ist eine direkte Änderung der Content-Objekte (und damit die erstgenannte Variante) der von uns bevorzugte Weg, da das Konzept der Con-

tent-Objekte es auf einfachem und übersichtlichem Wege erlaubt, Frontend-Ausgaben baukastenartig zusammenzubauen, statt für jeden Inhaltstypen User-Funktionen zu schreiben. Zudem greifen viele Extensions auf die Content-Objekte zurück, so dass eine Änderung der Inhaltselement-Ausgabe nicht ausreichen könnte, um eine barrierefreie Webseite zu erstellen.

Zukünftig wird »Accessible Content« den Weg der XCLASS weiterverfolgen. Wo es notwendig ist, werden Content-Objekte ausgetauscht. Sollte es in späteren Varianten einen Hook geben, der es möglich machen, direkte Austauschfunktionen für die Content-Objekt-Renderingfunktionen anzubieten, so wird die Extension entsprechend umgestellt.

12.9.3 Die XCLASS

»Accessible Content« ändert in der XCLASS `ux_tslib_cObj` die in der Tabelle genannten Content Objekte.

Content Objekt	Änderungen
`FORM`	`<textarea>` und `<input>` erhalten einen Default-Inhalt, der per JavaScript weggenommen wird, wenn der Besucher das Feld anklickt. Es wird ein neuer Feldtyp `<fieldset>` definiert.

Tabelle 12.6 Änderungen an Content-Objekten in der XCLASS ux_tslib_cObj

Wie »CSS Styled Content« auch, stellt »Accessible Content« eine Pluginklasse mit diversen User-Funktionen bereit. Diese User-Funktionen dienen dem von den Content Objekten entkoppelten, speziell auf das Inhaltselement zugeschnittenen, Rendering.

Inhaltstyp	User-Funktion
`textpic`	`tx_sbaccessiblecontent_pi1->render_textpic()`
`bullets`	`tx_sbaccessiblecontent_pi1->render_bullets()`
`table`	`tx_sbaccessiblecontent_pi1->render_table()`
`uploads`	`tx_sbaccessiblecontent_pi1->render_uploads()`

Tabelle 12.7 User-Funktionen der Plugin-Klasse in »Accessible Content«

Alle Methoden geben barrierefreien Quelltext aus.

12.9.4 Tabellen

Eine barrierefreie Tabelle benötigt eine Zusammenfassung (*Summary*), eine Tabellenüberschrift (*Caption*) sowie Kopfzeilen (<th> statt <td>), auf die sich die sonstigen Zellen beziehen. Nachfolgend ist der HTML-Quelltext einer barrierefreien Tabelle schematisch dargestellt:

```
1:  <table summary="Zusammenfassung">
2:  <caption>Dies ist eine Beispieltabelle</caption>
3:     <tr>
4:        <th scope="col" id="col0">Überschrift a</th>
5:        <th scope="col" id="col1">Überschrift b</th>
6:     </tr>
7:     <tr>
8:        <td headers="col0">Inhalt 1</td>
9:        <td headers="col1">Inhalt 2</td>
10:    </tr>
11: </table>
```

Listing 12.6 Barrierefreie Tabelle

Folgende Unterschiede bestehen zu einer normalen mit TYPO3 erstellbaren Tabelle:

- Der <table>-Tag erhält ein Attribut `summary`, in dem eine kurze Zusammenfassung des Tabelleninhaltes steht.
- Unterhalb des <table>-Tags und überhalb der ersten Zeile erscheint ein <caption>-Tag, der die Tabellenüberschrift beinhaltet
- Es gibt Kopfzeilen, die mittels <th>-Tag gesetzt werden. Ob sie sich auf eine Spalte oder eine Zeile definieren, wird in ihrem Attribut `scope` definiert. Sie erhalten ausserdem ein `id`-Attribut.
- Alle Inhaltszellen, die den <td>-Tag, erhalten ein `headers`-Attribut, das sich auf die id des <th>-Tags bezieht.

Die Standardversion des Table Wizard von TYPO3, die in Abbildung 12.13 dargestellt ist, enthält all diese Möglichkeiten nicht. Daher wurde für »Accessible Content« ein eigener, von der Standardvariante abgeleiteter Editor geschrieben (Abbildung 12.14). Ist die Extension installiert, so erhält man beim Klick auf das Wizard Icon den neuen *Table Wizard*.

Legt man ein Inhaltselement vom Typ »Tabelle« an, so erhält man nach Eingabe der Überschrift und erstem Speichern das Wizard Icon, das einen Link auf den *Table Wizard* enthält. Dies verdeutlicht Abbildung 12.12.

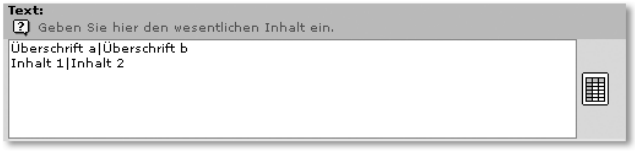


Abbildung 12.12 Inhalt der Tabelle und Wizard Icon

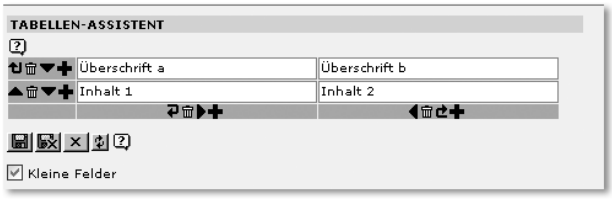


Abbildung 12.13 Der Table Wizard von TYPO3

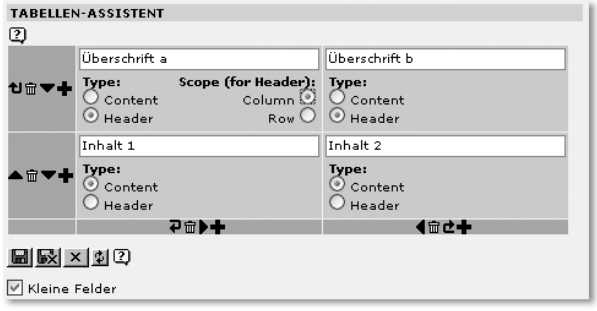


Abbildung 12.14 Ein neuer Table Wizard von Accessible Content

Wie Abbildung 12.12 zeigt, werden die Tabelleninhalte im Textformat gespeichert. Dabei ist | das Trennzeichen zwischen den Spalten und ein Zeilenumbruch das Trennzeichen zwischen den Zeilen.

Der neue *Table Wizard* hat die Möglichkeit, zusätzliche Zellinformationen zu speichern. Es kann zwischen »Content« (`<td>`) und »Header« (`<th>`) unterschieden und bei solchen Tabellenköpfen, wo die Zuordnung nicht eindeutig ist, der Scope bestimmt werden. Diese Zusatzinformationen werden mittels eines neuen Trennzeichens, der Tilde ~, abgespeichert (siehe Abbildung 12.15):

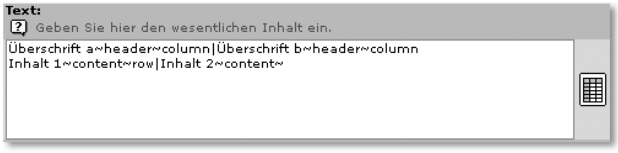


Abbildung 12.15 Neues Speicherformat der barrierefreien Tabellen

Jede Zelle trägt nun die zusätzlichen Informationen. Um diese auch ausgabeseitig zu berücksichtigen, wurde eine User-Funktion für das Rendering geschrieben. Im TypoScript wird dafür das Rendering für das entsprechende Content Object geändert (gekürzt):

```
tt_content.table = COA
tt_content.table {
   10 = < lib.stdheader
   20 = USER
   20.userFunc = tx_sbaccessiblecontent_pi1->render_table
   20.innerStdWrap.parseFunc = < lib.parseFunc
}
```

Somit ist es nun möglich, barrierefreie Tabellen zu erstellen. Genauere Informationen darüber, wie ein Wizard ausgetauscht werden kann, finden Sie im Abschnitt 13.4.4, *$TLA-Betrieb: Wizard austauschen*.

12.9.5 Formulare

Die Umgestaltung der Formulare gestaltete sich ähnlich wie diejenige der Tabellen. Auch hier wurde der Wizard ausgetauscht, um zusätzliche Informationen eingeben zu können. Eingabeseitig war hier im Wesentlichen dafür Sorge zu tragen, dass es möglich ist, das Formular in mittels `<fieldset>` gesetzte Sektionen zu unterteilen. Dazu wurde im Form Wizard ein neuer Elementtyp »Fieldset« angelegt. Ein gesetztes Fieldset wird entweder am Ende des Formulares oder beim Öffnen eines neuen Fieldsets geschlossen.

Abbildung 12.16 zeigt einen beispielhaften Eintrag im neuen Formular-Wizard. Der resultierende Quelltext ist nachfolgend dargestellt:

```
1:  <form action="index.php?id=6" name="8e6"
    enctype="multipart/form-data" method="post" target="page"
    onsubmit="return validateForm('8e68dd22fa267c12b607c6602d5e3b
    1c','name,Name%3A,email,e-Mail','','','')"><input
    type="hidden" name="locationData" value="6:tt_content:12" />
2:     <fieldset>
3:        <legend>Kontaktdaten</legend>
4:        <label for="name">Name:</label>
5:        <input type="text" name="name" id="name" size="40"
          onfocus="if(this.value=='Ihr Name')this.value='';"
          onblur="if(this.value=='')this.value='Ihr Name';"
          value="Ihr Name" />
6:        <label for="email">e-Mail</label>
```

```
7:        <input type="text" name="email" id="email" size="40"
          onfocus="if(this.value=='Ihre Mailadresse')this.
          value='';" onblur="if(this.value=='')this.value=
          'Ihre Mailadresse';" value="Ihre Mailadresse" />
8:     </fieldset>
9:     <fieldset>
10:    <legend>Eintrag</legend>
11:    <label for="comment">Ihr Kommentar</label>
12:    <textarea name="comment" id="comment" cols="30" rows="10"
       wrap="virtual" onfocus="if(this.value=='Kommentar')this.
       value='';" onblur="if(this.value=='')this.value=
       'Kommentar';">Kommentar</textarea>
13: </fieldset>
14: <fieldset>
15:    <legend>Absenden</legend>
16:    <label for="formtype_mail"></label>
17:    <input type="submit" name="formtype_mail"
       id="formtype_mail" value="Abschicken" />
18: </fieldset>
19: </form>
```

Listing 12.7 Formular über den verbesserten Formular-Wizard

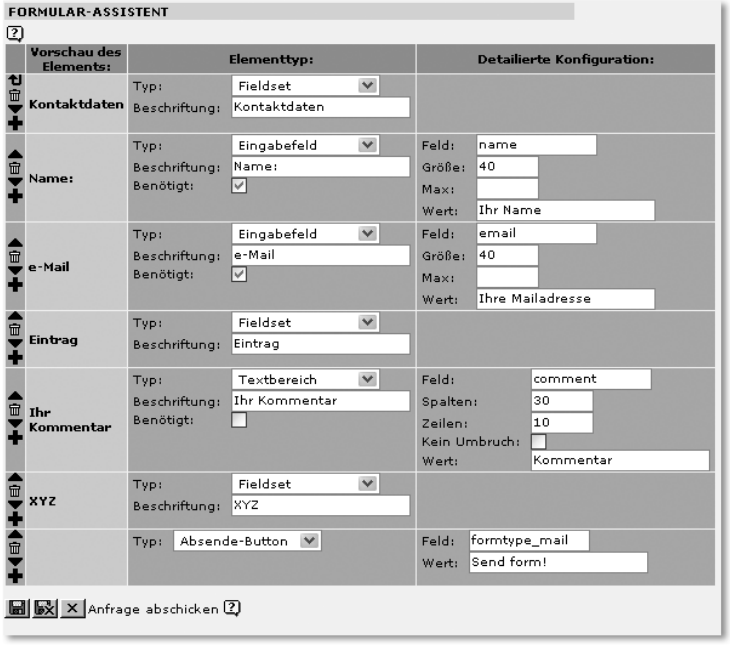


Abbildung 12.16 Eingabe eines Formulareintrags im neuen Formular-Wizard

Wie zu sehen, ist das Formular entsprechend der Vorgaben in `<fieldset>`-Sektionen unterteilt. Die Beschriftungen aller Eingabefelder sind in einem `<label>`-Tag verpackt, der in seinem `for`-Attribut angibt, auf welches Formularfeld er sich bezieht. Zusätzlich sind textbasierte Eingabefelder bereits mit einem Eingabewert vorgefüllt. Beim Anklicken dieser Eingabefelder sorgt ein JavaScript dafür, dass der Inhalt gelöscht wird. Verlässt man ein Eingabefeld und hat nichts eingetragen, so trägt das JavaScript den Vorgabewert wieder ein.

Anders als bei den Tabellen wurde die für Formulare zuständige Methode `FORM()` in der XCLASS umgeschrieben. Zukünftig ist es geplant, stattdessen auf eine User-Funktion zurückzugreifen.

12.9.6 Probleme mit Plugins

Im Gegensatz zu »CSS Styled Content« arbeitet »Accessible Content« zur Zeit als reine Extension, für die keine expliziten Core-Änderungen vorliegen. Das führt zu einem Problem mit der Frontendausgabe von Plugins, für das wir vorerst keine elegante Lösung anbieten können.

Fügt man ein Plugin, beispielsweise die »New front end login box« (newloginbox) ein, so erscheint dieses in der Frontendausgabe nicht. Verantwortlich dafür ist der in vielen Extensions verwendete Befehl `t3lib_extMgm::addPItoST43()`, der nur für die *Static Templates* »Content (default)« und »CSS Styled Content« funktioniert.

Abhilfe schafft hier leider nur, die von diesem Befehl getätigten Eintragungen im TypoScript für jede einzelne Extension vorzunehmen; hier ein Beispiel für die »New front end login box«:

```
tt_content.list.20.newloginbox_pi1 = < plugin.tx_newloginbox_pi1
tt_content.list.20.newloginbox_pi3 = < plugin.tx_newloginbox_pi3
```

Die entsprechenden Eintragungen können aus den Extension-Dateien **ext_tables.php** oder **ext_localconf.php** abgelesen werden. Diese sehen für die newloginbox wie folgt aus:

```
t3lib_extMgm::addPItoST43($_EXTKEY,'pi1/class.tx_newloginbox_
pi1.php','_pi1','list_type',0);
t3lib_extMgm::addPItoST43($_EXTKEY,'pi3/class.tx_newloginbox_
pi3.php','_pi3','list_type',0);
```

Wichtig sind der erste, zweite und dritte Parameter, die sich in TypoScript wie folgt wiederfinden:

```
tt_content.list.20.[Parameter 1][Parameter 3] = <
plugin.[Parameter 2 (nur Klassenname)]
```

Für einige Plugins haben wir die Einträge bereits im *Static Template* vorgenommen.

Sonderfall Indexed Search

Die Frontend-Ausgabe der beliebten Extension »Indexed Search Engine« (*indexed_search*) ist alles andere als barrierefrei. Daher haben wir auch für diese Extension neue Ausgabefunktionen geschrieben. Leider ist bei »Indexed Search Engine« die komplette Ausgabe direkt in der Plugin-Klasse realisiert – es stehen keine Templates wie z.B. für »News« zur Verfügung. In der aktuellen Version von »Accessible Content« haben wir daher Anpassungen in einer Klassendatei **class.ux_tx_indexedsearch.php** untergebracht, die in `ext_localconf.php` die Original-Klasse ersetzt.

In der mit TYPO3 3.8 mitgelieferten Version der Indexsuche wurden einige Funktionen verändert, so dass unsere Klasse leider nicht mehr funktioniert. Wir werden auf der Buchseite und unter **http://typo3.sunbeam-berlin.de/accessiblecontent/ (Linkcode 066)** eine neue Version unserer Klassenbibliothek bereitstellen.

So lange können Sie – wenn Sie »Accessible Content« auf einer 3.8er-Version einsetzen – die Modifikation von »Indexed Search Engine« deaktivieren, indem Sie in der Datei **ext_localconf.php** (im Verzeichnis der Extension »Accessible Content«) die folgende Zeile auskommentieren:

```
$TYPO3_CONF_VARS[TYPO3_MODE]["XCLASS"]['ext/indexed_search/
pi/class.tx_indexedsearch.php'] = t3lib_extMgm::extPath($_EXT-
KEY)."class.ux_tx_indexedsearch.php";
```

12.9.7 Weitere Entwicklung von »Accessible Content«

Die Arbeit an der Extension ist weit davon entfernt, abgeschlossen zu sein – vielleicht haben ja Sie auch den ein oder anderen Hinweis?

Wir werden die Arbeit an der Extension fortsetzen und die aktuelle Entwicklung auf der Website zum Buch und unter folgendem Link dokumentieren:

http://typo3.sunbeam-berlin.de/accessiblecontent/ (Linkcode 066)

Index

D

T