



Dokumentation

Inhalt

Inhalt.....	2
1 Einleitung.....	6
1.1 Was ist YAML?	6
1.2 Was ist YAML nicht?	7
1.3 Vorteile des Frameworks.....	7
1.3.1 Weiterentwicklung / Updates	8
1.4 Barrierefreiheit & Webstandards.....	8
1.5 Der Aufbau des Download-Pakets.....	11
1.5.1 Das Download-Paket	11
1.5.2 Die Dateien des Frameworks.....	11
1.5.3 Mitgelieferte Layoutbeispiele	13
1.5.4 Hilfsmittel zur Layoutentwicklung.....	14
1.6 Unterstützte Browser	15
1.7 IE 5/Mac, Netscape 4 & Co.....	15
1.8 Danksagung	16
2 Grundlagen	17
2.1 Ganzheitliches Konzept	17
2.2 Die Grundlage: <i>floats</i>	17
2.3 Markupfreies Clearing	18
2.3.1 Methode 1: Clearfix.....	18
2.3.2 Methode 2: Overflow	19
2.3.3 Warum zwei Clearing-Methoden?	19
2.4 Der Aufbau des XHTML-Quelltextes.....	19
2.4.1 Wahl des Doctypes	20
2.4.2 Die Struktur im Detail	20
2.4.3 Gestaltungsfreiheit durch das Kombinationsmodell.....	21
2.5 Reihenfolge der Spalten-Container im Quelltext	22
2.6 Die Funktionsweise von floats im Detail	23
2.6.1 Vorbereitung des Layouts.....	24
2.6.2 Aufbereitung der Inhalte	25
2.7 Das Clearing der Spalte #col3	26
2.7.1 Globales Clearing macht #col3 zur längsten Spalte.....	27
2.7.2 Eine spezielle Clearing-Lösung für den Internet Explorer	27

2.7.3	Grafikfreie Spaltentrenner und Spaltenhintergründe	29
2.8	Skip-Link-Navigation	30
2.8.1	Skip-Link-Navigation im YAML-Framework	30
3	CSS-Bausteine	32
3.1	Das CSS-Konzept	32
3.1.1	Einsatz der Kaskade	32
3.2	Namenskonventionen	33
3.2.1	Grundbausteine (core-Dateien)	33
3.2.2	Ergänzungsbausteine	33
3.2.3	Patches	33
3.2.4	Dateivorlagen	33
3.3	Das zentrale Stylesheet	34
3.3.1	Einbindung & Import der CSS-Bausteine	34
3.4	Das Basis-Stylesheet base.css	36
3.4.1	Browser Reset – Einheitliche Ausgangsbedingungen für alle Browser	36
3.4.2	Rohbau des Layouts	39
3.4.3	Weitere Bestandteile	40
3.5	CSS-Anpassungen für den Internet Explorer	42
3.5.1	Aufbau der CSS-Anpassungsdatei für den Internet Explorer	43
3.5.2	Struktur- und layoutunabhängige Bugfixes	44
3.5.3	Struktur- und layoutabhängige Bugfixes	49
3.6	Erstellung des Screenlayouts	52
3.6.1	Bestandteile des Screenlayouts	52
3.6.2	Gestaltung der Layoutelemente	52
3.6.3	Gestaltung der Navigationselemente und des Inhalts	53
3.6.4	Das Zusammensetzen des Layouts	54
3.7	Navigationsbausteine	54
3.7.1	Sliding Door Navigation	54
3.7.2	Shiny Buttons Navigation	55
3.7.3	Vertikale Listennavigation	56
3.8	Gestaltung der Inhalte	58
3.8.1	Die Vorlage content_default.css	58
3.9	Anpassung des Layouts für Printmedien	62
3.9.1	Vorbereitungen für den Ausdruck	62
3.9.2	Aufbau der Print-Stylesheets	63

3.9.3	Allgemeine Druckvorbereitung über <code>print_base.css</code>	64
4	Anwendung	68
4.1	Fünf Regeln	68
4.1.1	Mitgelieferte Beispiele	68
4.1.2	Tipps für CSS-Einsteiger	69
4.2	Empfehlung für die Projektstruktur	69
4.2.1	Dateien und Verzeichnisse anlegen	69
4.2.2	Anpassung der Dateipfade	69
4.2.3	Gestaltung des Layouts	70
4.3	Grundlegende Variationsmöglichkeiten.....	71
4.3.1	3-Spalten-Layouts.....	72
4.3.2	2-Spalten-Layouts.....	73
4.3.3	Weitere Alternativen zur Anordnung der Container.....	74
4.4	Freie Anordnung und Verwendung der Content-Spalten	75
4.4.1	Spalten in beliebiger Reihenfolge.....	75
4.4.2	Spaltenanordnung 1-3-2 und 2-3-1	77
4.4.3	Spaltenanordnung 1-2-3 und 3-2-1	78
4.4.4	Spaltenanordnung 2-1-3 und 3-1-2	79
4.4.5	Fazit	81
4.5	Subtemplates.....	81
4.5.1	Struktureller Aufbau	81
4.5.2	Anpassungen der Subtemplates für den Internet Explorer	85
4.5.3	Beispiele für die Anwendung von Subtemplates	86
4.5.4	Alternatives Layoutkonzept.....	87
4.6	Gestaltung der Spalten.....	87
4.6.1	Beispiel 1 - Spaltentrennlinien	88
4.6.2	Beispiel 2 - Spaltenhintergründe	89
4.7	Minimale & Maximale Breiten für flexible Layouts.....	89
4.7.1	Fehlende CSS-Unterstützung im Internet Explorer 5.x und 6.0	90
4.7.2	Lösung 1: IE Expressions.....	90
4.7.3	Lösung 2: Externes Javascript »minmax.js«	91
4.8	Entwurf & Fehlersuche.....	92
4.8.1	Kontroll-Automatik für <i>ie hacks.css</i>	92
4.8.2	Pixelraster für Positions- und Geometriekontrolle	93
4.8.3	Hervorhebungen von Elementen	94

4.9	Ausgewählte Anwendungsbeispiele.....	94
4.9.1	Das Screenlayout der Layoutbeispiele	94
4.9.2	Layoutentwurf "2col_left_seo"	97
4.9.3	Layoutentwurf "3col_fixed_seo"	99
4.9.4	Layoutentwurf "Flexible Grids"	102
5	Hinweise	106
5.1	Hilfsmittel	106
5.1.1	Dynamisch generierte Blindtexte	106
5.1.2	Bearbeitungshilfen für Dreamweaver	106
5.2	Tipps zum Entwurf flexibler Layouts	107
5.2.1	Umgang mit großen Elementen	107
5.2.2	Kleine Bildschirmgrößen.....	107
5.2.3	Flexible Randspalten.....	108
5.3	Bekannte Probleme	108
5.3.1	Internet-Explorer 5.x: Kollabierender Margin bei #col3.....	108
5.3.2	Mozilla & Firefox.....	109
5.3.3	Netscape.....	109
5.3.4	Opera.....	110
6	Changelog.....	111
6.1	Changelog 3.x	111
6.1.1	Änderungen in Version 3.0.3 [18.08.07]	111
6.1.2	Änderungen in Version 3.0.2 [01.08.07]	112
6.1.3	Änderungen in Version 3.0.1 [15.07.07]	113
6.1.4	Änderungen in Version 3.0 [09.07.07]	113

1 Einleitung

1.1 Was ist YAML?

YAML wurde als Basis für die Entwicklung flexibler Layouts entworfen. Einer der Schwerpunkte liegt damit bei den Anforderungen, die sich aus der Arbeit mit variablen Größenangaben ergeben. Daraus entwickelte sich ein Basis-Layout, welches folgende grundlegende Funktionen bietet:

Die wichtigsten Features:

- Flexibles, auf Barrierefreiheit ausgelegtes Basislayout mit Kopf- und Fußbereich sowie dem Inhaltsbereich mit ein bis drei Spalten,
- Browserübergreifend einheitliches Erscheinungsbild des Layouts,
- Größtmögliche Gestaltungsfreiheit für den Webdesigner (fixe/flexible Layouts, variable Spaltenbreiten usw.),
- Beliebige Reihenfolge der Spalteninhalte im Quelltext (Stichwort: *any order columns*),
- Funktional gegliederte Stylesheet-Vorlagen
- Spaltentrennlinien und Spaltenhintergründe können ohne Grafikeinsatz erzeugt werden und laufen immer bis zum Fußbereich,
- Flexible Raumaufteilung innerhalb der einzelnen Container über Subtemplates.

Mit dem vorhandenen Basislayout lassen sich Designs mit ein bis drei Spalten (mit fixen oder flexiblen Breiten) innerhalb sehr kurzer Zeit entwickeln. Über *Subtemplates* (flexible Raster) lässt sich das Spaltensystem nahezu beliebig erweitern und verschachteln. Erweitert wird das YAML-Basislayout durch spezielle Container, welche die Festlegung der Layoutbreite vereinfachen und z.B. grafische Umrandungen des Layouts ermöglichen. Wozu all diese vordefinierten Container?

Generell lassen sich zwei grundsätzliche Arbeitsweisen bei der Erstellung eines Layouts unterscheiden:

Das Bottom-Up-Prinzip

Der Seitenersteller beginnt mit dem Bau des Layouts quasi bei Null. Die im Layout benötigten Container werden erstellt und per CSS positioniert und gestaltet. Ein Basislayout ist in diesem Fall also nicht vorhanden. Während der Erstellung des Layouts müssen alle ggf. auftretenden Browserbugs gefunden und in der Regel individuell beseitigt bzw. umgangen werden.

Das Top-Down-Prinzip

In diesem Fall steht dem Seitenersteller ein browserübergreifend funktionstüchtiges, modular aufgebautes Basislayout zur Verfügung, welches alle häufig benötigten Layoutelemente enthält. Der Webdesigner modifiziert dieses Basislayout nach seinen Vorstellungen und optimiert anschließend den XHTML- und CSS-Code, indem er nicht benötigte Elemente aus dem Layout entfernt.

YAML wurde für die Arbeit nach dem *Top-Down-Prinzip* entworfen. Die Begriffe »Baukasten-System« oder »Framework« bezeichnen YAML daher wohl am treffendsten.

1.2 Was ist YAML nicht?

YAML ist *kein* Fertiglayout. Dies zu behaupten, widerspräche dem Entwurfsgedanken des *Top-Down-Prinzips*. Ohne eine Optimierung auf die jeweiligen Layoutanforderungen erzeugen nicht benötigte Elemente (HTML/CSS) unnötigen Ballast.

Anmerkung des Autors: Das YAML-Framework stellt als technische Grundlage ein browserübergreifendes Basislayout sowie zahlreiche hilfreiche CSS-Bausteine zur Verfügung, so dass dem Webdesigner wieder mehr Zeit und Raum für die kreative Gestaltung bleibt.

Nichts liegt mir ferner, als die monotone Optik von Reihenhäusern im Bereich des Webdesigns zu fördern, indem YAML als Fertiglayout angepriesen wird.

Natürlich ist es nicht verboten, YAML samt seiner zusätzlichen Bausteine als »ready to use« Layout einzusetzen. Bei der Verwendung sollten Sie jedoch neben der individuellen Gestaltung des Webauftritts immer auch dem Anspruch folgen, möglichst einfachen und schlanken Code zu erzeugen. Wartung und Fehlersuche im Code werden dadurch stark vereinfacht. Nicht benötigte Elemente im XHTML-Quelltext oder den CSS-Bausteinen sollten daher - nachdem das Layout steht - entfernt werden.

1.3 Vorteile des Frameworks

YAML ist mehr als ein einfaches mehrspaltiges Layout. Es ist ein praxisorientiertes Layout-Framework mit einem hohen Grad an Flexibilität. YAML stellt verschiedenste Bausteine für die Layouterstellung bereit und sorgt für deren reibungsloses Zusammenspiel. Daraus ergeben sich folgende Vorteile:

Browserkompatibilität

Die Grundbausteine von YAML garantieren eine browserübergreifend korrekte Darstellung des Layouts. Alle für die fehlerfreie Darstellung des Layouts erforderlichen Browser-Anpassungen sind in den Grundbausteinen bereits integriert. Der während der Layouterstellung üblicherweise erforderliche Zeitaufwand für umfangreiche Kompatibilitätstests zu verschiedensten Browsern kann auf ein Minimum reduziert werden.

Baukasten-Prinzip

Das Baukasten-Prinzip ermöglicht eine besonders effiziente Nutzung des vorhandenen Codes bei der Layouterstellung.

Mit den Grundbausteinen wird ein Basislayout mit voller Funktionalität bereitgestellt. Durch zusätzliche Bausteine kann diese Basis ergänzt oder modifiziert werden. Diese CSS-Bausteine sind universell einsetzbar. Einmal geschrieben und getestet, können sie bei Bedarf eingebunden werden und stehen für zukünftige Projekte zum Einsatz bereit.

Zwei Beispiele dafür sind einfache Variationsmöglichkeiten eines Screenlayouts mittels der *basemod*-Dateien oder auch die vorgefertigten Druck-Stylesheets.

Flexibilität in der Layoutgestaltung

Die Gestaltungsmöglichkeiten des Frameworks gehen weit über ein einfaches 3-Spalten-Layout hinaus. Die flexible Basis ermöglicht eine beliebige Anordnung der Inhaltsspalten am Bildschirm. Der dynamische Charakter der verwendeten *float*-Umgebungen ermöglicht die Modifikation hin zu 1- oder 2-Spalten-Layouts mit nur wenigen Handgriffen. Spalten- und Layoutbreiten können in beliebigen Maßeinheiten definiert werden. Die Mischung unterschiedlicher Einheiten bei den Spaltenbreiten ist dabei problemlos möglich.

Robuster Code

Der Aufbau der XHTML- und CSS-Struktur der Grundbausteine garantiert eine fast vollständige Unabhängigkeit vom Aufbau der später eingefügten Inhalte. Die Kapselung der Hauptelemente der Webseite in separaten DIV-Containern sichert die korrekte Positionierung der Elemente am Bildschirm, unabhängig von der Art der späteren Nutzung der Container.

1.3.1 Weiterentwicklung / Updates

Das YAML-Framework wird beständig weiterentwickelt. Alle Änderungen/Erweiterungen jeder neuen Version werden im [Changelog](#) zusammengefasst und ggf. auf weiterführende Informationen innerhalb der Dokumentation verwiesen.

Im Rahmen der Um- oder Neugestaltungen YAML-basierter Websites - oder falls die erweiterte Funktionalität neuerer YAML-Version innerhalb bestehender Seiten benötigt werden, sind Updates der Framework-Basis dank der funktionellen Gliederung der CSS-Bausteine jederzeit möglich. Die Trennung von YAML- und Nutzer-CSS erlaubt im Idealfall ein unproblematisches Update der Framework-Version.

Wichtig: YAML basiert seit langem auf robusten und stabilen Grundbausteinen. Von der prinzipiellen Updatemöglichkeit sollte bei bestehenden Webseiten dennoch nicht zum Selbstzweck Gebrauch gemacht werden. Ein fehlerfrei funktionierendes CSS-Layout benötigt keine monatlichen Sicherheitspatches!

Ein Update der Framework-Basis ist dann empfehlenswert, wenn sich bekannte CSS-Probleme einer bestehenden Webseite durch eine neue YAML-Version beseitigen lassen.

1.4 Barrierefreiheit & Webstandards

Die Diskussionen um die unterschiedliche Bedeutung der Begriffe *barrierefrei* und *barrierearm* möchte ich an dieser Stelle nicht führen. Ebenso wenig reicht der Platz, um die Vorteile von Webstandards erschöpfend zu behandeln. Vielmehr soll hier erläutert werden, was YAML zu den beiden Themengebieten beitragen kann.

Valider XHTML-Code und valide Stylesheets

Ein valides Grundgerüst ist der Anfang einer für alle Personengruppen (behindert oder nicht) zugänglichen Webseite. Die Validität garantiert ein Höchstmaß an Einheitlichkeit in der Darstellung der Webseite in unterschiedlichsten Browsern. Die einzelnen Bestandteile des YAML Frameworks basieren daher auf validem XHTML- und CSS-Code.

Umfangreiche Browserunterstützung

Hinter YAML steht das Ziel, ein browserübergreifend einheitliches Erscheinungsbild einer Webseite zu ermöglichen. Die Problematik der teilweise stark variierenden Unterstützung von CSS-Standards, insbesondere der zahlreichen CSS-Bugs des Internet Explorers ist hinreichend bekannt. Dennoch ist der Internet Explorer der unbestrittene weltweite Marktführer und wird daher umfassend unterstützt. Es ist nicht sinnvoll, ein CSS-Framework allein auf vermeintlich standardkonforme Browser zu optimieren.

Der Internet Explorer hat derzeit weltweit einen geschätzten Marktanteil von knapp 90%. Der Anteil des IE 5.x an dieser Zahl liegt im einstelligen Prozentbereich. Diese Anzahl entspricht immer noch in etwa dem Marktanteil vieler alternativer Browser wie Opera, Mozilla oder Safari. Allein der Firefox konnte in den letzten Jahren die 5-Prozent-Hürde deutlich überspringen. Die Unterstützung des IE 5.x durch YAML ist daher gleichermaßen sinnvoll und gerechtfertigt wie die Unterstützung moderner Browser.

Verzicht auf Layouttabellen

Zu Layouttabellen finden sich im Internet gegensätzliche Meinungen. Während allgemein Einigkeit in dem Punkt herrscht, dass verschachtelte Tabellenlayouts veraltet, alles andere als benutzerfreundlich und nur schwer wartbar sind, wird die Frage - ob Layouttabellen (unverschachtelt !) generell verteufelt werden sollten oder auch heute noch ihre Daseinsberechtigung haben - nach wie vor kontrovers diskutiert. Nachfolgend werden daher einige Vorteile aufgezählt, die sich aus dem Verzicht auf Layouttabellen im YAML-Framework ergeben:

- **Freie Spaltenanordnung**

Die Reihenfolge der Spaltencontainer im Quelltext ist vollkommen unabhängig von der Positionierung der Spalten am Bildschirm. Dadurch kann die Zugänglichkeit der Seiteninhalte für Textbrowser und Screenreader verbessert werden, welche die Seiteninhalte linearisiert darstellen bzw. auslesen. Hinzu kommen die Optimierungsmöglichkeiten der Inhalte für Suchmaschinen.

- **Flexibilität im Layout und in der Druckvorbereitung**

Durch einfaches Abschalten einzelner Spalten per CSS lassen sich neue Layoutvarianten für die Bildschirmdarstellung erzeugen (1- und 2-Spalten-Layouts). Für den Ausdruck der Webseite können innerhalb des Druck-Stylesheets unwichtige Seitenelemente abgeschaltet werden (Navigation, Sidebar, usw.). Des Weiteren können Spaltencontainer per CSS auf einfache Weise linearisiert werden, d.h. sie werden für den Ausdruck auf voller Breite untereinander dargestellt.

- **Rendering-Geschwindigkeit im Browser**

Tabellen werden vom Webbrowser erst gerendert, wenn alle Bestandteile der Tabellen vollständig geladen wurden. Im Gegensatz dazu beginnen Browser bereits nach Erhalt des ersten vollständigen DIV-Containers mit dem Rendering der Webseite. Tabellenlayouts benötigen daher im Vergleich eine längere Wartezeit bis eine Seite im Browser darstellt. Auch in der heutigen Zeit sind Modem- und ISDN-Verbindungen Normalität. Insbesondere hier sind erhöhte Wartezeiten besonders spürbar und lästig.

Ausrichtung auf flexible Größenangaben

Ein weiterer wichtiger Punkt auf dem Weg zu barrierefreien Webseiten ist der Einsatz relativer Maßeinheiten (z.B. bei Layoutbreiten oder Schriftgrößen). Zugänglichkeitsprobleme entstehen nicht nur für behinderte Menschen sondern für jeden von uns, wenn fixe Layouts und fest vorgeschriebene, zu kleine Schriftgrößen das Lesen erschweren oder Webseiten sich nicht vernünftig ausdrucken lassen. Die flexible Ausrichtung aller Gestaltungselemente (Spaltengrößen, Randabstände, Schriftgrößen) war einer der wichtigsten Ansatzpunkte bei der Entwicklung des YAML-Frameworks.

Semantischer Code

Die semantisch korrekte Auszeichnung von Inhalten trägt ebenfalls zu schlankerem Code, zur besseren Übersichtlichkeit in alternativen Browsern und mehr Zukunftssicherheit bei. Das YAML-Framework liefert dabei in erster Linie den gestalterischen Rahmen einer Webseite, welcher *unabhängig* von der Art der späteren Inhalte funktionieren muss. Die Einbeziehung von Inhaltselementen in die Layoutgestaltung, was bei geschicktem Einsatz zu einer Verringerung der im Basislayout vorhandenen DIV-Ebenen führen könnte, kann von YAML nicht vorausgesehen werden. Die Optimierung von XHTML-Markup und der Stylesheets liegt im Ermessen des Webdesigners nach Fertigstellung des Layouts.

Skip-Link-Navigation

Neben der Möglichkeit der freien Spaltenanordnung, die eine optimale Linearisierung der Inhalte in Textbrowsern und Screenreadern ermöglicht, erleichtert die Skip-Link-Navigation vor allem in Screenreadern die Bewegungsfreiheit auf der Webseite, in dem Sprungmarken zu wichtigen Inhaltselementen (z.B. Navigation, Inhaltsbereich) der Webseite definiert werden.

Das YAML-Framework stellt ein flexibel einsetzbares Grundgerüst bereit, welches sich an den Anforderungen für barrierefreies Webdesign orientiert und die Vorteile von Webstandards konsequent nutzt. In diesem Zusammenhang möchte ich, nicht ohne ein klein wenig Stolz, auf das [Redesign 2006](#) der Webseite [Einfach für Alle](#) verweisen. Die Webseite ist eine Initiative der [Aktion Mensch](#) und widmet sich seit vielen Jahren intensiv dem Thema *Barrierefreies Webdesign*. Das aktuelle flexible Mehrspaltenlayout des Jahres 2006 basiert nach Aussagen der Entwickler in großen Teilen auf YAML.

An dieser Stelle möchte ich auf einige weiterführende Literaturquellen im Internet hinweisen. Beide angesprochenen Themengebiete lassen sich hier nur sehr grob umreißen. Daher empfehle ich interessierten Lesern die nachfolgend aufgelisteten Artikel.

Weiterführende Links

- [BITV Reloaded](#)
- ["Nicht unsere Zielgruppe!": Wirtschaftlichkeit von Barrierefreiheit](#)
- [Retro-Coding: Semantischer Code ist der Anfang von gutem Design](#)
- [Semantischer Code - Definitionen, Methoden, Zweifel](#)

1.5 Der Aufbau des Download-Pakets

Im Folgenden soll der Aufbau des Download-Pakets besprochen werden, welches Sie sich auf der [Startseite](#) herunterladen können. Das Download-Paket enthält neben den Dateien des Frameworks eine vollständige Dokumentation, zahlreiche Anwendungsbeispiele sowie einige hilfreiche Werkzeuge zur Layoutentwicklung.

1.5.1 Das Download-Paket

Datei/Verzeichnis	Beschreibung
documentation/	Hier finden Sie die Dokumentation des Frameworks in den Sprachen Deutsch und Englisch im PDF-Format. Es handelt sich dabei um einen vollständigen Abzug der Online-Dokumentation von YAML.de . Lesen Sie die Dokumentation sorgfältig und beachten Sie die hervorgehobenen Hinweise bei der Anwendung des Frameworks.
yaml/	In diesem Verzeichnis befinden sich alle Dateien des Frameworks. Hierbei handelt es sich zum einen um fertige, einsatzbereite CSS-Bausteine als auch um Dateivorlagen für eigentliche Layoutgestaltung. Die Bedeutung der einzelnen Bausteine wird im Rahmen der Dokumentation ausführlich erläutert. Hinweise zum praktischen Einsatz des Frameworks finden Sie in Kapitel 4 .
examples/	Dieses Verzeichnis enthält eine Vielzahl von Anwendungsbeispielen des YAML-Frameworks anhand vollständiger Layoutbeispiele. Die Beispiele sind nach Themenbereichen gegliedert. Im Rahmen der Dokumentation wird auf einige dieser Beispiele vertiefend eingegangen.
tools/	In diesem Verzeichnis befinden sich einige Werkzeuge zur Layoutentwicklung. Die hier abgelegten Dateien sind für die Funktion des Frameworks nicht erforderlich. Sie stellen somit keine festen Bestandteile des YAML-Frameworks dar.

1.5.2 Die Dateien des Frameworks

Das YAML-Framework besteht im Grunde genommen aus einer vordefinierten XHTML-Struktur sowie aus einer Reihe von CSS-Dateien mit unterschiedlichen Funktionen. Diese CSS-Dateien befinden sich im Verzeichnis *yaml*.

Neben den eigentlichen CSS-Bausteinen, finden Sie in diesem Ordner zudem Dateivorlagen (*Drafts*), die Sie für die Anwendung des Frameworks zur Gestaltung eigener Layouts verwenden können. Diese Vorlagen sollen Ihnen in erster Linie den Zugang zum Framework erleichtern und grundlegende Arbeitsschritte vereinfachen.

Datei/Verzeichnis	Beschreibung
/yaml/	
<i>central_draft.css</i> <i>markup_draft.html</i>	<p>Dies ist das Stammverzeichnis des YAML-Frameworks. Es beinhaltet die Vorlagedatei <i>central_draft.css</i> für ein so genanntes <i>zentrales Stylesheet</i> (siehe Abschnitt 3.3).</p> <p>Über ein solches Stylesheet erfolgt innerhalb von YAML die Einbindung der CSS-Bausteine in den (X)HTML-Quelltext der Webseite. Alle im Layout benötigten CSS-Bausteine werden innerhalb dieser Datei über die <code>@import</code>-Regel eingebunden.</p> <p>Weiterhin finden Sie hier die Datei <i>markup_draft.html</i>, welche die Quelltext-Struktur des YAML-Frameworks enthält.</p>
/yaml/core/	
<i>base.css</i> <i>ie hacks.css</i> <i>print_base.css</i> <i>slim_base.css</i> <i>slim_ie hacks.css</i> <i>slim_print_base.css</i>	<p>In diesem Verzeichnis befinden sich die CSS-Grundbausteine von YAML. Sie stellen den Kern oder auch das Fundament des Frameworks dar. In Verbindung mit vordefinierten XHTML-Markup und der Datei <i>base.css</i> entsteht ein robustes dreispaltiges Basislayout mit Kopf- und Fußbereich (siehe Abschnitt 3.4: Das Basis-Stylesheet).</p> <p>In der Datei <i>ie hacks.css</i> werden alle layout- und strukturunabhängigen CSS-Anpassungen für den Internet Explorer (Versionen 5.x - 7.0) zusammengefasst, siehe (Abschnitt 3.5: CSS-Anpassungen für den Internet Explorer). Sie ist damit ein weiterer Grundbaustein und ihre Einbindung somit in jedem YAML-basierten Layout zwingend erforderlich. Beide Grundbausteine zusammen sorgen für die browserübergreifend einheitliche und fehlerfreie Darstellung des Basislayouts.</p> <p>Als drittes finden Sie hier die <i>print_base.css</i>. Diese enthält grundlegende Anpassungen des Layouts für die Druckausgabe.</p> <p>Von jedem dieser drei Stylesheets existiert weiterhin eine so genannte <i>slim</i>-Variante. Diese sind für den produktiven Einsatz gedacht und sind hinsichtlich der Dateigröße optimiert.</p>
/yaml/screen/	
<i>basemod_draft.css</i> <i>content_default.css</i>	<p>In diesem Verzeichnis befinden CSS-Bausteine, die für die Gestaltung des Bildschirmlayouts verantwortlich sind.</p> <p>Die Datei <i>basemod_draft.css</i> ist eine Vorlage für die Erstellung des Screenlayouts. Sie kann in eigene Projekte kopiert und vordefinierten Container gestaltet bzw. um zusätzliche Elemente ergänzt werden. Innerhalb eines YAML-basierten Layouts (<i>zentrales Stylesheet</i>) werden in der Regel ein oder mehrere solcher Modifikationen (<i>basemod</i>-Dateien) verwendet (siehe Abschnitt 3.6: Erstellung des Screenlayouts sowie Kapitel 4).</p> <p>Als zweites befindet sich in diesem Verzeichnis die Datei <i>content_default.css</i>. Darin werden oft benötigte Inhaltselemente vordefiniert. Auch diese Datei kann als Vorlage in eigene Projekte übernommen und angepasst werden. Nähere Informationen finden Sie im Abschnitt 3.8.</p>

/yaml/navigation/**images/****nav_shinybuttons.css****nav_slidingdoor.css****nav_vlist.css**

In diesem Unterordner finden Sie verschiedene Navigationsbausteine. Innerhalb des YAML-Frameworks werden verschiedene Listennavigationen (horizontale und vertikale Navigationslisten) mitgeliefert.

- *nav_shinybuttons* (Horizontale Navigation)
- *nav_slidingdoor* (Horizontale Navigation)
- *nav_vlist* (Vertikale Navigation)

Nähere Informationen hierzu finden Sie im [Abschnitt 3.7](#).

/yaml/print/**print_003_draft.css****print_020_draft.css****print_023_draft.css****print_100_draft.css****print_103_draft.css****print_120_draft.css****print_123_draft.css**

Dieses Verzeichnis beinhaltet die CSS-Vorlagen zur Druckvorbereitung YAML-basierter Layouts.

Bei diesen Bausteinen handelt es sich um Modifikationen des Bildschirm-Layouts. Nähere Informationen zu Drucklayouts finden Sie im [Abschnitt 3.9: Anpassung des Layouts für Printmedien](#).

/yaml/patches/**patch_layout_draft.css****patch_nav_vlist.css**

Dieses Verzeichnis beinhaltet die Anpassungsdateien für den Internet Explorer. Die Datei *patch_layout_draft.css* stellt eine Vorlage für eine solche Anpassungsdatei dar.

In einem solchen Stylesheet werden alle für das Layout erforderlichen CSS-Anpassungen (CSS-Hacks) für den Internet Explorer zusammengefasst und über einen sogenannten *Conditional Comment* in die Webseite eingebunden (siehe [Abschnitt 3.5: CSS-Anpassungen für den Internet Explorer](#)).

Als zweites finden Sie in diesem Verzeichnis die Datei *patch_nav_vlist.css*. Diese Datei gehört zur Navigationsbaustein *nav_vlist.css* und stellt für diesen die notwendigen CSS-Anpassungen für den Internet Explorer bereit. Nähere Informationen hierzu finden Sie im [Abschnitt 3.7](#).

1.5.3 Mitgelieferte Layoutbeispiele

Die nachfolgend aufgeführten Layoutbeispiele sollen einen kleinen Einblick in die vielfältigen Anwendungsmöglichkeiten des Frameworks geben. Auf einige Beispiele wird im Rahmen der Dokumentation genauer eingegangen, andere sollen lediglich Anregungen zur Lösung oft gestellter Gestaltungsfragen beisteuern.

Die zu den jeweiligen Beispielen benötigten CSS-Bausteine des YAML-Frameworks finden Sie innerhalb der hier angegebenen Verzeichnisse im Unterordner *css*. Um die Bedeutung der einzelnen CSS-Bausteine deutlich zu machen, wurden die Datei- und Verzeichnisnamen des YAML-Frameworks ohne Optimierungen übernommen.

Datei/Verzeichnis**Beschreibung****/examples/01_layout_basics/****3col_standard.html**

In diesem Verzeichnis finden Sie zwei sehr einfache Beispiele. Das Beispiel *3col_standard.html* beinhaltet das YAML-Basislayout, einem einfachen, flexiblen 3-Spalten-Layout



















mit einer horizontalen Navigation dar.	
/examples/02_layouts_2col/	
2col_left_13.html	In diesem Verzeichnis finden Sie alle wichtigen Kombinationen für 2-Spalten-Layouts auf Grundlage des YAML-Basislayouts.
2col_left_31.html	
2col_right_13.html	
2col_right_31.html	
/examples/03_layouts_3col/	
3col_1-2-3.html	In diesem Verzeichnis finden Sie alle Variationen für 3-Spalten-Layouts auf Grundlage des YAML-Basislayouts. Nähere Informationen hierzu finden Sie im Abschnitt 4.4: Freie Spaltenanordnung .
3col_1-3-2.html	
3col_2-1-3.html	
3col_2-3-1.html	
3col_3-1-2.html	
3col_3-2-1.html	
/examples/04_layouts_styling/	
3col_column_backgrounds.html	In diesem Verzeichnis finden Sie zahlreiche Beispiele mit zum Teil umfangreicheren grafischen Anpassungen des Basislayouts. Sie finden hier auch Beispiele zum Einsatz grafischer Spaltentrenner - und -Hintergründe sowie zur Erstellung umlaufender grafischer Rahmen.
3col_column_dividers.html	
3col_faux_columns.html	
3col_gfxborder.html	
/examples/05_layouts_advanced/	
2col_left_seo.html	In diesem Verzeichnis finden Sie drei Beispiellayouts, die sich an üblichen praktischen Anforderungen orientieren. In diesen Beispielen werden jeweils verschiedene Funktionen von YAML in Kombination eingesetzt und erläutert.
3col_fixed_seo.html	
flexible_grids.html	
/examples/06_layouts_minmax_for_ie/	
minmax_js.html	Hier finden Sie ein Beispiel zur Simulation der CSS-Eigenschaften <code>min-width</code> und <code>max-width</code> für den Internet Explorer 5.x und 6.0 über ein spezielles Script. Nähere Informationen hierzu finden Sie im Abschnitt 4.6: Minimale & Maximale Breiten .

1.5.4 Hilfsmittel zur Layoutentwicklung

Wie bereits in der Übersicht erwähnt, finden Sie in diesem Verzeichnis einige Werkzeuge zur Layoutentwicklung. Die hier abgelegten Werkzeuge/Dateien sind für die Funktion des Frameworks nicht erforderlich. Sie stellen keine festen Bestandteile des Frameworks dar.

Datei/Verzeichnis	Beschreibung
/tools/dreamweaver/	
<i>base_dw7.css</i>	Der Dreamweaver ist ein vielseitiger und oft genutzter Editor zur Erstellung von Webseiten. Seine Fähigkeiten zur WYSIWYG-Darstellung CSS-basierter Layouts sind jedoch etwas eingeschränkt. Um die Arbeit mit YAML-basierten Layouts im Entwurfsmodus des Dreamweavers zu erleichtern, finden Sie in diesem Verzeichnis speziell angepasste Entwurfs-Stylesheets. Nähere Informationen finden Sie in Abschnitt 5.1 .
/tools/javascript/	
<i>ftod.js</i>	Dieses kleine Script dient zur dynamischen Erstellung von Blindtexten. Es wird im Rahmen der Anwendungsbeispiele eingesetzt.
<i>minmax.js</i>	Dieses Script ermöglicht die Verwendung der CSS-Eigenschaften <code>min-width</code> und <code>max-width</code> im Internet Explorer. Nähere Informationen hierzu finden Sie im Abschnitt 4.6: Minimale & Maximale Breiten .

1.6 Unterstützte Browser

-  **Windows**
 -  Internet Explorer 5.01
 -  Internet Explorer 5.5
 -  Internet Explorer 6.0
 -  Internet Explorer 7.0
-  **Macintosh OS**
 -  Safari 1.0.3 +
 -  Camino 0.6 +
-  **Linux**
 -  Konqueror 3.3 +
 -  Galeon 1.3 +
 -  Epiphany 1.4.8 +
 -  Lynx (Textbrowser)
- **Betriebssystem übergreifend**
 -  Firefox 1.0 +
 -  Mozilla Suite 1.7.1 +
 -  SeaMonkey 1.0 +
 -  Netscape 8.0 +
 -  Opera 6 +

Die hier aufgeführten Browser werden von YAML in vollem Umfang unterstützt. YAML-basierte Layouts können somit fehlerfrei dargestellt werden.

Ein Plus (+) hinter der Versionsnummer bedeutet, dass auch alle Folgeversionen gleichermaßen fehlerfrei mit YAML zusammenarbeiten sollten.

1.7 IE 5/Mac, Netscape 4 & Co.

Der Internet Explorer 5 für Macintosh und der Netscape Browser in der Version 4, sowie alle andern veralteten Browser haben eine Sonderstellung in der Unterstützung durch YAML.

Veraltete Browser haben große Schwierigkeiten bei der Darstellung aktueller CSS-Layouts. Daher ist es sinnvoll, diesen Browsern das eigentliche CSS-Layout vorzuenthalten - welches sie überfordern würde - und so dem Nutzer dennoch freien Zugang zu den Inhalten zu ermöglichen.



Innerhalb der CSS-Bausteine von YAML wird häufig mit CSS-Regeln wie @import oder @media gearbeitet. Sowohl der Internet Explorer 5/Mac, der Netscape-Browser der Version 4 sowie viele andere veraltete Browser können wenigstens eine der beiden Regeln nicht interpretieren und

werden so automatisch von den CSS-Deklarationen ausgeschlossen. Der Besucher bekommt die Inhalte unformatiert angezeigt, gelangt jedoch zumindest an die Informationen der Webseite.

Speziell einige alte Versionen des Netscape-Browsers stürzen bereits bei einem einfachen, floatenden Bild gnadenlos ab. Die konsequente Umsetzung des Ausschlussprinzips für veraltete Browser von allen CSS-Deklarationen ermöglicht erst den gesicherten Zugang zu den Inhalten.

Kurz: Veraltete Browser werden von YAML in der Weise unterstützt, dass sie bewusst von der fehlerhaften Interpretation der CSS-Formatierung ausgeschlossen werden. Die Inhalte bleiben damit vollständig nutzbar, werden jedoch lediglich im dem vom Browser vorgegebenen Standarddesign angezeigt, ähnlich wie das auch in Textbrowsern (z.B. Lynx) der Fall ist.

1.8 Danksagung

"Yet Another Multicolumn Layout" (kurz YAML) ist ein Ein-Mann-Projekt und entstand im Frühjahr 2005 aus dem Wunsch heraus, ein flexibles und universell einsetzbares Basis-Layout für meine eigenen kleinen Webseiten-Projekte zu besitzen. Es ist als Hobbyprojekt gestartet denn ich bin hauptberuflich Bauingenieur und beschäftige mich lediglich nebenberuflich mit Webdesign. Die Veröffentlichung der Version 1.0 im Oktober 2005 geht auf eine Anregung von Jens Grochtdreis zurück, der mich bereits während der Entwicklung unterstützte.

Seither hat sich das Projekt zu einem umfangreichen und stabilen CSS Framework entwickelt. Mit jeder neuen Version wächst das Interesse genauso wie die Anzahl der Kommentare und E-Mails in meinem Postfach. Dieses Feedback ist für mich als Entwickler besonders wichtig und hilfreich. Daher möchte ich mich an dieser Stelle bei allen Nutzern für ihre Unterstützung bedanken.

In YAML 3 sind viele Ideen und Hinweise engagierter Nutzer eingeflossen. Für diese Feedback und für die wertvolle Unterstützung auf vielen anderen Gebieten, bedanke ich mich besonders bei ...

- [Jens Grochtdreis](#) (für die zahlreichen Hinweise und Diskussionen und überhaupt)
- [Dieter Bunkerd & Detlef Schäbel](#) (für die Hilfe bei TYPO3 und überhaupt)
- [Peter Müller](#) (für zahlreiche Anregungen zum strukturellen Aufbau der Version 3)
- [Ansgar Hein](#) (für die technischen Anregungen zur Version 3)
- [Dirk Ginader](#) (für die Unterstützung bei einigen jQuery-Ideen)
- [Tomas Caspers](#) (für seine Tipps in Sachen Barrierefreiheit)
- [Folkert Groeneveld](#) (für das neue YAML-Logo)
- [Genevieve Cory](#) (für die Übersetzung der Dokumentation)

Weiterhin möchte ich mich bei Reinhard Hiebl, Alexander Hass, Sven Kausche und Bernd Fink bedanken, die als Beta-Tester auf die Qualität meiner Arbeit geachtet haben.

2 Grundlagen

2.1 Ganzheitliches Konzept

Wie aus dem [einleitenden Kapitel](#) bereits deutlich wird, stecken in YAML viele Vorüberlegungen, die sich am einfachsten anhand der XHTML-Quelltextstruktur erläutern lassen. Die hohe Flexibilität von YAML bedingt dabei auch ein gewisses Maß an Komplexität, vor dem Ihnen nicht bange sein muss. In diesem und den folgenden Kapiteln das Grundkonzept von YAML unter Verwendung zahlreicher Beispiele und Quelltextauszüge erläutert.

CSS zu erlernen bzw. es effektiv und fehlerfrei einzusetzen ist nur möglich, wenn man die Fallstricke kennt, die an den Wegrändern lauern. Wie im richtigen Leben ist auch bei der Arbeit mit CSS nicht immer Eitel Sonnenschein. Der Internet Explorer sticht hier besonders heraus und bereitet Anfängern wie Profis immer wieder Kopfzerbrechen durch seine Vielzahl an CSS-Bugs. Aber Angst machen gilt nicht. Trotz dieser Bugs werden Sie sehen, dass sich auch der Internet Explorer problemlos zur fehlerfreien Darstellung moderner barrierearmer CSS-Layouts bewegt werden kann.

Im Rahmen der Dokumentation werde ich meine Erläuterungen daher nicht nur auf standardkonforme Browser stützen, sondern an entsprechender Stelle auf Probleme im Internet Explorer und mögliche Lösungswege aufzeigen. Das verstehe ich unter einem ganzheitlichen Konzept.

Lassen Sie uns beginnen ...

2.2 Die Grundlage: *floats*

Wird ein Element (z.B. ein Bild oder ein Tabelle) als Fließobjekt deklariert, so wird es aus dem normalen Textfluss herausgelöst und stattdessen vom nachfolgenden Text wie ein Hindernis in einem Strom umflossen. Für diese Art der Positionierung von Elementen wird lediglich die links- oder rechtsbündige Lage (`float:left` oder `float:right`) innerhalb des verfügbaren Raumes vereinbart. Der Browser kümmert sich anschließend selbstständig um die korrekte Platzierung der restlichen Inhalte um das Fließobjekt herum.



Hinweis: Um die Funktionsweise von *floats* (Fließumgebungen) besser zu verstehen, sollten Sie sich mit der Theorie beschäftigen. Empfehlenswert ist hierfür der Artikel "[Floats: Die Theorie](#)". Bei diesem Artikel handelt es sich um die deutsche Übersetzung von [Andreas Kalt](#) und [Jens Grochtdreis](#) des englischen Originals "[Float: The Theory](#)" von Big John.

Insbesondere bei flexiblen Layouts bzw. Spalten mit flexiblen Breiten erweisen sich in den Text eingebettete Fließobjekte als vorteilhaft, da der Browser den Textumbruch selbstständig festlegt und die Inhalte optimal innerhalb der Spalte anordnen kann.

Die Aufhebung des Textflusses erfolgt z.B. mit der CSS-Eigenschaft `clear:Wert` ([Beschreibung](#)). Nachteilig dabei ist, dass entsprechend der Funktionsweise des Textflusses gemäß W3C Definition dieser nicht automatisch, z.B. bis zum Ende des laufenden Absatzes oder der nächsten Teilüberschrift, gestoppt werden kann.

Zur Aufhebung des Textflusses ist daher im Regelfall zusätzlicher und daher auch optisch sichtbarer HTML-Code erforderlich. Üblich ist der Einsatz leerer `p`- oder `hr`-Tags. Praktisch ist dies jedoch ganz sicher nicht.

```
<p style="clear:left;">&nbsp;</p>
```

Für die Layoutgestaltung ist dies besonders nachteilig, da solcher, für das Layout "unnützer" HTML-Code vom Browser dargestellt wird und somit im Layout ungewollte vertikale Abstände erzeugt.

Durch den gezielten Einsatz von CSS lässt sich dieser Nachteil umgehen und macht Fließumgebungen für die Layoutgestaltung interessant. Im Frühjahr 2005 beschäftigten sich mehrere Webdesigner mit diesem Thema und veröffentlichten interessante Ideen.

Zwei dieser markupfreien Methoden zum *Clearen* von Fließumgebungen kommen in YAML zum Einsatz. Die beiden Methoden werden im folgenden Abschnitt (rechte Spalte) erläutert.

2.3 Markupfreies Clearing

Der Umgang mit *floats* war lange Zeit sehr aufwändig, da zum Beenden des Textflusses zusätzlicher Code (Markup) benötigt wurde. Ein effektiver Einsatz von *floats* für Layoutzwecke war somit sehr umständlich weil man kaum um Inline-CSS herum kam. Demzufolge wurden *floats* in der Vergangenheit oft nur für einfachste Layoutaufgaben, wie der Ausrichtung von Bildern, herangezogen.

Mit den erweiterten Möglichkeiten von CSS 2 bzw. CSS 2.1 und der mittlerweile guten Browserunterstützung sind die Einsatzgebiete für *floats* nahezu grenzenlos. Der Schlüssel dazu ist das *markupfreie Clearing* der Fließumgebungen über CSS.

2.3.1 Methode 1: Clearfix

Die Clearfix Methode entstammt Big Johns Artikel "How To Clear Floats Without Structural Markup", der die von Tony Aslett [csscreator.com] entwickelte clearing Methode in einem ausführlichen Tutorial erläutert. Eine deutsche Übersetzung dieses Tutorials gibt es hier.

```
/* Clearfix-Hack */
.clearfix:after {
    content: ".";
    display: block;
    height: 0;
    clear: both;
    visibility: hidden;
}

.clearfix {display: inline-table;}

/* Hides from IE-mac */
* html .clearfix {height: 1%;}
.clearfix {display: block;}
/* End hide from IE-mac */
```

Der IE7 verlangt nach einer geringfügigen Anpassung, welche im Artikel "[New clearing method needed for IE7?](#)" erläutert wird.

2.3.2 Methode 2: Overflow

Eine weitere und vor allem sehr einfache Methode zeigt Paul O'Brien auf. Sie wird in dem Artikel [Simple Clearing of Floats](#) ausführlich erläutert. Hierbei bekommt das kapselnde DIV die CSS-Eigenschaft `overflow:auto`; zugewiesen. Diese Methode hat sich insbesondere bei der Kapselung von Fließumgebungen innerhalb der Content-Spalten als robust erwiesen.

Der Wert `auto` erzeugt jedoch in einigen Browsern zu unerwünschte Scrollbalken an den Rändern des umschließenden Containers. Innerhalb des YAML Frameworks wird daher `overflow:hidden` verwendet. Das Scrollbalkenproblem tritt hierbei nicht auf.

```
/* Clearing mit overflow */
.floatbox { overflow: hidden; }
```

Näheres dazu im Abschnitt 2.6: Funktionsweise von *floats*.

2.3.3 Warum zwei Clearing-Methoden?

Die Frage ist durchaus berechtigt und hat eine ebenso klare Antwort. Obwohl beide Methoden prinzipiell zum selben Ergebnis führen, dass das Elternelement die Fließumgebung einschließt, ist die technische Funktionsweise verschieden.

In Abhängigkeit davon, an welcher Stelle innerhalb des Layouts die CSS-Eigenschaft `clear` eingesetzt wird, kann diese global auf das gesamte Layout oder nur lokal innerhalb des Elterncontainers wirken. Eine genaue Erläuterung dazu finden Sie im [Abschnitt 2.6: Die Funktionsweise von floats](#).

Die Overflow-Variante kommt an solchen Stellen zum Einsatz, an denen ein Clearing mit Clearfix nicht erwünscht ist (Stichwort: Globale Wirkung der Eigenschaft `clear`).

2.4 Der Aufbau des XHTML-Quelltextes

Das Ziel des YAML-Frameworks besteht darin, ein universell einsetzbares, browserübergreifend voll funktionsfähiges Layout zu liefern indem die dafür erforderlichen XHTML-Strukturen unabhängig von den späteren Inhalten bereitgestellt werden. Speziell bei bedeutet dies, dem Seitenersteller größtmögliche Freiheiten bei der Wahl fixer oder flexibler Layouts bzw. Spaltenbreiten zu lassen. Weiterhin soll auch ein gewisser Komfort geboten werden, weshalb oft benötigte Elemente vordefiniert sind bzw. gestalterische Erfordernisse in der Struktur berücksichtigt werden.

Das Ergebnis ist eine universelle Quelltext-Struktur, die ohne Änderungen im zugrunde liegenden Markup eine Vielzahl an Modifikationsmöglichkeiten per CSS bietet. Die Quelltext-Struktur liegt dem im Download-Paket als inhaltsleere HTML-Datei bei.

[/yaml/markup_draft.html](#)

2.4.1 Wahl des Doctypes

Für die vorliegende Quelltextstruktur wurde der Doctype *XHTML 1.0 Transitional* gewählt. Grundsätzlich obliegt die Entscheidung über den Doctype bei Ihnen. Sie können also ebenso eine *Strict*-Variante von *XHTML* verwenden oder beispielsweise mit *HTML 4.01* arbeiten, falls dies Ihre Inhalte erfordern sollten.

Standard Mode

In diesem Darstellungsmodus interpretiert der Browser (X)HTML so, wie es vom W3C definiert wird. Bei Fehlern im (X)HTML-Code kommt es schnell zu schwerwiegenden Darstellungsfehlern. Für fehlerfreie Seiten besteht in diesem Darstellungsmodus jedoch größtmögliche Sicherheit für ein browserübergreifend einheitliches Erscheinungsbild einer Webseite.

Quirks Mode

In diesem Darstellungsmodus ist der Browser deutlich fehlertoleranter und wird versuchen, unter allen Umständen eine brauchbare Webseite anzuzeigen. Dieser Modus wird in allen Browsern automatisch verwendet, wenn kein oder ein sehr alter Doctype (oder auch ein falsch geschriebener) im HTML-Dokument vorgegeben wird. Der Internet Explorer 5.x kennt übrigens nur diesen Darstellungsmodus.

Entscheidend für die korrekte Darstellung des Layouts - insbesondere im Internet Explorer - ist letztlich der durch den gewählten Doctype im Browser aktivierte Darstellungsmodus. Alle CSS-Bausteine von YAML, einschließlich der CSS-Anpassungen für den Internet Explorer sind darauf ausgerichtet, dass sich der Browser in einem standardkonformen Darstellungsmodus (**Standard Mode**) befindet.

2.4.2 Die Struktur im Detail

Lassen Sie uns nun einen Blick auf das Fundament des YAML-Frameworks werfen. Hier ein Auszug der Datei *markup_draft.html*:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head> ... </head>

<body>
<div id="page_margins">
  <div id="page">
    <div id="header"> ... </div>
    <div id="nav"> ...</div>

    <!-- begin: main content area #main -->
    <div id="main">

      <!-- begin: #col1 - first float column -->
      <div id="col1">
        <div id="col1_content" class="clearfix">
          ...
```

```

        </div>
    </div>

    <!-- begin: #col2 - second float column -->
    <div id="col2">
        <div id="col2_content" class="clearfix">
            ...
        </div>
    </div>

    <!-- begin: #col3 static column -->
    <div id="col3">
        <div id="col3_content" class="clearfix">
            ...
        </div>
        <!-- IE Column Clearing -->
        <div id="ie_clearing">&nbsp;</div>
    </div>

    <!-- end: #main -->
</div>

<!-- begin: #footer -->
<div id="footer"> ... </div>
</div>
</div>
</body>
</html>

```

Der äußerste DIV-Container `#page_margins` regelt in erster Linie die Gesamtbreite des Layout. Er umfasst alle weiteren Container, daher kann die Layoutbreite bzw. die maximale und minimale Breite eines flexiblen Layouts an dieser Stelle festgelegt werden.

Weiterhin kann der Container zusammen mit dem Container `#page` für die Bereitstellung von grafischen Layouträndern verwendet werden, doch dazu später mehr. Beiden Containern wird im IE die proprietäre Eigenschaft `hasLayout` zugewiesen, zur Vermeidung verschiedener CSS-Bugs (z.B. [Escaping Floats Bug](#) bei Verwendung horizontaler Menüs auf *float*-Basis). Für weitere Informationen hierzu, siehe [Abschnitt 3.5: CSS-Anpassungen für den Internet Explorer](#)

Es folgen die Container für den Kopfbereich `#header`, die Hauptnavigation `#nav` sowie der Hauptinhaltsbereich `#main` mit seinen drei Spalten. Den Abschluss bildet, wie zu erwarten, der `#footer` welcher die Inhalte der Fußzeile aufnimmt.

Der im Quelltextauszug rot markierte Bereich des *IE Column Clearings* ist eine Besonderheit von YAML. Die Bedeutung und Wirkungsweise dieses Containers wird im [Abschnitt 2.7: Das Clearing von #col3](#) dieses Kapitels ausführlich besprochen.

2.4.3 Gestaltungsfreiheit durch das Kombinationsmodell

Das Box-Modell, wie es seit CSS 1 existiert, ist eindeutig auf die Arbeit fixen Maßeinheiten (z.B. Pixelmaßen) ausgelegt. Die Gesamtbreite eines Containers wird über die Addition der einzelnen Bestandteile des Modells (`width`, `padding`, `border`) bestimmt.

Bei der Mischung von Maßeinheiten (z.B. `width:25%; padding:0 10px;`) innerhalb eines Containers ist es jedoch nicht mehr möglich, die Gesamtbreite der Box vorab rechnerisch zu ermitteln. Die Gestaltungsfreiheit bei der Erstellung flexibler Layouts ist damit von vorn herein eingeschränkt.

Ein weiteres Problem bei flexiblen Spaltenbreiten betrifft speziell den Internet-Explorer. Dieser interpretiert im *Quirks Mode* das [CSS-Box-Modell](#) falsch. Der IE 6 lässt sich durch einen passenden *Doctype* in den standardkonformen Darstellungsmodus versetzen. Jedoch wurde YAML von Beginn an für eine volle Unterstützung der Version 5.x des Internet Explorers ausgelegt. Diese Version arbeitet generell im *Quirks Mode*.

Um den IE trotzdem zur Darstellung der korrekten Breite zu überreden, existiert wurde der [Box-Modell-Hack](#) entwickelt, bzw. noch zahlreiche weitere Variationen dieses Hacks. Alle Varianten dieses Hacks basieren jedoch auf dem gleichen Prinzip: Über die Ausnutzung von Parser-Bugs wird dem IE eine modifizierte Breite zugeschoben welche die Fehlinterpretation berücksichtigt und im Ergebnis zu Darstellung der korrekten Breite führt. Dieses Prinzip scheitert jedoch bei der Mischung von Maßeinheiten auf Grund der oben beschriebenen Problematik und stellt damit eine weitere massive Einschränkung für die Gestaltungsfreiheit des Seitenerstellers dar.

Die Lösung all dieser Probleme liegt in dem bei YAML umgesetzten Kombinationsmodell für die Spalten des Basislayouts, durch jeweils zwei ineinander verschachtelte DIV-Container.

```
<!-- begin: #col1 - first float column -->
<div id="col1">
  <div id="col1_content" class="clearfix">
    ...
  </div>
</div>
```

Die Gesamtbreite der Spalte wird dem äußeren Container `#colx` zugewiesen. Die Innenabstände (`padding`) und eventuelle Rahmendefinitionen (`border`) gehen hingegen an den inneren Container `colx_content`, welcher wiederum keine definierte Breite (`width:auto`) erhält.

Auf diese Weise ist die Gesamtbreite eines Containers `#colx` immer eindeutig. Somit sind beliebige Kombinationen auf unterschiedlichen Maßeinheiten möglich, was den Gestaltungsfreiraum für flexible Layouts enorm erhöht und gleichzeitig wird das Auftreten Box-Modell-Bug des Internet Explorers geschickt umgangen.

2.5 Reihenfolge der Spalten-Container im Quelltext

Beide Spalten `#col1` und `#col2` sind *float*-Umgebungen. Bei der dritten Spalte `#col3` handelt es sich um einen statischen Container. Die Reihenfolge, in der diese drei Container im Quelltext erscheinen, ist daher nicht beliebig wählbar. Die *float*-Objekte (`#col1` und `#col2`) müssen im Quelltext immer **vor** dem statischen Objekt (dem Container `#col3`) befinden.

Die CSS-Deklarationen der *float*-Spalten finden sich in der Datei `yaml/core/base.css`:

```
#col1 {
  float: left;
```

```

    width: 200px; /* Standard-Wert */
}

...

#col2 {
    float: right;
    width: 200px; /* Standard-Wert */
}

```

Im Basis-Layout werden die beiden Spaltencontainer `#col1` und `#col2` an den linken bzw. rechten Rand *gefloatet*. Somit fällt `#col3` die Rolle der mittleren Spalte in diesem dreispaltigen Grundaufbau zu.

Wie man in der [XHTML-Struktur](#) erkennt, sind die einzelnen Spalten nicht durch zusätzliche Container (oft als *wrapper* bezeichnet) verschachtelt. Alle drei Spaltencontainer befinden sich innerhalb von `#main` auf einer Strukturebene. Die beiden *float*-Spalten befinden sich jedoch außerhalb des normalen Elementflusses. Der statische Container `#col3` beansprucht daher die gesamte verfügbare Breite für sich. Die beiden Container `#col1` und `#col2` schweben sprichwörtlich darüber.

Damit das keinen Ärger gibt und sich die Inhalte von `#col3` nicht mit denen der beiden *float*-Spalten überdecken, müssen per CSS entsprechende Vorkehrungen getroffen werden. Für die *float*-Spalten wird im Basislayout eine Standardbreite von 200 Pixeln festgelegt. Mit der Angabe eines 200 Pixel breiten Außenabstands (*margin*) sowie mit der Breite `width:auto;` für `#col3`, werden die Inhalte von `#col3` in das Korsett zwischen `#col1` und `#col2` gezwungen. Die hier angegebenen CSS-Deklarationen finden Sie in der Datei: `yaml/core/base.css`.

```

#col3 {
    width:auto;
    margin-left: 200px; /* Standard-Wert */
    margin-right: 200px; /* Standard-Wert */
}

```

Wichtig: Die Reihenfolge Container `#col1`, `#col2` und `#col3` sollte im (X)HTML-Quelltext unverändert bleiben. Ordnen Sie ihre Inhalte in der gewünschten Abfolge in die Spaltencontainer ein. Die Reihenfolge der Inhalte im Quelltext ist vollkommen unabhängig von der Darstellung der Spalten auf der Webseite. Details dazu erfahren Sie im [Abschnitt 4.4: Freie Anordnung und Verwendung der Content-Spalten](#).

Damit wäre geklärt, wie die drei Container `#col1`, `#col2` und `#col3` im Quelltext angeordnet sind und wie sie per CSS grundlegend positioniert werden. Bleibt also eine Frage: Warum werden die drei Spalten im Quelltext **unverschachtelt** innerhalb von `#main` angeordnet?

Die Antwort darauf folgt im [Abschnitt 2.7: Das Clearing der Spalte #col3](#). Zuvor jedoch folgt zum bessern Verständnis der Materie ein kleiner Einschub zur Funktionsweise von *floats*.

2.6 Die Funktionsweise von floats im Detail

Beim Umgang mit *float*-Umgebungen ist Folgendes zu beachten: Innerhalb eines statischen Elements wirkt die CSS-Eigenschaft `clear: left | right | both` nicht lokal innerhalb des umgebenden Elternelements, sondern **global**, d.h. es werden alle auf der Webseite vorhandenen *float*-

Umgebungen auf der gleichen Strukturebene berücksichtigt. Wie sich dieses Verhalten auswirkt, lässt sich am besten an einem Beispiel erklären.

Warnung: Im Internet-Explorer 5.x und 6.0 kann es beim Betrachten dieser Datei zu Darstellungsfehlern kommen. Die IE-Float-Bugs sind hier nicht beseitigt. Verwenden Sie daher gegebenenfalls einen anderen Browser (Firefox, Safari, Opera ...).

2.6.1 Vorbereitung des Layouts

Zunächst muss dafür gesorgt werden, dass innerhalb der Spalten frei mit Fließobjekten gearbeitet werden kann. Dazu muss sichergestellt werden, dass die späteren Inhalte in jedem Fall vollständig von den statischen Containern `#col1_content`, `#col2_content` und `#col3_content` umschlossen werden.

Zu diesem Zweck wird den drei Containern die CSS-Klasse: `.clearfix` zugewiesen. Der Clearfix-Hack sorgt dafür, dass alle Inhalte (statische Inhalte und/oder Fließumgebungen) automatisch eingeschlossen werden. Die Definition der Klasse finden Sie in der Datei `base.css`.

```
/* Clearfix-Methode zum Clearen der Float-Umgebungen */
.clearfix:after {
  content: ".";
  display: block;
  height: 0;
  clear: both;
  visibility: hidden;
}

/* Diese Angabe benötigt der Safari-Browser zwingend !! */
.clearfix { display: block; }
```

Wichtig: Obwohl die Klasse `.clearfix` im YAML-Framework nur an Block-Level-Elemente (DIV-Container) vergeben wird, benötigt der Safari Browser zwingend die explizite Angabe von `display: block`. Andernfalls wird der Container `#col3_content` mit einer viel zu geringen Breite dargestellt. Für alle anderen modernen Browser, wie der Firefox oder Opera, ist dieser Wert redundant.

Wie man sieht, kommt auch beim Clearfix-Hack `clear: both` zum Einsatz. Innerhalb der *float*-Spalten `#col1` und `#col2` wirkt die Eigenschaft `clear` daher nur lokal. Genau, wie man es sich wünscht. Innerhalb des statischen Containers `#col3` jedoch wirkt `clear: both` jedoch global und sorgt somit dafür, dass der Container `#col3_content` bis zum unteren Ende der längsten *float*-Spalte verlängert wird. Auch dieses Verhalten ist innerhalb des YAML-Frameworks ausdrücklich erwünscht.

Der Internet Explorer kann bis zur Version 6 mit der CSS Pseudo-Klasse `:after` leider nichts anfangen. Die Clearfix-Methode ist aber nicht vollständig wirkungslos im Internet Explorer. Innerhalb der beiden Container `#col1_content` und `#col2_content` wird sie zum Einschließen der Inhalte herangezogen. Hierzu sind noch ein paar Anpassungen für den IE von Nöten. Diese Anpassungen werden zentral in der Datei `yaml/core/ie hacks.css` verwaltet. Näheres, siehe [Abschnitt 3.5: CSS-Anpassungen für den Internet Explorer](#).

```

/*-----*/
/* Workaround: Clearfix-Anpassung für alle IE-Versionen */
/*
** IE7 - x
*/
.clearfix { display: inline-block; }
/*
** IE5.x/Win - x
** IE6 - x
*/
* html .clearfix { height: 1%; }
.clearfix { display: block; }
/*-----*/

```

Der Internet Explorer ignoriert die Eigenschaft `clear:both`; jedoch aufgrund der fehlenden CSS-Unterstützung der Pseudo-Klasse `:after` und löst somit kein globales Clearing innerhalb von `#col3` aus. Um den IE trotzdem zu einem globalen Clearing zu bewegen, wird ein spezieller DIV-Container (`#ie_clearing`) am Ende von `#col3` in die Quelltext-Struktur eingebaut. Eine detaillierte Erläuterung hierzu finden Sie im nachfolgenden [Abschnitt 2.7: Das Clearing von #col3](#).

Hinweis: Als weiterführende Informationsquelle, vor allem was die Funktionsweise von *float*-Umgebungen und den Umgang mit verschiedenen Browsern betrifft, sei an dieser Stelle auf den sehr ausführlichen Artikel "[Grundlagen für Spaltenlayouts mit CSS](#)" von Mathias Schäfer im [SelfHTML-Weblog](#) verwiesen.

2.6.2 Aufbereitung der Inhalte

Für die späteren Inhalte muss eine Möglichkeit gefunden werden, den Textfluss innerhalb der statischen Container `#col3` zu steuern, ohne das globale Verhalten von `clear:both` auszulösen. Innerhalb der floatenden Spalten `#col1` und `#col2` ist die Verwendung dieser Eigenschaft unproblematisch, da das Clearing hier generell nur lokal innerhalb der Spalten wirkt. Innerhalb von `#col3` ist die Wirkungsweise, wie eben beschrieben, jedoch global und würde zu großen vertikalen Zwischenräumen führen. Zumindest, wenn man nichts dagegen tut.

Die Lösung liegt in der Overflow-Methode, die ebenfalls das Einschließen von Fließumgebungen ermöglicht. Die Overflow-Methode arbeitet mit der Eigenschaft `overflow:hidden`, daher entstehen keine Konflikte mit dem Clearing der Spalten. Für die Aufbereitung der Inhalte steht im YAML-Framework die CSS-Klasse `.floatbox` zur Verfügung deren Anwendung in den zwei folgenden Beispielen erläutert wird.

Die Definition der CSS-Klasse `.floatbox` finden Sie in der Datei `base.css`.

```

/* Clearen per Overflow */
.floatbox { overflow:hidden; }

```

Auch bei `.floatbox` benötigt der Internet Explorer etwas Hilfe. Diese findet sich wiederum in der globalen Anpassungsdatei `ie hacks.css` (Nähreres, siehe [Abschnitt 3.5: CSS-Anpassungen für den Internet Explorer](#)).

```

/* .floatbox-Anpassung für IE */
* html .floatbox {width:100%;}

```

In den Spalten kann beliebig mit Fließobjekten gearbeitet werden. Dabei kann es sinnvoll sein, den Textfluss auf einen bestimmten Bereich zu beschränken, z.B. bis zur nächsten Teilüberschrift. Dadurch kann verhindert werden, dass beispielsweise eine Grafik in einen nachfolgenden, inhaltlich fremden, Abschnitt hinein fließt.

Dazu wird der Inhaltsbereich, über den sich der Textfluss erstrecken soll, gekapselt. Hierzu steht die CSS-Klasse `.floatbox` (basierend auf der Overflow-Methode) bereit. Dazu zwei Beispiele:

Beispiel 1: Ein Bild soll innerhalb eines Absatzes (Paragraph) vom Text umflossen werden. In diesem Fall wird dem umschließenden `p`-Tag die Klasse `.floatbox` zugewiesen. Der Textfluss ist damit auf diesen einen Absatz begrenzt - es ist kein weiterer HTML-Code zum Beenden des Textflusses erforderlich.

```
<p class="floatbox">
  
  Hier folgt der Text des Absatzes, welcher das Bild umfließt ...
</p>

<p>Hier ist der Textfluss zu Ende. Dieser Absatz beginnt immer unterhalb
des Bildes.</p>
```

Beispiel 2: Ein Bild soll innerhalb eines Textabschnitts, bestehend aus mehreren Absätzen umflossen werden. Der Textfluss soll jedoch vor der nächsten Teilüberschrift enden.

Dazu wird der entsprechende Abschnitt mit einem speziellen DIV-Container `class="floatbox"` gekapselt. Innerhalb dieses DIV-Containers können Fließobjekte mit `float:left` oder `float:right` beliebig verwendet werden:

```
...
<h2>Teilüberschrift 1</h2>
<div class="floatbox">
  
  <p> ... viel viel Fließtext ...</p>
  <p> ... noch mehr Fließtext ...</p>
  <p> ... und noch ein letzter Absatz im Textfluss</p>
</div>

<h2>Teilüberschrift 2</h2>
...
```

Der Textfluss ist durch die Kapselung auf den DIV-Container beschränkt. Es braucht am Ende daher kein spezieller HTML Code mit `clear:both`; eingefügt werden.

2.7 Das Clearing der Spalte #col3

Im [vorigen Abschnitt](#) wurde bereits auf das globale Verhalten von `clear:both` und seine Auswirkungen innerhalb des statischen Containers `#col3` hingewiesen. Während dieser Effekt in Bezug auf die Position der Inhalte innerhalb von `#col3` unerwünscht ist, wird dieser Effekt innerhalb von YAML gezielt ausgenutzt, um `#col3` immer zur längsten Spalte werden zu lassen — unabhängig vom Füllstand der einzelnen Spalten.

Das Ziel dieser Bemühungen ist es, die CSS-Eigenschaft `border` von `#col3` dazu zu verwenden, um vertikale Spaltentrennlinien (vertikale Voll-, Strich- oder Punktlinien) oder auch einfarbige Spaltenhintergründe für die *float*-Spalten ohne Grafikeinsatz erzeugen zu können. Durch das globale Clearing würden diese immer bis zum `#footer` reichen. Damit entstünde eine alternative Möglichkeit zur grafischen Layoutgestaltung, die zudem extrem einfach editierbar wäre.

2.7.1 Globales Clearing macht #col3 zur längsten Spalte

Wie wird `#col3` zur längsten Spalte? In allen modernen Browser (Mozilla, Firefox, Opera usw.) erfolgt dieses bereits ohne weitere Maßnahmen. Da es sich bei `#col3` um einen statischen Container handelt, wirkt das durch die Klasse `clearfix` ausgelöste Clearing von `#col3_content` global und erzwingt eine Verlängerung von `#col3` bis zum unteren Ende der längsten *float*-Spalte. Näheres zur Funktionsweise der `clearfix`-Klasse finden Sie im [Abschnitt 2.6: Die Funktionsweise von floats](#).

2.7.2 Eine spezielle Clearing-Lösung für den Internet Explorer

Im Internet Explorer funktioniert dieses globale Clearing der Clearfix-Lösung nicht, da der Internet Explorer die CSS2-Pseudoklasse `:after`, welche die Eigenschaft `clear:both` enthält, nicht versteht. Daher muss am Ende der Spalte `#col3` ein zusätzliches HTML-Element eingefügt werden, welches die Clearing-Anweisung beinhaltet. Dies geschieht in Form eines unsichtbaren `DIV`-Tags.

```
...
<!-- IE column clearing -->
<div id="ie_clearing"> </div>
...
```

Schauen wir uns nun den *unsichtbaren* `DIV`-Container etwas genauer an. Ich erwähnte bereits, dass dieser spezielle Container nur im Internet Explorer benötigt wird. In allen modernen Browser wird er daher komplett abgeschaltet. Die dafür nötigen Angaben finden sich in der Datei `base.css` im Verzeichnis `yaml/core/`:

```
/* IE-Clearing: ... */
#ie_clearing { display: none }
```

Die Anpassung der Eigenschaften dieses speziellen Clearing-DIVs für den Internet Explorer sind in der Datei `ie hacks.css` im Verzeichnis `yaml/core/` abgelegt:

```
#ie_clearing {
  display:block; /* DIV sichtbar machen */
  \clear:both; /* Normales Clearing für IE5.x/Win */

  width: 100%; /* IE-Clearing mit 100%-DIV für IE 6 */
  font-size:0;
  margin: -2px 0 -1em 1px; /* IE-Clearing mit übergroßem DIV für IE7 */
}

* html { margin: -2px 0 -1em 0 }

/* Vermeidung horizontaler Scrollbalken bei randabfallenden Layouts im IE7
*/
```

```
html {margin-right: 1px}
* html {margin-right: 0} /* Der IE6 benötigt das nicht */

#col3_content { margin-bottom:-2px }
#col3 { position:relative } /* Erforderlich für IE7 */
```

IE-Clearing im Internet Explorer 5.x

Mit `display:block` wird im IE zunächst die Darstellung des Containers aktiviert. Danach folgt das eigentliche Clearing mittels `\clear:both`. Mit Hilfe des Backslash wird Parser-Bug des IE 5.x und 6.0 ausgenutzt, der dafür sorgt, dass die Eigenschaft nur vom Internet Explorer 5.x verstanden wird.

Wichtig: Es handelt sich hierbei um das Standardvorgehen zum Clearen von Fließumgebungen. Allerdings tritt hierbei im Internet Explorer v5.x bis v7 ein unangenehmer Bug auf, der unter bestimmten Randbedingungen zum Kollabieren des linken Margins von `#col3` führt. Nähere Informationen dazu finden Sie im [Abschnitt 5.3: Bekannte Probleme - Internet Explorer](#). Im IE 5.x ist dieser Bug nicht zu beheben, daher wird bei dieser Browserversion das reguläre Clearing dennoch eingesetzt.

Für den Internet Explorer 6 und 7, die ebenfalls von diesem Bug betroffen sind, wird ein spezielles Clearing-Verfahren eingesetzt, welches das Auftreten dieses Bugs verhindert.

IE-Clearing im Internet Explorer 6.0

Die Clearing-Lösung basiert im Grunde darauf, dass der Internet Explorer übergroße Elemente innerhalb von `#col3` selbstständig unter die *float*-Spalten umbricht. Um dies zu erreichen, erhält der DIV-Container `#ie_clearing` die Eigenschaft `width: 100%` im IE6. Da die *float*-Spalten jedoch den zur Verfügung stehenden Platz in jedem Fall auf unter 100 Prozent einschränken, wird ein Umbruch des Containers unter die *float*-Spalten quasi provoziert.

IE-Clearing im Internet Explorer 7.0

Der IE7 benötigt hierfür einen Box mit einer Größe von über 100 Prozent. Daher erhält der Container ein zusätzlichen linken Margin von 1 Pixel `margin: -2px 0 -1em 1px`. Der Internet Explorer 7 hat jedoch einen Bug, der dazu führt, dass dieser überstehende Pixel - der im Layout keinerlei Bedeutung hat - bei randabfallenden Layouts (`body`, `#page_margins` und `#page` auf 100% Breite und ohne Rahmen) horizontale Scrollbalken erzeugt. Um auch diesen Sonderfall abzufangen, erhält das HTML-Element `html` einen 1 Pixel breiten seitlichen Margin.

```
/* Vermeidung horizontaler Scrollbalken bei randabfallenden Layouts im IE7
*/
html {margin-right: 1px}
* html {margin-right: 0}
```

Durch diesen Trick werden horizontale Scrollbalken verhindert und der verbleibende 1 Pixel breite Rand neben dem vertikalen Scrollbalken des IE7 wird in der Regel nicht störend wahrgenommen.

Zum Schluss folgt noch eine weitere Hilfestellung für den Internet Explorer 7. Dem Container `#col3` muss zum Abschluss die Eigenschaft `position:relative` zugewiesen werden. Ohne sie würde der Internet Explorer 7 den Container `#ie_clearing` ignorieren.

Clearing-Container im Layout verstecken

Die Margins in den anderen Richtungen `margin: -2px 0 -1em 1px` dienen lediglich dazu, den Container in allen IE-Versionen optisch "unsichtbar" werden zu lassen. Um den DIV-Container endgültig unsichtbar zu machen wird die Schriftgröße auf Null gesetzt. Damit schrumpft die Höhe des Containers auf 2 Pixel zusammen. Diese letzten 2 Pixel werden durch einen weiteren negativen Margin an `#col3_content` ausgeglichen. Damit wird der DIV-Container im Layout nicht sichtbar, erfüllt jedoch seine Aufgaben.

Und noch eine letzte Anpassung ist erforderlich. Das IE-Clearing funktioniert nur solange die Spalte `#col3` **nicht** die proprietäre Eigenschaft `hasLayout` zugewiesen bekommt. Genau das kann jedoch beispielsweise bei der Beseitigung des 3-Pixel-Bugs (siehe [Abschnitt 3.5: CSS-Anpassungen für den Internet Explorer](#)) geschehen. In diesem Fall sind die Spaltentrenner also nicht einsetzbar. Dennoch muss für ein korrektes Clearing der Spalten gesorgt werden, um den Footer unterhalb der Spalten zu platzieren. Dies geschieht einfach, indem dem Container `#footer` in der Datei `base.css` ebenfalls die Eigenschaft `clear:both`; zugewiesen wird.

2.7.3 Grafikfreie Spaltentrenner und Spaltenhintergründe

Geschafft: Jetzt ist es möglich, die CSS-Eigenschaft `border` von `#col3` für vertikale Spaltentrennlinien einzusetzen und oder auch einfarbige Spaltenhintergründe für `#col1` und `#col2` zu definieren, die bis zum Footer reichen. Und all das, ohne eine einzige Grafik. Als Beispiel könnte man eine vertikale Punkt-Linien erzeugen:

```
#col3 {
  border-left: 2px #eee dotted;
  border-right: 2px #eee dotted;
}
```

Sie wollen einen Beweis, dass es funktioniert? Bitte schön.

/examples/04_layouts_styling/3col_column_dividers.html

Eine detaillierte Beschreibung dieses Beispiels sowie ein weiteres Beispiel zur Erzeugung farbiger Spaltenhintergründe mit Hilfe dieser Technik finden Sie im [Abschnitt 4.2: Gestaltung der Spalten](#).

Hinweis: Die Anwendung dieser Technik ist nur sinnvoll in Verbindung mit Spaltenanordnungen bei denen sich `#col3` in Mittellage befindet, also 1-3-2 und 2-3-1 bzw. bei 2-Spalten-Layouts. Nähere Informationen zur freien Spaltenanordnung finden Sie im [Abschnitt 4.4: Freie Spaltenanordnung](#).

Bei Verwendung der Spaltanordnungen 1-2-3 / 3-2-1 oder 2-1-3 / 3-1-2 kann diese Technik leider generell nur bedingt eingesetzt werden, da hier im Internet Explorer das Verlängern von `#col3` bis auf Höhe der längsten *float*-Spalte nicht funktioniert. Bei diesen Layoutvarianten sollten Sie auf die "[Faux Columns](#)" Technik zur Gestaltung von Spaltenhintergründen zurückgreifen.

Damit sind der strukturelle Aufbau des XHTML-Quelltextes und die Funktionsweise des IE-Clearings von YAML besprochen. Das Fundament steht somit. Als letzter Punkt der Quelltext-Struktur fehlen noch die Skip-Links, welche im folgenden Abschnitt erläutert werden.

2.8 Skip-Link-Navigation

Skip-Links verbessern in erster Linie die Nutzungsqualität einer Webseite für Internet-Nutzer, die auf die Hilfe von Screenreadern angewiesen sind. Screenreader linearisieren den Inhalt der Webseite und lesen ihn der Reihe nach vor. Skip-Links sollten möglichst an oberster Stelle im Quelltext angeordnet und stellen Sprungmarken zu den wichtigsten Bereichen innerhalb der Webseite (Navigation, Inhalt, usw.) bereit.

Zum Teil ergibt sich daraus die Diskussion, den Inhalt einer Webseite möglichst weit oben im Quelltext anzuordnen und Navigationselemente im Quelltext nach unten zu schieben. Auf diese Weise gelangt der Nutzer direkt zum Inhalt, ohne sich auf jeder Seite erst die Navigationselemente vorlesen lassen zu müssen.

Doch was, wenn der Nutzer den Inhalt gar nicht lesen will? Während er noch nach den richtigen Informationen sucht, wäre es sogar störend, wenn erst nach dem gesamten Seiteninhalt die Navigationselemente für das Erreichen des nächsten Untermenüpunktes folgen. Es gibt daher keine festgeschriebene optimale Platzierung der Inhalte im Quelltext. Vielmehr benötigen wir ein Hilfsmittel, um dem Nutzer die Möglichkeit zu geben, schnell zu allen wichtigen Bereichen der Webseite zu gelangen. Skip-Links sind hierfür ein sehr einfacher und effektiver Weg.

2.8.1 Skip-Link-Navigation im YAML-Framework

In der Quelltextstruktur von YAML befinden sich die Skip-Links im DIV-Container `#topnav`, noch vor dem Link zum Impressum. Der erste Link `#navigation` führt an allen weiteren in `#topnav` und in `#header` enthaltene Inhalten vorbei direkt zur Hauptnavigation. Der zweite Skip-Link `#content` zeigt auf den Startpunkt des eigentlichen Inhaltsbereiches der Webseite.

```
...

<div id="header">
  <div id="topnav">
    <a class="skip" href="#navigation" title="Direkt zur Navigation
springen">Zur Navigation springen</a>
    <a class="skip" href="#content" title="Direkt zum Inhalt springen">Zum
Inhalt springen</a>
  </div>
  ...
</div>

...

<div id="nav">
  <a id="navigation" name="navigation" />
  ...
</div>

...

<div id="col3">
  <div id="col3_content" class="clearfix">
    <a id="content" name="content" />
    ...
  </div>
</div>
```

```

    </div>
    ...
  </div>

```

Die Sprungmarke für `#content` muss je nach Nutzung der einzelnen Spalten durch den Seitenersteller platziert werden. In dem hier abgedruckten Quelltext-Auszug soll der Seiteninhalt in der Container `#col2` beginnen. Der Container `#col1` enthält in diesem Gedankenbeispiel eine zweite Navigationsebene beherbergen und wird daher übersprungen. Diesen Aufbau der Skip-Link-Navigation finden sie auf innerhalb YAML-Dokumentation selbst.

Unsichtbar und Barrierefrei

Nutzer ohne Behinderung, die sich mit einem Standardbrowser durch das Internet bewegen, benötigen diese Navigationshilfe in der Regel nicht. Aus diesem Grund werden die Skip-Links in der normalen Bildschirmansicht und auch im Ausdruck verborgen.

Die dazugehörige CSS-Klasse `.skip` wird in der CSS-Datei `base.css` (siehe [Abschnitt 3.3: Das Basis-Stylesheet](#)) definiert:

```

/**
 * @section hidden elements | Versteckte Elemente
 * @see      ...
 *
 * Skip-Links und versteckte Inhalte
 */

/* Klassen für unsichtbare Elemente im Basislayout */
.skip, .hideme, .print {
  position: absolute;
  top: -1000em;
  left: -1000em;
  height: 1px;
  width: 1px;
}

/* Skip-Links für Tab-Navigation sichtbar schalten */
.skip:focus, .skip:active {
  position: static;
  top: 0;
  left: 0;
  height: auto;
  width: auto;
}

```

Damit sind die Skip-Links am Bildschirm und im Ausdruck unsichtbar. Die Eigenschaft `display:none`; wäre hier problematisch, da viele Screenreader dadurch die Links nicht anzeigen. Eine sehr gute Übersicht zu möglichen Darstellungsmethoden von Skiplinks enthält der Artikel "[Skip Navigation](#)" von Jim Thatcher. Bei Verwendung der Tab-Navigation im Browser sollten Skip-Links allerdings sichtbar werden, um keine Verwirrung beim Nutzer zu erzeugen. Hierfür wird die Klasse `.skip` für die beiden Zustände `:focus` und `:hover` sichtbar geschaltet.

Diese Art der Navigationshilfe kann selbstverständlich durch zusätzliche Sprungmarken erweitert werden. Dies liegt im Ermessen des Webdesigners und sollte wohlüberlegt und zurückhaltend geschehen.

3 CSS-Bausteine

3.1 Das CSS-Konzept

Das CSS-Konzept von YAML basiert auf dem Baukasten-Prinzip sowie auf der Kaskade. Die CSS-Definitionen des Basislayouts sind nach Funktion in mehrere separate CSS-Bausteine (Dateien) aufgeteilt

- Positionierung der Hauptbereiche der Webseite (Header, Footer, Spalten)
- Screenlayout: Gestaltung der Hauptbereiche
- Formatierung der Inhalte
- Gestaltung von Navigationselementen
- Druckvorlagen

Das fertige Layout setzt sich anschließend aus mehreren dieser Bausteine zusammen. Diese Trennung nach Funktionen erleichtert die Bearbeitung und verbessert die Übersichtlichkeit.

Weiterhin erfolgt eine strikte Trennung von regulärem CSS und speziell für den Internet Explorer erforderlichen Anpassungen (Bugfixes für CSS-Bugs). Viele dieser Bugfixes für den Internet Explorer machen Gebrauch von den ebenso zahlreichen Parser-Bugs, die den Internet Explorer dazu veranlassen, ungültige oder fehlerhafte CSS-Deklarationen dennoch zu akzeptieren.

In den seltensten Fällen lassen sich jedoch bei Vermischung von regulärem CSS und den IE-Bugfixes validierende Stylesheets erzeugen. Zudem stören sie in der Regel Übersichtlichkeit im CSS. Eine Zusammenfassung der Anpassungen in eine spezielle Datei ermöglicht zugleich einen besseren Überblick bezüglich der einzelnen IE-Browserversionen, welche zum Teil unterschiedliche Hilfestellungen benötigen.

3.1.1 Einsatz der Kaskade

Neben der thematischen Gliederung der Stilzuweisungen in verschiedene CSS-Bausteine wird innerhalb von YAML an vielen Stellen intensiv von der CSS-Kaskade Gebrauch gemacht.

Über die Kaskade entscheidet der Browser, welche CSS-Eigenschaften für die Darstellung ein Element relevant ist. Diese Kaskade ist in 4 Stufen unterteilt:

- Stufe 1: Herkunft der Deklarationen (Browser-, Autoren-, Benutzer-Stylesheet).
- Stufe 2: Sortierung nach Ursprung und Gewicht
- Stufe 3: Sortierung nach Spezifität der Selektoren
- Stufe 4: Sortierung nach der Reihenfolge des Auftretens

Über die CSS-Grundbausteine (*base.css* und *ie hacks.css*) wird dem Seitenersteller ein dreispaltiges Basislayout als Grundlage für die Layouterstellung zur Verfügung gestellt. Diese Stylesheets werden in jedem YAML-basierten Layout eingebunden und bleiben immer unverändert.

Das Basislayout kann dann durch gezieltes Überschreiben von Stil-Deklarationen modifiziert und durch Ergänzung von Eigenschaften gestaltet werden. Alle Eingriffe (Gestaltung & Modifikation) des
YAML 3.0 | Dokumentation

Seitenerstellers erfolgen in gesonderten Stylesheets. Auf diese Weise bleibt YAML als stabile Basis auf unterster Ebene immer vorhanden.

3.2 Namenskonventionen

Innerhalb der Dokumentation, sowie bei der Benennung von Dateien und Verzeichnissen des Frameworks werden immer wieder bestimmte Bezeichnungen verwendet, deren Bedeutung hier kurz erläutert werden soll.

3.2.1 Grundbausteine (core-Dateien)

Die *core*-Dateien bilden den Kern - oder das Fundament - des YAML-Frameworks und befinden sich im Ordner *yaml/core/*.

Sie stellen die Funktionalität des Frameworks bereit und sind für die browserübergreifend einheitliche Darstellung des Layouts verantwortlich. Sie bilden daher die Grundbausteine jedes YAML-basierten Layouts und werden in jedem Fall benötigt.

3.2.2 Ergänzungsbausteine

YAML basiert auf der Ausnutzung der Kaskade. Die eigentliche Layoutgestaltung erfolgt über die *Modifikationen* des von YAML bereit gestellten Basislayouts. Hierfür stellt YAML einige vorgefertigte CSS-Bausteine (z.B. Navigationsbausteine) sowie Vorlagen für oft benötigte Bausteine (z.B. das Screenlayout) zur Verfügung. Diese Bausteine sind funktional gegliedert:

- Screenlayout - *screen/*
- Printlayout - *print/*
- Navigation - *navigation/*

Bei unverändertem Einsatz, sollten sie direkt aus dem Ordner *yaml/* in das Layout importiert werden. Eigene Stylesheets oder Modifikationen dieser Bausteine sollten hingegen im eigenen *css*-Ordner verwaltet werden.

3.2.3 Patches

Eine *Patch*-Datei fasst alle notwendigen CSS-Anpassungen für den Internet Explorer in einer CSS-Datei zusammen. Diese wird über einen *Conditional Comment* in den (X)HTML-Quelltext eingebunden und sorgt somit für die fehlerfreie Darstellung des Layouts.

3.2.4 Dateivorlagen

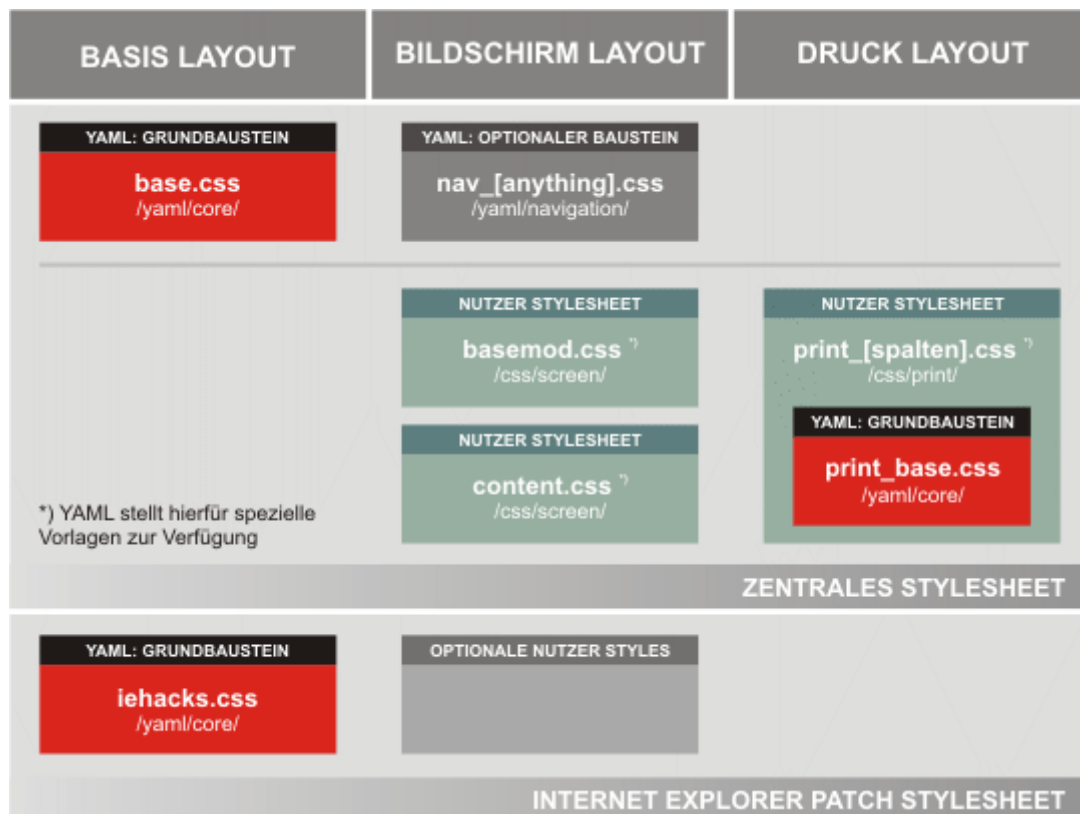
YAML bietet stellt für oft benötigte Bausteine Dateivorlagen zur Verfügung. Diese Vorlagen sind an der Endung des Dateinamens **_draft.** erkennbar.

Für die praktische Arbeit mit YAML kopieren Sie sich die Vorlagen, mit denen Sie arbeiten wollen, in Ihren *css*-Ordner und benennen sie um.

3.3 Das zentrale Stylesheet

Das CSS-Konzept von YAML basiert, wie gerade besprochen, auf dem Baukastenprinzip sowie auf der Kaskade. Die CSS Bausteine sind nach Funktion (Positionierung der Layoutelemente, Formatierung der Inhalte usw.) aufgeteilt.

Das folgende Diagramm zeigt die Funktion der einzelnen innerhalb des YAML-Frameworks verfügbaren Bausteine und deren Bedeutung.



Für jedes YAML-basierte Layout existiert ein solches zentrales Stylesheet, welches alle für das Layout benötigten Bausteine (Grundbausteine, Screenlayout, Navigation, Druckvorlagen) einbindet. Ein vollständiges Layout setzt sich also immer aus mehreren dieser Bausteine zusammen. Diese Trennung nach Funktionen erleichtert die Bearbeitung und verbessert die Übersichtlichkeit.

Wie erfolgt nun der praktische Umgang mit YAML?

3.3.1 Einbindung & Import der CSS-Bausteine

Auch der Aufbau des zentralen Stylesheets - und damit die Einbindung von YAML in eigene Projekte - lässt sich am einfachsten anhand eines Beispiels erläutern.

Hinweis: Kopieren Sie sich den Ordner `yaml` aus dem Download-Paket auf Ihren Server in die gleiche Verzeichnisebene, in der sich auch Ihr `css`-Ordner befindet. Diese Trennung zwischen Ihren eigenen CSS-Dateien und den Dateien des Frameworks sollten Sie beibehalten, um Updates einfach einspielen zu können.

Die Einbindung des Layouts in den (X)HTML-Quelltext erfolgt über ein sogenanntes *zentrales Stylesheet*, welches üblicherweise über das `link` Element im HTML-Header der Webseite integriert wird.

```
<head>
...
<link href="css/layout_3col_standard.css" rel="stylesheet"
type="text/css"/>
...
</head>
```

Dieses *zentrale Stylesheet* enthält Ihr Layout und sollte sich daher in Ihrem `css`-Ordner befinden. Innerhalb dieses Stylesheets werden die für das jeweilige Layout benötigten CSS-Bausteine über die `@import` Regel eingebunden.

```
/* import core styles | Basis-Stylesheets */
@import url(../yaml/core/base.css);

/* import screen layout | Screen-Layout einbinden */
@import url(../yaml/navigation/nav_shinybuttons.css);
@import url(screen/basemod.css);
@import url(screen/content.css);

/* import print layout | Druck-Layout einbinden */
@import url(../yaml/print/print_003_draft.css);
```

Wie Sie erkennen, wird zunächst das Basis-Stylesheet `base.css` des YAML-Frameworks eingebunden. Es wird direkt aus dem Ordner `yaml/core/` geladen.

Im zweiten Schritt wird das Screenlayout zusammengesetzt. Dabei wird in diesem Beispiel ein Stylesheet für die Navigation (`nav_shinybuttons.css`) geladen. Dieses soll unverändert verwendet werden, daher erfolgt wiederum der Zugriff direkt auf den Ordner `yaml`. Das Screenlayout und die Gestaltung der Inhalte legen Sie selbst fest. Daher sollten diese Dateien in Ihrem eigenen `css`-Ordner abgelegt werden.

Im dritten und letzten Schritt folgt der Einbau des Drucklayouts. Auch hierbei bietet YAML vorgefertigte Bausteine. In diesem Beispiel wird einer dieser CSS-Bausteine für die Bereitstellung eines Printlayouts (die Datei `print_003_draft.css`) ohne Anpassungen aus dem Ordner `yaml/print/` verwendet.

Wichtig: Das Grundprinzip, die Trennung der eigenen CSS-Dateien von den YAML-Dateien, bietet viele Vorteile und hat sich in der Praxis bestens bewährt.

Wollen Sie Änderungen an einzelnen Dateien des Frameworks vornehmen oder mit einer der Dateivorlagen aus dem Framework arbeiten, legen Sie sich eine Kopie der betreffenden Datei in Ihrem `css`-Ordner an und arbeiten Sie nicht mit dem Original.

Bausteine, die Sie unverändert einsetzen, importieren Sie hingegen direkt aus dem `yaml`-Ordner in Ihr Layout. Beim Einspielen von Updates des Frameworks müssen Sie dann nur noch den `yaml`-Ordner austauschen.

Anpassungen für den Internet Explorer

Mit der Einbindung des *zentralen Stylesheets* in den (X)HTML-Quelltext sind alle modernen Browser (Firefox, Safari, Opera, usw.) versorgt. Lediglich der Internet Explorer benötigt zusätzliche CSS-Anpassungen zur fehlerfreien Darstellung YAML-basierter CSS-Layouts. Diese werden im (X)HTML-Quelltext über einen so genannten *Conditional Comment* bereit gestellt.

```
<head>
...
<!--[if lte IE 7]>
<link href="css/patches/patch_3col_standard.css" rel="stylesheet"
type="text/css" />
<![endif]-->
</head>
```

Dies ist ein spezieller Kommentar, welcher ausschließlich vom Internet Explorer verstanden und ausgewertet wird. Er ermöglicht es, **nur** dem Internet Explorer ein weiteres, speziell auf das jeweilige Layout angepasstes Stylesheet zu übergeben. Im vorliegenden Beispiel ist dies die Datei `patch_3col_standard.css`, welche alle CSS-Hilfestellungen für den Internet Explorer liefert.

Näheres zu deren Funktion erfahren Sie im [Abschnitt 3.5: CSS-Anpassungen für den Internet Explorer](#). Für alle anderen Browser ist dies hingegen ein normaler HTML-Kommentar, daher ignorieren sie dessen Inhalt.

3.4 Das Basis-Stylesheet `base.css`

Wichtig: Das Stylesheet `base.css` aus dem Verzeichnis `yaml/core/` ist einer der Grundbausteine des YAML-Frameworks. Es stellt die grundlegende Funktionalität des Frameworks bereit (Browser-Reset, Clearing, Subtemplates usw.). Dieses Stylesheet ist in jedem YAML-basierten Layout erforderlich und sollte generell unverändert bleiben!

3.4.1 Browser Reset – Einheitliche Ausgangsbedingungen für alle Browser

Die Aufgabe von YAML ist, ein möglichst einheitliches, browserunabhängiges Erscheinungsbild des Layouts zu sichern.

Dazu ist es zunächst erforderlich, browserübergreifend für einheitliche Ausgangsbedingungen zu sorgen. Diese ist ohne Zutun seitens YAML nicht gegeben, denn jeder Browser hält gewisse Standard-Formatierungen vor (und natürlich jeder Browser andere), um auch unformatierte Inhalte möglichst lesbar auf dem Bildschirm darzustellen.

Also richten wir den Blick auf die ersten Zeilen des Basis-Stylesheets `base.css`:

```
/**
 * @section browser reset
 * @see    ...
 */

* { margin:0; padding:0; }
option {padding-left: 0.4em}
```

```

* html body * { overflow:visible }
* html iframe, * html frame { overflow:auto }
* html frameset { overflow:hidden }

html { height: 100%; margin-bottom: 1px; }
body {
    font-size: 100.01%;
    position: relative;
    color: #000;
    background: #fff;
    text-align: left;
}

fieldset, img { border:0 solid; }

ul, ol, dl { margin: 0 0 1em 1em }
li { margin-left: 1.5em; line-height: 1.5em; }

dt { font-weight: bold; }
dd { margin: 0 0 1em 2em; }

blockquote { margin: 0 0 1em 1.5em; }

```

Abstände und Rahmenbreiten zurücksetzen

Mit `* { margin:0; padding:0; }` werden die Innen- und Außenabstände aller HTML-Elemente (dafür sorgt der Stern-Selektor) auf Null gesetzt. Diese Lösung hat den Vorteil, auf einfache Weise wirklich alle HTML-Elemente zu erreichen.

Für `select` Elemente ergibt sich daraus ein kleines Problem. Die oben genannte Anweisung setzt auch das `padding` des `option` Elementes (den Auswahlbutton der Selectbox) auf Null, wodurch er unter Windows den letzten Buchstaben des Inhaltstextes überdeckt. Dieses Problem wird durch `option {padding-left: 0.4em}` behoben, indem der Standardwert wieder eingetragen wird.

Hinweis: Der Eintrag kann durch die CSS-Eigenschaft `* {border: 0;}` ergänzt werden. Dies sorgt jedoch dafür, dass die Browser die Vorformatierung von Formular-Elementen (Textarea oder Submit-Button) verlieren.

In diesem Fall müssen daher für diese Elemente in der CSS-Datei *content.css* (siehe [Abschnitt 3.8: Gestaltung der Inhalte](#)) entsprechende Standardvorgaben vorhanden sein, da sie sonst am Bildschirm schwer zu erkennen sind.

Weiter wird für die HTML-Elemente `fieldset` und `img` die Standardgröße des Rahmens (`fieldset, img { border:0 solid; }`) auf Null gesetzt.

Vermeidung des Italics-Bugs im IE

Dieser Bugfix für die Vermeidung des Italics-Bug des Internet Explorers 5.x und 6.0 stellt eine Ausnahme dar. Während alle weiteren CSS-Anpassungen für den Internet Explorer bei YAML in gesonderten Stylesheets zusammengefasst sind, muss dieser Bugfix vor allen layoutspezifischen CSS-Deklarationen platziert werden, um seine Wirkung zu entfalten.

```

* html body * { overflow:visible }
* html iframe, * html frame { overflow:auto }
* html frameset { overflow:hidden }

```

Eine genaue Beschreibung dieses Bugfixes finden Sie im [Abschnitt 3.5: CSS-Anpassungen für den Internet Explorer](#).

Springende, zentrierte Layouts im Firefox & Safari

Bei Seiten, die vollständig in den Viewport des Browserfenster passen, blenden der Firefox und auch der Safari-Browser die vertikalen Scrollbalken vollständig aus. Wird die Webseite plötzlich höher als die Größe des Viewports, werden vertikalen Scrollbalken wieder eingeblendet. Dieses Ein-/Ausblenden führt bei zentrierten Layouts zu einem unangenehmen seitlichen Versatz. Das Layout "springt" seitlich hin und her.

```
html { height: 100%; margin-bottom: 1px; }
```

Diese Deklaration erzwingt unabhängig von dem Inhalt der Webseite die Anwesenheit des vertikalen Scrollbalkens. Das Abschalten der Scrollbalken kann damit wirksam unterbunden werden.

Schriftgrößen und Rundungsfehler

Die Deklaration `body { font-size: 100.01% }` bewirkt einen Ausgleich von Rundungsfehlern, speziell in älteren Opera- und Safari-Versionen. Beide würden andernfalls zu kleine Schriftgrößen berechnen.

Hinweis: In früheren Versionen von YAML wurden auch Schriftgrößenkorrekturen für die Elemente `select`, `input` und `textarea` für den Safari 1.x Versionen vorgenommen. Diese führen jedoch in aktuellen Firefox 2.x zu Problemen und werden daher ab V3.0 nicht mehr verwendet. Der Safari 1.x ist mittlerweile hingegen kaum mehr verbreitet.

Standardvorgaben für Listen und Zitate

HTML-Listen sowie Elemente zur Kennzeichnung von Zitaten (`blockquote`, `ol`, `ul`, `dl`) benötigen eine Gestaltung in Form von Randabständen und Zeilenhöhen, um ein browserübergreifend einheitliches Erscheinungsbild zu gewährleisten. Die stark variierenden browsereigenen Deklarationen wurden im Rahmen der Schaffung einheitliche Ausgangsbedingungen durch die Deklaration `* {margin:0; padding:0;}` gelöscht.

```
ul, ol, dl { margin: 0 0 1em 1em }
li { margin-left: 1.5em; line-height: 1.5em; }

dt { font-weight: bold; }
dd { margin: 0 0 1em 2em; }

blockquote { margin: 0 0 1em 1.5em; }
```

3.4.2 Rohbau des Layouts

Die Datei *base.css* liefert für die Elemente der Quelltextstruktur die grundlegende Angaben zur Positionierung innerhalb des dreispaltigen Basislayouts. Hier noch einmal alle Elemente in der Übersicht:

- `#page_margins` und `#page` Beide Container können zur grafischen Gestaltung der Randbereiche des Layout verwendet werden. Im Regelfall wird über `#page_margins` die Layoutbreite bzw. die minimale und maximale Breite eines flexiblen Layouts festgelegt.
- `#header` Kopfbereich des Layouts, nimmt in der Regel ein Logo sowie den Container `#topnav` auf. Dieser ist per Default absolut in der rechten oberen Ecke des Headers positioniert und dient zur Aufnahme von Pflichtlinks (z.B. dem Impressum), Skip-Navigation usw.
- `#nav` Container für Hauptnavigation
- `#main` Hauptinhaltsbereich der Seite. Darin enthalten sind die drei Inhaltscontainer.
 - `#col1` & `#col1_content` - Erster *float*-Container des Hauptteils
 - `#col2` & `#col2_content` - Zweiter *float*-Container des Hauptteils
 - `#col3` & `#col3_content` - Statischer Container des Hauptteils, einschließlich des Containers `#ie_clearing` als Hilfestellung für die korrekte Darstellung im IE.
- `#footer` - Fußbereich des Layouts



Die CSS-Deklarationen beinhalten mit Ausnahme der Vorgabe von Standardbreiten für die beiden *float*-Container als linke und rechte Spalten keinerlei optische Vorgaben im Sinne eines Screenlayouts.

```
#header { position:relative }
#topnav { position:absolute; top: 10px; right: 10px; text-align: right }
#nav { clear:both; width: auto }
#main { clear:both; width: auto }

#col1 { float: left; width: 200px }
#col2 { float:right; width: 200px }
#col3 { width:auto; margin: 0 200px }

#footer { clear:both; display:block }
```

Erhöhung der Robustheit für Spalten-Container

Die Spalten-Container erhalten spezielle CSS-Eigenschaften zugewiesen, um die Robustheit des Basis-Layouts zu erhöhen.

```
#col1 {z-index: 3;}
#col2 {z-index: 5;}
#col3 {z-index: 1;}
#col1_content {z-index: 4;}
#col2_content {z-index: 6;}
#col3_content {z-index: 2;}

#col1_content, #col2_content, #col3_content { position: relative; }
```

Die Vergabe des `z-index` in dieser Reihenfolge bewirkt, dass so das Überdecken der Inhalte der statischen Spalte `#col3` durch die *float*-Spalten verhindert wird, falls der im [Abschnitt 5.3: Bekannte Probleme](#) beschriebene Bug im IE 5.x auftreten sollte.

Ein weiterer Grund für die Vorgabe der `z-index` Werte ist rein praktischer Natur. Normalerweise werden die DIV-Container vom Browser in der Reihenfolge gerendert, in der sie im Quelltext stehen. Für die volle Funktionalität ist jedoch wichtig, dass die statische Spalte - die im Quelltext immer nach den *float*-Spalten kommt - zuerst gerendert wird und ggf. von den Randspalten überlappt wird. Bei der Gestaltung der Spalten (siehe [Abschnitt 4.2](#)), kann diese `z-Index`-Sortierung später ausgenutzt werden.

Die Eigenschaft von `position: relative;` ist es eine vorbereitende Maßnahme, um bei der späteren Verwendung ggf. Elemente auch absolut innerhalb der Spalten positionieren zu können. Gleichzeitig wird damit bewirkt, dass die Spalteninhalte im Internet Explorer nicht plötzlich verschwinden bzw. erst nach Markierung oder Änderung der Fenstergröße sichtbar werden.

Hinweis: Im YAML-Framework sind Standard-CSS für moderne Browser und alle für den Internet Explorer *zusätzlich* erforderlichen CSS-Anpassungen voneinander getrennt.

Die Datei *ie hacks.css* beinhaltet alle layoutunabhängigen Anpassungen für diesen Browser. Sie ist neben der Quelltextstruktur und der *base.css* der dritte Grundbaustein des YAML-Frameworks (siehe [Abschnitt 3.5: CSS-Anpassungen für den Internet Explorer](#)).

3.4.3 Weitere Bestandteile

Generische CSS-Klassen zur Layoutgestaltung

YAML bietet ab Version 3.0 eine zweite, optional einsetzbare Möglichkeit zur einfachen Modifizierung der Spaltenanzahl im Basislayout an. Neben der klassischen Methode, wie sie in [Abschnitt 4.4: Freie Spaltenanordnung](#) beschrieben wird, kann das Basislayout zusätzlich über folgende Standardklassen modifiziert werden.

```
/**
 * @section Generische Klassen zur Layoutumschaltung
 * @see      ...
 *
 * .hidenone   -> show all columns
 * .hideleft  -> 2-column-layout (using #col2 and #col3)
 * .hideright -> 2-column-layout (using #col1 and #col3)
 * .hidenone  -> single-column-layout (using #col3)
 */

.hidenone #col3 {margin: 0 200px}
.hideboth #col3 {margin-left: 0; margin-right: 0}
```

```
.hideleft #col3 {margin-left: 0; margin-right: 200px}
.hideright #col3 {margin-left: 200px; margin-right: 0}

.hideboth #col1, .hideboth #col2 {display:none}
.hideleft #col1 {display:none}
.hideright #col2 {display:none}
```

Für den praktischen Einsatz sollte die jeweils benötigte Klasse entweder an das `body` Element, Container `#page_margins` oder an den Container `#main` vergeben werden.

Diese Klassen müssen selbstverständlich für die praktische Anwendung auf die gewünschten Spaltenbreiten des Screenlayouts angepasst werden. Sie dienen daher an dieser Stelle in erster Linie der Orientierung bzw. liefern wiederum Standardwerte.

Hinweis: Der Einsatz dieser Klassen zur Modifizierung des Layouts ist besonders in Verbindung mit Content-Management-Systemen sinnvoll. Bei vielen CM-Systemen ist der Zugang zum Header-Bereich eines Templates schwer oder unmöglich, sodass der Austausch eines Stylesheets zur Modifikation des Layouts unmöglich oder zumindest kompliziert ist. Alternative Versionen des Grundlayouts bedingen daher meistens eigene Templates.

Dagegen ist eine Manipulation der HTML-Elemente innerhalb von `body` in der Regel unkompliziert. Über die hier definierten generischen Klassen kann ein Template daher sehr einfach manipuliert werden.

Skip-Links und Unsichtbare Inhalte

Im Sinne der Gestaltung möglichst barrierearmer Webseiten stellt YAML über die *base.css* vordefinierte CSS-Klassen zur Verfügung, um Inhalte in der regulären Darstellung am Bildschirm auszublenden, sie jedoch gleichzeitig für den Ausdruck der Webseiten oder auch alternative Ausgabemedien (z.B. Screenreader) zugänglich zu machen.

```
/**
 * @section Versteckte Elemente
 * @see    ...
 *
 * (en) skip links and hidden content
 * (de) Skip-Links und versteckte Inhalte
 */

.skip, .hideme, .print {
  position: absolute;
  top: -1000em;
  left: -1000em;
  height: 1px;
  width: 1px;
}

.skip:focus, .skip:active {
  position: static;
  top: 0;
  left: 0;
  height: auto;
  width: auto;
}
```

Die Definition der CSS-Klasse `.skip` für die Bereitstellung von vordefinierten Sprungmarken wurde bereits in [Abschnitt 2.8: Skip-Link-Navigation](#) erläutert.

Daneben werden jedoch noch weitere Standardklassen für das Verstecken von Inhalten (bzw. für die Bereitstellung von Zusatzinformationen für z.B. Screenreader) definiert. Die CSS-Klasse `.hideme` bewirkt ein generelles Verstecken von Inhalte für alle visuellen Ausgabemedien.

Die CSS-Klasse `.print` erlaubt ein Verstecken von Inhalte im Screenlayout, erlaubt aber die Ausgabe auf Printmedien. Das Sichtbarmachen dieser Klasse erfolgt in der Datei `print_base.css` deren Aufgaben im [Abschnitt 3.9: Das Drucklayout](#) erläutert werden. Beide Klassen sind barrierefrei gestaltet, sodass die Inhalte für Screenreader jederzeit zugänglich sind.

Weitere Deklarationen

Das Basis-Stylesheet `base.css` enthält noch weitere Deklarationen, die für die browserübergreifend einheitliche Darstellung YAML-basierter Layouts erforderlich sind, auf welche jedoch im Rahmen der Dokumentation an anderer Stelle detailliert eingegangen wird.

Methoden zum Markupfreien Clearing

Enthält die für die korrekte Darstellung der float-Umgebungen erforderlichen Clearing-Lösungen Clearfix und Overflow (siehe Abschnitt 2.6: Die Funktionsweise von floats)

Subtemplates

Klassendefinitionen für die Subtemplates (siehe [Abschnitt 4.5: Subtemplates](#))

3.5 CSS-Anpassungen für den Internet Explorer

Der Internet Explorer bietet seit der Version 5 eine umfassende Unterstützung für CSS 1 und eine gute Unterstützung für CSS 2. Insbesondere die CSS 2-Unterstützung enthält jedoch zahlreiche Fehler, die bei Nichtbeachtung schnell zu Darstellungsfehlern in CSS-Layouts führen können.

Die [Quelltextstruktur](#) des YAML-Basislayouts ist so angelegt, dass sich mit auf dieser Basis mit Hilfe von CSS eine Vielzahl möglicher Layoutvariationen erstellen lassen, ohne dass Änderungen im zugrunde liegenden Markup erforderlich sind. Um diese große Flexibilität gewährleisten zu können, müssen die zahlreichen CSS-Bugs des Internet Explorers zuverlässig abgefangen werden.

Das Auftreten von CSS-Bugs des Internet Explorers richtet sich nach dem Vorhandensein bestimmter Quelltext-Strukturen in Verbindung mit bestimmten Kombinationen aus floatenden, positionierten oder statischen Elementen. Aufgrund der im YAML-Framework fest vorgegebenen Quelltext-Struktur und der bekannten Modifikationsmöglichkeiten (Spaltenanordnungen) ist das Auftreten der meisten CSS-Bugs berechenbar und damit sehr gut beherrschbar.

Der Umgang mit diesen Bugs richtet sich nach der Art des Auftretens, wobei hier innerhalb des YAML-Frameworks zwei Fälle unterschieden werden:

Struktur- bzw. layoutunabhängige Bugfixes

Für den überwiegenden Teil der CSS-Bugs lässt sich aufgrund der bekannten Quelltextstruktur sehr einfach beheben. Ist ein solcher Bugfix auch noch in allen möglichen

Modifikationen (Spaltenanordnungen) des Basislayouts ohne störende Nebenwirkungen einsetzbar, so wird dieser als *struktur-* bzw. *layoutunabhängig* bezeichnet.

Alle *struktur-* bzw. *layoutunabhängigen* Bugfixes werden bei YAML in einem speziellen Stylesheet, der Datei *ie hacks.css* im Verzeichnis *yaml/core/* zusammengefasst und sollte immer unverändert bleiben.

Struktur- bzw. layoutabhängige Bugfixes

Ein geringer Teil der CSS-Bugs wird nicht in jedem Fall ausgelöst, bzw. es gibt keinen Bugfix, der unabhängig vom jeweils umgesetzten Layout immer funktioniert. Diese Bugfixes müssen durch den Seitenersteller in Abhängigkeit des von ihm umgesetzten Layouts ggf. angepasst werden. Sie werden daher als *struktur-* bzw. *layoutabhängig* bezeichnet.

Hierzu zählen weiterhin all jene Bugfixes, welche die Korrektur der fehlerhaften Darstellung von Inhaltselementen betreffen. Da YAML die Inhalte nicht kennt, muss hier der Seitenersteller diese Bugfixes generell selbst ergänzen.

Zu jedem YAML-basierten Layout sollte daher Anpassungsdatei *patch_[layout].css* für den Internet Explorer angelegt werden, wobei sich der Platzhalter *[layout]* im Dateinamen, der besseren Übersicht wegen, am Namen des zugehörigen zentralen Stylesheets orientieren sollte. Eine Vorlage für ein solches Anpassungs-Stylesheet, die Datei *patch_layout_draft.css* befindet sich im Verzeichnis *yaml/patches/*.

3.5.1 Aufbau der CSS-Anpassungsdatei für den Internet Explorer

Wie eben erläutert, gehört zu jedem YAML-basierten Layout (bzw. zu jedem zentralen Stylesheet, siehe [Abschnitt 3.3](#)) eine IE-Anpassungsdatei *patch_[layout].css*. Der Aufbau eines solchen Stylesheets soll im Folgenden erläutert werden. Hierzu ein Blick in die Vorlagedatei *patch_layout_draft.css* aus dem Verzeichnis *yaml/patches/*.

```
/* Layout independent adjustmenst | Layout-unabhängige Anpassungen ---- */
@import url(/yaml/core/ie hacks.css);

/* Layout dependent adjustments | Layout-abhängige Anpassungen ----- */
@media screen
{
    /* add your adjustments here | Fügen Sie Ihre Anpassungen hier ein */
    ...
}
```

Wie Sie erkennen, werden in dieser Datei sowohl layoutunabhängige- als auch layoutabhängige CSS-Anpassungen zusammengefasst. Dies hat den Vorteil, dass Sie nur diese eine CSS-Datei in das Layout integrieren brauchen.

Im ersten Teil erfolgt der Import der Datei *ie hacks.css* aus dem *core/* Verzeichnis des YAML-Frameworks. Wie bereits erwähnt, enthält diese Datei alle *struktur-* und *layoutunabhängigen* Bugfixes und kann daher unverändert in jedes YAML-basierte Layout eingebunden werden.

Im zweiten Teil finden Sie eine inhaltsleere `@media` Regel. Ab hier können Sie weitere, ggf. vorgefertigte IE-Stylesheets einbinden (z.B. für den Navigationsbaustein `nav_vlist`). Weiterhin ist dies der Platz, an dem Sie *struktur- bzw. layoutabhängige* Bugfixes bzw. Bugfixes für die korrekte Darstellung von Inhaltselementen ergänzen können.

Dieses IE-Anpassungsstylesheet übernimmt damit ähnliche Aufgaben, wie das zentrale Stylesheet. In ihm werden letztlich sämtliche CSS-Anpassungen zusammengefasst und an den Internet Explorer übergeben.

Einbindung der CSS-Anpassungen in das YAML-Layout

Viele Bugfixes stützen sich auf die Ausnutzung der ebenso zahlreich vorhandenen Parser-Bugs - insbesondere der älteren Versionen des Internet Explorers. Der daraus entstehende CSS-Code ist nicht immer valide und sollte daher ausschließlich dem IE zugänglich gemacht werden. Dies erfolgt über einen Conditional Comment innerhalb des HTML Headers `<head>..</head>`. Dies wurde am [Ende des Abschnitts 3.3: Das zentrale Stylesheet](#) bereits erwähnt.

```
...
<!--[if lte IE 7]>
  <link href="css/patches/patch_col3_standard.css" rel="stylesheet"
  type="text/css" />
<![endif]-->
</head>
```

Die Bedingung `lte IE 7` bedeutet: "kleiner oder gleich Internet Explorer Version 7.0". Dieser spezielle Kommentar wird nur vom Internet Explorer erkannt und die darin befindlichen Anweisungen ausgeführt. Für alle anderen Browser handelt es sich um einen normalen HTML-Kommentar dessen Inhalt ignoriert wird.

Nachfolgend werden alle für den Layoutprozess relevanten CSS-Bugs des Internet Explorers kurz erläutert und die Integration entsprechender Bugfixes/Workarounds im Rahmen des YAML-Frameworks beschrieben.

3.5.2 Struktur- und layoutunabhängige Bugfixes

Alle *struktur- und layoutunabhängigen* Bugfixes für CSS-Bugs des Internet Explorers werden in der Datei `ie hacks.css` im Ordner `yaml/core/` zusammengefasst.

Wichtig: Das Stylesheet `ie hacks.css` aus dem Verzeichnis `yaml/core/` ist einer der Grundbausteine des YAML-Frameworks. Es liefert alle struktur- und layoutunabhängigen Bugfixes für den Internet Explorer (Versionen 5.x/Win - 7.0/Win). Diese Korrekturen sind essentiell für die Robustheit und fehlerfreie Darstellung YAML-basierter Layouts im Internet Explorer. Dieses Stylesheet ist in jedem YAML-basierten Layout erforderlich und sollte generell unverändert bleiben!

Grundlegende CSS-Anpassungen

In der `base.css` werden zu Beginn einige Deklarationen eingefügt, die den Firefox- und Safari-Browser dazu zwingen sollen, generell vertikale Scrollbalken darzustellen. Im Internet-Explorer sind diese Zwangsmaßnahmen nicht erforderlich, da die Scrollbalken im IE immer sichtbar sind.

```
html { height: auto; margin-bottom:0; }
```

Die nächste Deklaration betrifft in erster Linie den Internet Explorer 7. Dieser hat Schwierigkeiten beim Seitenzoom, YAML-basierter Layouts.

```
body { position:relative }
* html body { position:static}
#main{ position:relative }
```

Die relative Positionierung von `body` beseitigt die Zoom-Schwierigkeiten des IE7 weitestgehend. Zusätzlich wird auch `#main` mit dieser Eigenschaft ausgestattet. Dies ist hilfreich zur Vermeidung fehlerhafter Spaltenpositionierung infolge der Größenänderung des Browserfensters beim Einsatz der JS-Expressions im IE6.

Clearing-Methoden für IE anpassen

Die CSS-Anpassungen für das Clearfix-Clearing basieren auf den Erkenntnissen von [Roger Johansson](#) und berücksichtigen bereits den IE 7.

```
/* Anpassungen für Clearfix-Methode */
.clearfix { display: inline-block }
.clearfix { display: block }
* html .clearfix { height: 1% }

/* Anpassungen für Overflow-Methode */
* html .floatbox { width:100% }
```

Der zweite Teil betrifft die CSS-Klasse `.floatbox`, durch welche die Overflow-Methode in YAML integriert ist. Die älteren IE-Versionen (5.x und 6.0) werden durch die Breitenangabe mit der Eigenschaft `hasLayout` versorgt, wodurch diese Clearing-Methode auch in diesen Browsern benutzbar wird.

Robustheit des Layouts erhöhen

Zahlreiche CSS-Bugs lassen sich durch die Aktivierung der proprietären Eigenschaft `hasLayout` des Internet Explorers sehr einfach beseitigen. Für einige, der in der Quelltextstruktur vordefinierten Container kann dieser Bugfix ohne quasi vorsorglich angewandt werden.

```
#page_margins, #page, #header, #nav, #main, #footer { zoom:1 }
#page_margins, #page { height: 1% }
* html #header, * html #nav, * html #main, * html #footer { width: 100% }
* html #header, * html #nav, * html #main, * html #footer { wid\th: auto }
```

Die beiden das Layout umspannenden Container `#page_margins` und `#page` erhalten `hasLayout` über die Eigenschaft `zoom:1` (IE6 & 7) bzw. `height:1%` (IE 5.x) zugewiesen. Auf die Verwendung der Eigenschaft `width` wurde an dieser Stelle bewusst verzichtet. Da die Datei `ie hacks.css` als letzte im Browser importiert wird, könnten hierdurch gestalterische Festlegungen des Seitenerstellers im Layout wieder aufgehoben werden.

Bei den inneren Containern wiederum ist die Verwendung von `height` problematisch, falls Container mit fixer Höhe erstellt werden sollen. Für den IE6 wird daher auf die proprietäre Eigenschaft `zoom` zurückgegriffen. Die Verwendung von `zoom:1` hat keinerlei störende Nebenwirkungen. Zur Unterstützung des IE5.x wird der Box-Modell-Bug ausgenutzt, wodurch die Deklaration `width:100%`

problemlos verwendet werden kann. Der IE 5.0 kennt die Eigenschaft `zoom` noch nicht, weshalb diese zusätzliche Deklaration notwendig wird.

Unvollständige Darstellung der Spalteninhalte vermeiden

```
* html #col1 { position:relative }
* html #col2 { position:relative }
* html #col3 { position:relative }
```

Ein weiterer Workaround hilft Darstellungsprobleme in älteren IE-Versionen zu vermeiden. Im IE5.x und IE6.0 kann es vorkommen, dass Inhalte nur teilweise oder gar nicht dargestellt werden. Die relative Positionierung der Spaltencontainer beseitigt dieses Problem.

Nach diesen allgemeinen Vorsichtsmaßnahmen, soll es im Folgenden um den Umgang mit den wichtigsten bekannten CSS-Bugs und deren Beseitigung gehen.

Escaping Floats Bug

	IE 5.x/Win	IE 6.0	IE 7.0
Bug aktiv	Ja	Ja	Ja

Beim [Escaping Floats Bug](#) positioniert der Internet Explorer *floats* innerhalb eines DIV-Containers fehlerhaft. Zwei Probleme treten dabei auf. Erstens wird die Größe des umgebenden DIV-Containers falsch berechnet und zweitens laufen die *floats* nach rechts aus dem Container heraus.

Beide Probleme lassen sich durch die Aktivierung von *hasLayout* z.B. mittels `height:1%` lösen. Innerhalb des Basislayouts ist ein entsprechender Bugfix bereits unter dem Punkt "Robustheit des Layouts erhöhen" integriert, sodass dieser Bug nicht mehr auftreten kann. Es sind daher keine weitergehenden Maßnahmen erforderlich.

Guillotine Bug

	IE 5.x/Win	IE 6.0	IE 7.0
Bug aktiv	Ja	Ja	Ja*

Für den [IE/Win Guillotine Bug](#) des Internet Explorers existieren zahlreiche Auslöskriterien, insbesondere Hover-Effekte bei Hyperlinks. Er ist sicherlich der mit Abstand bekannteste CSS-Bug des IE. Allerdings ist auch das sichere Vermeiden dieses Bugs nicht ganz einfach. Vermeiden lässt er sich fast nur, indem man — zumindest im Internet Explorer — auf Hover-Effekte verzichtet.

```
/* Guillotine Bug bei Änderung der Hintergrundfarbe von Links */
a, a:hover { background: transparent; }
```

Im IE7 soll dieser Bug eigentlich vollständig behoben sein. Allerdings wurde verschiedentlich - ohne genaue Testcases - von weiteren kollabierenden Abständen berichtet. Der Bugfix wird daher zur Sicherheit auch dem IE7 zugänglich gemacht.

Double Float-Margin Bug

	IE 5.x/Win	IE 6.0	IE 7.0
Bug aktiv	Ja	Ja	Nein

Bei der Positionierung von Float-Containern mit seitlichen Margins verdoppelt der Internet Explorer die Werte dieser Margins ("[Doubled Float-Margin Bug](#)") und sorgt daher gelegentliche Layout-Probleme, speziell bei der [freien Anordnung der Contentspalten](#).

Bugfix: Der Bug ist glücklicherweise recht einfach zu beheben. Den beiden *float*-Spalten `#col1` und `#col2` wird dazu die Eigenschaft `display:inline` zugewiesen. Diese Eigenschaft wird bei *float*-Objekten von allen modernen Browsern ignoriert und sorgt dafür, dass der Internet Explorer 5.x und 6.0 die Margins korrekt darstellen.

```
...
* html #col1 { display: inline; }
* html #col2 { display: inline; }
...
```

Expandierende Boxen im Internet Explorer

	IE 5.x/Win	IE 6.0	IE 7.0
Bug aktiv	Ja	Ja	Nein

Der Internet Explorer hat große Schwierigkeiten im Umgang mit übergroßen Inhalten innerhalb von Boxen mit festgelegter Breite. Siehe dazu [Internet Explorer and the Expanding Box Problem](#).

Bugfix: Zur Beseitigung der Schwierigkeiten beim Textumbruch des Internet Explorers und damit einer saubereren Darstellung des Layouts erzwingt man einen speziellen Textumbruch-Modus des Internet Explorers:

```
...
* html #col1_content { word-wrap: break-word }
* html #col2_content { word-wrap: break-word }
* html #col3_content { word-wrap: break-word }
...
```

Bei `word-wrap: break-word` handelt es sich um keine standardisierte CSS-Eigenschaft sondern um eine spezielle Eigenschaft, welche nur der Internet Explorer versteht. Sie erlaubt es dem Browser, Text nicht am Wortende sondern nach *jedem* Buchstaben umzubrechen. Dies beeinträchtigt zwar geringfügig die Lesequalität des Textes bei einer sehr kleinen Spaltenbreite, sichert dafür aber die Konsistenz des Layouts. Die älteren Versionen 5.x des Internet Explorers sprechen auf diesen Hack leider nicht an.

Übergroßen Inhaltselementen kann nur layoutabhängig begegnet werden. Hierfür werden weiter unten auf der Seite Vorschläge erläutert.

Internet Explorer und das Italics Problem

	IE 5.x/Win	IE 6.0	IE 7.0
Bug aktiv	Ja	Ja	Nein

Der [Italics-Bug](#) des Internet Explorers ist einer der am schwierigsten zu erkennenden und daher vermutlich auch einer der am wenigsten bekanntesten CSS-Bugs. Der IE erweitert die Breite eines Containers sobald Inhalte mittels `<i>` oder `` in Italics (also "schräg gestellte" Textpassagen) an den rechten Zeilenrand stoßen. Die CSS-Eigenschaft `font-style: italics` kann den Bug ebenso auslösen.

Die in diesem Fall stattfindende Erweiterung der Breite des Elterncontainers führt zu Problemen in *float*-basierten Layouts, da der entsprechende Container plötzlich nicht mehr in das Layout passt. Das Problem betrifft in erster Linie die statische Spalte `#col3`. In Verbindung mit dem Fehlen von *hasLayout* können statische Container sogar vollkommen ausgeblendet werden.

Bugfix: Der für dieses Problem bekannte Bugfix über die Zuweisung der CSS-Eigenschaft `overflow:visible`; ist sehr einfach. Damit er jedoch seine Wirkung voll entfalten kann, muss er möglichst allen Elementen einer Webseite zugewiesen werden. Dabei muss ein störender Einfluss auf den Layoutprozess vermieden werden. Aus diesem Grund wird der Bugfix in die *base.css* eingefügt, wo er **vor** jeglichen Layoutdefinitionen platziert wird.

```
* html body * { overflow:visible }
* html iframe, * html frame { overflow:auto }
* html frameset { overflow:hidden }
```

Obwohl der Wert *visible* den Standard der Eigenschaft `overflow` darstellt, und seine Vorgabe daher eigentlich überflüssig ist, behebt dies das Italics-Problem ab dem IE5.5+. Für den IE5.01 gibt es leider keine Lösung. Allerdings ist dieser Browser auch kaum noch anzutreffen.

Zusätzlich müssen jedoch noch für den IE5.x und 6.0 noch Korrekturen vorgenommen werden, um eine korrekte Darstellung von `textarea`- und `input`-Elementen sicherzustellen. Diese werden in der *ie hacks.css* definiert:

```
* html textarea {overflow:scroll; overflow-x: hidden}
* html input {overflow: hidden}
```

Disappearing List Background Bug

	IE 5.x/Win	IE 6.0	IE 7.0
Bug aktiv	Ja	Ja	Nein

Der [Disappearing List-Background Bug](#) des Internet Explorers wird ausgelöst, sobald Listen innerhalb von floatenden DIV-Containern platziert werden. Innerhalb von YAML betrifft dies vorrangig Listen innerhalb der *float*-Spalten `#col1` und `#col2` sowie jede Liste innerhalb floatender Inhaltselemente. Der Bug sorgt dafür, das Hintergrundfarben oder -grafiken von Listenelementen gar nicht oder nur teilweise dargestellt werden.

Bugfix: Zur Beseitigung des Bugs wird den Listen die Eigenschaft `position:relative` zugewiesen. Dies hat im Allgemeinen keine Auswirkungen auf das Layout und beseitigt den Bug zuverlässig.

```
...
* html ul { position: relative }
* html ol { position: relative }
* html dl { position: relative }
...
```

List Numbering Bug

	IE 5.x/Win	IE 6.0	IE 7.0
Bug aktiv	Ja	Ja	Ja

Der *List Numbering Bug* des Internet Explorers ist der letzte Bug in der Liste der struktur- und layoutunabhängig zu behebbaren CSS-Bugs. Er wird ausgelöst, sobald Listenelemente innerhalb geordneter Listen das Merkmal *hasLayout* erhalten. In diesem Fall versagt in allen IE-Versionen die korrekte Nummerierung der Listenelemente.

Bugfix: Zur Beseitigung des Bugs wird den Listenelementen die Eigenschaft `display:list-item` zugewiesen. Dies hat im Allgemeinen keine Auswirkungen auf das Layout und beseitigt den Bug zuverlässig.

```
body ol li { display:list-item; }
```

Über den Zusatz `body` im Selektor wird die Spezifität des Bugfixes erhöht.

3.5.3 Struktur- und layoutabhängige Bugfixes

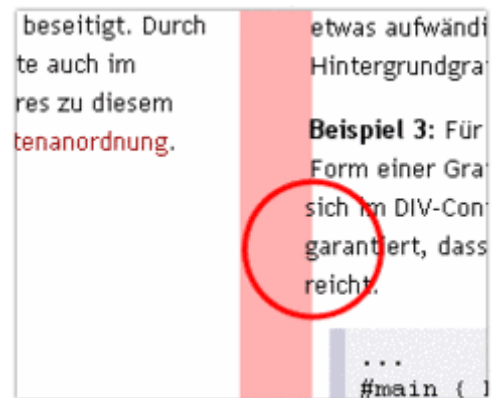
Wie bereits im [einleitenden Abschnitt zu den IE-Anpassungen](#) erläutert, können nicht alle Bugfixes struktur- und layoutunabhängig formuliert werden, sodass sie in jedem mit YAML umsetzbaren Layout funktionieren. Diese Bugfixes müssen vom Seitenersteller an die jeweiligen Randbedingungen angepasst werden.

Weiterhin zählen hierzu all jene Bugfixes, welche die Korrektur der fehlerhaften Darstellung von Inhaltselementen betreffen. Da YAML die Inhalte nicht kennt, müssen auch diese durch den Seitenersteller selbst ergänzt werden. All diese Bugfixes sollten in das zum Layout gehörende IE-Anpassungsstylesheet *patch_[layout].css* eingefügt werden.

3-Pixel-Jog Bug

	IE 5.x/Win	IE 6.0	IE 7.0
Bug aktiv	Ja	Ja	Nein

Problemstellung: Sobald sich links neben der statischen Container `#col3` ein floatender Container befindet, tritt der [3-Pixel-Jog Bug](#) des Internet Explorers auf. Wenn der Inhalt der statischen Spalte `#col3` länger als der Inhalt der daneben befindlichen *float*-Spalte wird, entsteht ein horizontaler Versatz der Inhalte in `#col3` um 3 Pixel. Die Auswirkungen sind im nebenstehenden Screenshot sichtbar.



Lösung: Die Vergabe der CSS-Eigenschaft `height: 1%` an `#col3` sorgt dafür dass der Bug verschwindet. Die Funktionsweise des Hacks beruht einmal mehr darauf, dem entsprechenden Container im Internet Explorer mit der Eigenschaft *hasLayout* zu belegen.

Allerdings korrigiert der IE dabei nicht den fehlerhaften Versatz, sondern verschiebt alle Elemente des Containers `#col3` um den gleichen Betrag von 3 Pixeln. Daher ist es anschließend möglich, diesen Versatz Hilfe zweier negativer Margins zu korrigieren. Diese Korrektur muss in Abhängigkeit der Spaltenanordnung erfolgen. Hier beispielhaft die Lösung für das Basislayout, mit Spaltenbreiten der *float*-Spalten von jeweils 200 Pixeln:

```
/* LAYOUT-ABHÄNGIGE ANPASSUNGEN -----*/
...
* html #col3 { height: 1%; }
* html #col1 {margin-right: -3px;}
* html #col2 {margin-left: -3px;}
* html #col3 { margin-left: 197px; margin-right: 197px; }
...
```

Hinweis: Die Anwendung dieses Bugfixes für alle sechs möglichen Spaltenanordnungen des Basislayouts wird in den Beispiellayouts im Ordner *examples/03_layouts_3col/* demonstriert.

Wichtig: Bei Anwendung dieses Bugfixes ist der Einsatz der *grafikfreien Spaltentrennlinien* nur noch eingeschränkt möglich. Sie reichen in diesem Fall nicht mehr in jedem Fall bis zum Footer.

In diesen Fällen müssen Sie auf die Technik "[Faux Columns](#)" zur Gestaltung von Spaltenhintergründen mittels Hintergrundgrafiken zurückgreifen.

Handhabung übergroßer Elemente

Das Expanding-Box-Problem des Internet Explorer 5.x und 6.0 wurde bereits angesprochen und ein entsprechender Bugfix für einen flexibleren Textumbruch in die Datei iehacks.css integriert. Was noch fehlt, sind Werkzeuge zum Umgang mit übergroßen Block-Elementen (Formulare, Tabellen, Bilder usw.).

Innerhalb flexibler Layouts können solche Elemente innerhalb von Spalten mit flexiblen Breiten im IE größere Probleme verursachen, da der IE zwanghaft den jeweiligen Elterncontainer aufweiten will, anstatt das diese Elemente über die Nachbarspalten hinweggleiten, wie in allen anderen Browsern.

YAML bietet hierfür zwei Lösungswege an. Zum einen können solche Elemente mit einem DIV-Container der Klasse `.floatbox` umschlossen werden. Wird das betreffende Inhaltselement zu breit für den Elterncontainer, so werden die überstehenden Bereiche abgeschnitten. Auf diese Weise werden Layoutprobleme umgangen.

Als Alternative hierzu steht die CSS-Klasse `.slidebox` zur Verfügung. Sie kann direkt an das betreffende übergroße Element vergeben werden und bewirkt, dass dieses über Nachbarbereiche des Layouts hinweggleitet, ohne dass der IE den Elterncontainer aufweitet und damit das Layout zerstört.

```
.slidebox {
  margin-right: -1000px;
  position: relative;
  height: 1%
}
```

Hinweis: Diese Klasse sollte nur statischen Elementen zugewiesen werden. Bei floatenden Elementen bewirkt der negative Margin einen ungewollten Versatz.

Fehlende Hintergründe von Block-Elementen

Der "Disappearing List-Background Bug" ist nicht der einzige Bug, der zur fehlerhaften Darstellung von Hintergrundfarben und -grafiken führt. Generell haben IE 5.x und IE 6.0 Probleme mit der Darstellung von Hintergrundgrafiken, bei Elemente mit der Eigenschaft `display: block`, solange `hasLayout` nicht aktiviert ist.

In der Regel müssen diese Anpassungen für Inhaltselemente durch den Seitenersteller ergänzt werden. Hierzu eignen sich beispielsweise die CSS-Eigenschaften `width`, `height`, oder `zoom` durch Vorgabe von konkreten Werten außer `auto`.

3.6 Erstellung des Screenlayouts

Mit der Erstellung des Screenlayouts beginnt die eigentliche Arbeit für den Seitenersteller. Die bisher vorgestellten CSS-Grundbausteine *base.css* und *ie hacks.css* übernehmen die Bereitstellung eines browserübergreifend fehlerfrei dargestellten Basislayouts, welches jedoch kein selbstständiges optisches Erscheinungsbild mitbringt.

Um diese Basis nicht zu gefährden, sollten Sie Ihre eigenen CSS-Deklarationen in einem gesonderten Stylesheet zusammenfassen. YAML stellt Ihnen auch hierfür passende Strukturen bereit, mit denen Sie arbeiten können aber nicht zwingend müssen.

3.6.1 Bestandteile des Screenlayouts

Die Erstellung des Screenlayouts gliedert sich grob in drei relativ unabhängige Teilgebiete:

- Gestaltung der Layoutelemente (Header, Footer, Inhaltsbereich)
- Gestaltung von Navigationselementen
- Gestaltung der Inhalte

Bei allen drei Teilgebieten können Sie wiederum auf Dateivorlagen zur eigenen Gestaltung bzw. vorgefertigte CSS-Bausteine des YAML-Frameworks zurückgreifen.

3.6.2 Gestaltung der Layoutelemente

Für die Gestaltung des Basislayouts, welches sich aus der Quelltextstruktur des Frameworks ergibt, finden Sie im Verzeichnis *yaml/screen/* eine inhaltsleere Gestaltungsvorlage mit dem Namen *basemod_draft.css*.

```
@media all
{
  /*-----*/

  /**
   * Gestaltung des YAML Basis-Layouts
   *
   * @section layout-basics
   */

  /* Randbereiche & Seitenhintergrund */
  body { ... }

  /* Layout: Breite, Hintergrund, Rahmen */
  #page_margins { ... }
  #page{ ... }

  /* Gestaltung der Hauptelemente des Layouts */
  #header { ... }
  #tovnav { ... }
  #main { ... }
  #footer { ... }

  /*-----*/
}
```

```

/**
 * Formatierung der Inhaltsbereiche
 *
 * @section layout-main
 */

#col1 { }
#col1_content { }

#col2 { }
#col2_content { }

#col3 { }
#col3_content { }

/*-----*/

/**
 * Gestaltung weiterer Layoutelemente
 *
 * @section layout-misc
 */

...
}

```

Diese Vorlage beinhaltet alle Elemente des Basislayouts. Sie können sich diese Vorlage kopieren und darin anfangen, die einzelnen Container nach Ihren Wünschen zu gestalten. Zusätzlich in das Layout integrierte Elemente sollten Sie am Ende der Datei ergänzen.

Selbstverständlich sind Sie aber auch bei dieser Arbeit nicht auf sich allein gestellt. Im Ordner *examples/* des Download-Paketes finden Sie zahlreiche, nach Themen geordnete Anwendungsbeispiele. Bei all diesen Beispielen kommt eine einheitliche Grundversion eines Screenlayouts zum Einsatz. Sie finden es im jeweiligen Verzeichnis *css/screen/* innerhalb der Beispielthemen in Form der CSS-Datei *basemod.css*.

In diesen zahlreichen Beispielen werden die verschiedenen Modifikationsmöglichkeiten des YAML-Basislayouts demonstriert. Bei allen diesen Eingriffen und Anpassungen bildet diese Datei den Ausgangspunkt.

Hinweis: An dieser Stelle wird nur die grundlegende Vorgehensweise bei der Erstellung des Screenlayouts besprochen. [Kapitel 4](#) widmet sich ausführlich den umfangreichen Modifikationsmöglichkeiten des Frameworks und geht auf viele der beiliegenden Beispiele intensiver ein.

3.6.3 Gestaltung der Navigationselemente und des Inhalts

Bei diesen beiden Punkten haben Sie als Seitenersteller generell alle Freiheiten. Sie können hier von Grund auf selbst Hand anlegen oder CSS-Bausteine des YAML-Frameworks als Grundlage für die Gestaltung verwenden. Aufgrund des Umfangs der von YAML bereit gestellten CSS-Bausteine, werden diese in den nachfolgenden beiden Abschnitten ([Abschnitt 3.7: Bausteine für die Navigation](#) und [Abschnitt 3.8: Gestaltung der Inhalte](#)) besprochen.

3.6.4 Das Zusammensetzen des Layouts

Bisher wurden die einzelnen CSS-Bausteine des Frameworks, sowie der grundlegende Weg zur Erstellung des Screenlayouts besprochen. Nun müssen diese einzelnen Teile noch zu einem Ganzen zusammengesetzt werden. Hier kommt das *zentrale Stylesheet* ins Spiel.

Im [Abschnitt 3.3: Das zentrale Stylesheet](#) wird der Zusammenbau des Layouts anhand des Beispiels *3col_standard.html* aus dem Verzeichnis *examples/01_layouts_basics/* des Download-Paketes erläutert.

Fügen Sie alle CSS-Bausteine Ihres Layouts zusammen und binden Sie Ihr zentrales Stylesheet in Ihre Webseite ein. Vergessen Sie nicht, auch bereits das IE-Anpassungsstylesheet anzulegen, damit der Internet Explorer das Stylesheet *ie hacks.css* erhält, welches für die fehlerfreie Darstellung des Layouts zwingend erforderlich ist.

Steht das Screenlayout, dann können Sie sich um eventuell erforderliche CSS-Anpassungen für den Internet Explorer kümmern und diese in Ihrem IE-Anpassungsstylesheet nach Bedarf ergänzen.

3.7 Navigationsbausteine

Zum vollständigen Screenlayout gehören neben der Gestaltung der im Basislayout befindlichen Grundelemente selbstverständlich auch Navigationselemente. Auf Grund der Komplexität, welche diese Navigationsbausteine annehmen können, werden diese ebenfalls in einzelnen CSS-Dateien verwaltet. Die Einbindung in das Layout erfolgt dann wiederum über das *zentrale Stylesheet*.

Innerhalb des YAML-Frameworks stehen im Verzeichnis *yaml/navigation/* einige vorgefertigte Navigationsbausteine zur Verfügung.

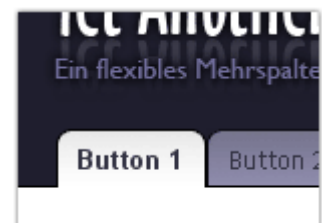
- Horizontale Listennavigation "Sliding Door II" — *nav_slidingdoor.css*
- Horizontale Listennavigation "Shiny Buttons" — *nav_shinybuttons.css*
- Vertikale Listennavigation — *nav_vlist.css*

Alle bereitgestellten Navigationsbausteine unterstützen die TAB-Navigation. Die Anwendung dieser Bausteine - speziell der Aufbau der zugrunde liegenden Quelltextstruktur und der zur Verfügung stehenden Klassen und IDs - wird im Folgenden kurz erläutert. Selbstverständlich sind Sie bei der Verwendung des YAML-Frameworks jedoch nicht an die Verwendung dieser Bausteine gebunden.

3.7.1 Sliding Door Navigation

Bei der ersten Listennavigation handelt es sich um eine Reiternavigation nach dem Vorbild [Sliding Door](#) (bzw. [Sliding Door II](#)) von [A-List-Apart](#). Zu beiden ALA-Artikeln gibt es deutsche Übersetzungen ([Teil 1](#) und [Teil 2](#)) von [Klaus Langenberg](#).

Es handelt sich dabei um eine einstufige horizontale Navigation mit grafischen Hover-Effekten der einzelnen Listenelemente. Die Funktion des Hover-Effekts ist jedoch auf standardkonforme Browser (Firefox, Safari, Opera bzw. IE7) beschränkt. Der Hover-Effekt wird im IE5.x und IE6.0 nicht unterstützt.



Das zugrunde liegende XHTML-Markup beider Varianten der Reiternavigation ist identisch und sehr einfach. Die Menüpunkte werden jeweils als ungeordnete Listen repräsentiert. Der aktive Menüpunkt wird durch die Vergabe der `id="current"` an das zugehörige Listenelement hervorgehoben. Alternativ kann zur Hervorhebung das Element `strong` verwendet werden, welches die Inhalte des aktiven Listenelementes einschließen muss. Beide Wege sind technisch gleichwertig.

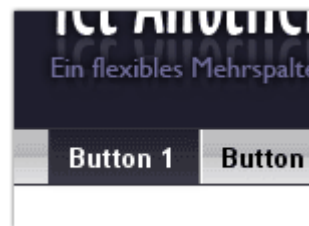
Hierzu ein Quelltextauszug, der den Aufbau des Markups verdeutlicht:

```
<div id="nav_main" >
  <ul>
    <li id="current"><strong>Button 1</strong></li>
    <li><a href="#">Button 2</a></li>
    <li><a href="#">Button 3</a></li>
    <li><a href="#">Button 4</a></li>
    <li><a href="#">Button 5</a></li>
  </ul>
</div>
```

examples/07_navigation/menu_slidingdoor.html

3.7.2 Shiny Buttons Navigation

Die *Shiny Buttons* Navigation verzichtet weitgehend auf grafische Elemente. Sie baut ebenfalls auf das gleiche XHTML-Markup auf, welches auch die zuvor erläuterte *Sliding Door Navigation* verwendet. Der Wechsel zwischen beiden Gestaltungsvarianten kann daher einfach durch den Austausch des CSS-Bausteins innerhalb des *zentralen Stylesheets* der Website erfolgen.



Die Menüpunkte werden als ungeordnete Liste organisiert. Der aktive Menüpunkt wird wiederum durch die ID `current` des zugehörigen Listenelementes hervorgehoben. Über den Außenabstand `margin-left: 50px` des Selektors `ul` wird die Ausrichtung des Menüs vom linken Rand gesteuert (Details, siehe `yaml/navigation/nav_shinybuttons.css`).

```
<div id="nav_main" >
  <ul>
    <li id="current"><strong>Button 1</strong></li>
    <li><a href="#">Button 2</a></li>
    <li><a href="#">Button 3</a></li>
    <li><a href="#">Button 4</a></li>
    <li><a href="#">Button 5</a></li>
  </ul>
</div>
```

examples/07_navigation/menu_shinybuttons.html

3.7.3 Vertikale Listennavigation

Bei dieser Navigation handelt es sich um eine vertikale Navigationsliste. Die Navigationsliste kann sowohl mit fixen Breite als auch mit flexiblen Breiten eingesetzt werden. Sie ermöglicht bis zu 4 Gliederungsebenen und erlaubt die Hervorhebungen des Menütitels (ID `title`).

Die Hervorhebung des aktiven Menüpunkts kann wahlweise über die ID `active` oder über die Auszeichnung des Inhalts mit `strong` innerhalb des Listenelementes erfolgen. Ebenso ist es möglich, gesonderte Zwischentitel für Menüpunkte untergeordneter Gliederungsebene zu generieren.

Hierfür wird das `span` Element verwendet. Alle Listenelemente enthalten einen Hover-Effekt für den Menüpunkt unter dem Mauszeiger.

Die Einbindung dieses Navigationsbausteins erfolgt über die Datei `nav_vlist.css` aus dem Ordner `yaml/navigation/`. Der zugehörige (X)HTML-Markup gestaltet sich wie folgt:

XHTML Struktur
Einführung
Aufbau des Quelltextes
Reihenfolge der Spalten
Funktionsweise von <i>floats</i>
Der Clou
Skiplinks

```
<ul id="submenu">
  <li id="title">Titel</li>
  <li><a href="#">Button 1</a></li>
  <li><a href="#">Button 2</a></li>
  <li><span>Ebene 3</span>
    <ul>
      <li><a href="#">Button 3.1</a></li>
      <li id="active">Button 3.2</li>
      <li><a href="#">Button 3.3</a></li>
    </ul>
  </li>
  <li><a href="#">Button 4</a></li>
  <li><a href="#">Button 5</a></li>
</ul>
```

examples/07_navigation/menu_vertical_listnav.html

Anpassungen für den Internet Explorer

Bei Verwendung dieses Navigationsbausteins muss die Datei `patch_nav_vlist.css` aus dem Ordner `yaml/patches/` in das zugehörige IE-Anpassungsstylesheet des Layouts importiert werden:

```
/* Layout-unabhängige Anpassungen -----*/
@import url(/yaml/core/ie hacks.css);
@import url(/yaml/patches/patch_nav_vlist.css); /* Box Modell Korrekturen */

/* Layout-abhängige Anpassungen -----*/
@media screen
{
  ...
}
```

Aufgrund der generell fehlerhaften Interpretation des Box-Modells im Internet Explorer 5.x (siehe [Abschnitt 2.4](#)) benötigt diese Navigation zur Unterstützung dieses Browsers spezielle Anpassungen.

```
...
* html #submenu li a,
* html #submenu li strong,
* html #submenu li span,
* html #submenu li#title,
* html #submenu li#active { width: 100%; width: 90%; }

* html #submenu li ul li a,
* html #submenu li ul li strong,
* html #submenu li ul li span,
* html #submenu li ul li#active { width: 100%; width: 80%; }
...
```

Für den Internet Explorer 5.x/Win wird auf diesem Wege die Breite der Listenelemente auf 100 Prozent gesetzt werden, um die fehlerhafte Interpretation des Box-Modells auszugleichen.

3.8 Gestaltung der Inhalte

YAML ist ein Layout-Framework und damit in erster Linie für die Bereitstellung einer spaltenbasierten Layoutstruktur verantwortlich, die unabhängig von den später eingefügten Inhalten korrekt dargestellt wird.

Die strukturelle, semantische und optische Aufbereitung der Inhalte obliegt in erster Linie Ihnen als Seitenersteller. Dennoch stellt YAML für die Gestaltung der Inhalte eine Gestaltungsvorlage in Form der Datei `content_default.css` im Verzeichnis `yaml/screen/` zur Verfügung. In dieser Vorlage werden grundlegende Formatierungen für Standard-Elemente bereit gestellt.

Diese Vorlage können Sie sich für Ihre Projekte kopieren, entsprechend Ihren Anforderungen ändern bzw. erweitern und über das zentrale Stylesheet in Ihr YAML-basiertes Layout integrieren.

3.8.1 Die Vorlage `content_default.css`

Auch die Inhalte einer Webseite bedürfen einer sorgfältigen Gestaltung. Jeder Browser hält hierfür ein eigenes Set von Standardformatierungen vor, wodurch es immer wieder zu kleinen und größeren Unterschieden in der Darstellung kommt.

Festlegung einer Basis-Schriftgröße

Der erste Schritt auf dem Weg zu einer einheitlichen Darstellung der Inhalte ist die Festlegung einheitlicher Schriftgrößen für alle Standardelemente. Um hier die unterschiedlichen Browservorgaben zurückzusetzen, werden in einem ersten Schritt über den `html *` Selektor sämtliche Schriftgrößendefinitionen für alle Elemente auf den Standardwert von üblicherweise 16 Pixel zurückgesetzt. Die unrunde Zahl dient dabei dem Ausgleich von Rundungsfehlern in einigen älteren Browsern.

Hinweis: Die Verwendung von `html *` an Stelle von `*` sichert, dass im Internet Explorer die Verwendung von Javascript-Expressions möglich ist, um die CSS-Eigenschaften `min-width` und `max-width` zu simulieren. Siehe [Abschnitt 4.7](#).

```
/* (en) reset font size for all elements to standard (16 Pixel) */
/* (de) Alle Schriftgrößen auf Standardgröße (16 Pixel) zurücksetzen */
html * { font-size: 100.01% }

/* (en) reset monospaced elements to font size 16px in Gecko browsers */
/* (de) Schriftgröße von monospaced Elemente auf 16 Pixel setzen */
textarea, pre, tt, code {
    font-family: "Courier New", Courier, monospace;
}

/* (en) base layout gets standard font size 12px */
/* (de) Basis-Layout erhält Standardschriftgröße von 12 Pixeln */
body {
    font-family: 'Trebuchet MS', Verdana, Helvetica, Arial, sans-serif;
    font-size: 75.00%
}
```

Im zweiten Teil folgt die Korrektur einer Eigenart aller Gecko-basierten Browser. Hierbei werden Elemente mit der Schriftart *monospace* (`textarea`, `pre`, `tt`, `code`) nicht auf die Standardgröße von

16px sondern nur auf 13px zurückgesetzt. Durch Vorgabe alternativer Schriftarten (*Courier New* oder *Courier*) kann dieses Problem vermieden werden und auch diese Elemente werden auf 16px zurückgesetzt.

Im Anschluss folgt die Wahl einer neuen, sinnvollen Standardschriftgröße für das `body` Element. Aufgrund der Vererbung dieser Eigenschaft, wird diese Festlegung für alle Elemente innerhalb von `body` übernommen. Als Basis wird eine serifenlose Schriftart mit einer Schriftgröße von 12 Pixeln gewählt.

Überschriften und Absatztexte

In einem nächsten Schritt werden auf Grundlage der Basis-Schriftgröße die Größen der Überschriften sowie Abstände und Zeilenhöhen für Absatztexte festgelegt.

```
h1,h2,h3,h4,h5,h6 { font-weight:bold; margin: 0 0 0.25em 0; }
h1 { font-size: 200% } /* 24px */
h2 { font-size: 166.67% } /* 20px */
h3 { font-size: 150% } /* 18px */
h4 { font-size: 133.33% } /* 16px */
h5 { font-size: 116.67% } /* 14px */
h6 { font-size: 116.67; font-style:italic } /* 14px */

p { line-height: 1.5em; margin: 0 0 1em 0 }
```

Wichtig: Generell sollte bei der Definition von Schriftgrößen darauf geachtet werden dass nur relative Maßeinheiten vergeben werden, um in allen Browsern den Textzoom zu ermöglichen.

Sobald Werte in der Einheit **Pixel** [px] vorgegeben werden, verlieren Nutzer des Internet Explorers (einschließlich der Version 7) die Möglichkeit, die Schriftgröße der Webseite über den Text-Zoom des Browsers entsprechend ihrer Lesegewohnheiten einfach anzupassen.

Gestaltung von HTML-Listen

Der nächste Block betrifft die Gestaltung von HTML-Listen. Die Vorgabewerte entsprechen den Angaben aus der *base.css*. Diese Redundanz ist an dieser Stelle gewollt. Anhand der Vorgaben, gehen Anpassungen deutlich leichter von der Hand.

```
/* ### lists | Listen ##### */

ul, ol, dl { line-height: 1.5em; margin: 0 0 1em 1em }
li { margin-left: 1.5em; line-height: 1.5em }

dt { font-weight: bold }
dd { margin: 0 0 1em 2em }
```

Textauszeichnungen

Zitate, Inhaltliche Hervorhebungen, Abkürzungen/Akronyme sowie vorformatierte Texte (bzw. Code-Auszüge) sind oft verwendete Textauszeichnungen und werden daher als solche an dieser Stelle mit grundlegenden Gestaltungsmerkmalen (Schriftart, Abstände, usw.) versehen.

```
/* ### text formatting | Textauszeichnung ### */

cite, blockquote { font-style:italic }
blockquote { margin: 0 0 1em 1.5em }
```

```
strong,b { font-weight: bold }
em,i { font-style:italic }

pre, code { font-family: monospace; font-size: 1.1em; }

acronym, abbr {
  letter-spacing: .07em;
  border-bottom: .1em dashed #c00;
  cursor: help;
}
```

Generische Klassen zur Auszeichnung und Positionierung von Inhalts-Elementen

```
/**
 * Generic Content Classes
 * (en) standard classes for positioning and highlightning
 * (de) Standardklassen zur Positionierung und Hervorhebung
 *
 * @section content-generic-classes
 */

.note {background: #dfd; padding: 1em; ...}
.important {background: #ffd; padding: 1em; ...}
.warning {background: #fdd; padding: 1em; ...}

.float_left {float:left; display:inline;
             margin-right:1em; margin-bottom:0.15em}
.float_right {float:right; display:inline;
              margin-left:1em; margin-bottom:0.15em}
.center {text-align:center; margin: 0.5em auto}
```

Für die Hervorhebung von Elementen entsprechend Ihrer inhaltlichen Relevanz (Allgemeine Information, Wichtiger Hinweis, Warnung) stehen drei CSS-Klassen zur Verfügung.

Für die horizontale Ausrichtung von Block-Elementen werden ebenfalls drei CSS-Klassen bereit gestellt. Hierbei wird zwischen links- bzw. rechtsbündiger Ausrichtung bzw. einer Zentrierung unterschieden.

Automatische Auszeichnung externer Hyperlinks

Die Auszeichnung externer Hyperlinks erfolgt automatisch per CSS. Dieses Vorgehen wird auf den eigentlichen Inhaltsbereich des Layouts, den Container `#main` begrenzt. Passen Sie dazu die URL-Bezeichnung an ihre eigene Domain an.

```
/**
 * External Links
 * (en) classification and formatting of hyperlinks via CSS
 * (de) Klassifizierung und Gestaltung von Hyperlinks mit CSS
 *
 * @section                content-external-links
 * @app-yaml-default        disabled
 */

/*
#main a[href^="http://www.my-domain.com"],
#main a[href^="https://www.my-domain.com"]
{
  padding-left: 12px;
}
```

```
background-image: url('your_image.gif');
background-repeat: no-repeat;
background-position: 0 0.45em;
}
```

Wenn Sie mit relativen Pfadangaben für seiteninterne Links arbeiten, können Sie sogar auf die Angabe der URL verzichten.

```
#main a[href^="http:"], a[href^="https:"] { ... }
```

Hinweis: Die Stildeklarationen zur automatischen Auszeichnung externer Links sind in der Vorlage zunächst auskommentiert und werden daher nicht automatisch verwendet.

Wichtig: Die automatische Linkauszeichnung erfordert die Unterstützung von CSS 2.1-Pseudoklassen durch den Webbrowser. Der Internet Explorer erfüllt bis zur aktuellen Version 7.0 diese Anforderungen leider nicht.

Gestaltung einfacher Tabellen

Der nächste Block beschäftigt sich mit der Darstellung einfacher Tabellen. Normale Tabellen werden mit automatischer Breite erzeugt, während durch die Klasse `.full` eine volle Breite erzwungen werden kann. Wichtig hierbei: Bei Verwendung dieser Klasse führen zusätzlich definierte seitliche Abstände oder Rahmen automatisch zu übergroßen Elementen.

Als zweite vordefinierte CSS-Klasse ermöglicht `.fixed` die Erstellung von Tabellen, deren Zellen bei übergroßen Inhalten nicht mitwachsen. Auf diese Weise lassen sich Tabellen leichter in flexible Layouts einbinden.

```
/**
 * Tables | Tabellen
 * (en) Generic classes for table-width ...
 * (de) Generische Klassen für die Tabellenbreite ...
 *
 * @section content-tables
 */

table { width: auto; border-collapse: collapse; margin-bottom: 0.5em; }
table.full { width: 100% }
table.fixed { table-layout: fixed }

th, td { padding: 0.5em; }
thead th { background: #444; color: #fff }
tbody th { background: #ccc; color: #333 }
tbody th.sub { background: #ddd; color: #333 }
```

Ansonsten sind die Vorgaben recht übersichtlich. Anhand der Unterscheidung zwischen `thead` und `tbody` sowie der Elemente `th` und `th.sub` können Spalten- und Zeilenüberschriften übersichtlich gestaltet werden.

Sonstiges

Hier findet sich zuletzt noch die Definition für eine 1 Pixel breite HR-Linie. Damit sind alle Formatanweisungen innerhalb der Datei `content_default.css` besprochen.

3.9 Anpassung des Layouts für Printmedien

Neben der Darstellung einer Webseite am Bildschirm ist die Aufbereitung der Seiteninhalte für den Druck ein wichtiger Punkt des Webdesigns. Bei diesem Übergang zwischen der Onlinewelt und dem Druckerpapier darf ein attraktives Screendesign kein Hindernis für eine übersichtliche und gut lesbare Druckfassung der Webseite sein.

Die Ausgabe der Seiteninhalte auf Papier bedeutet den Übergang von einem interaktiven zu einem passiven Medium. Papier ist ein Medium mit einem festen Seitenverhältnis und begrenzten Abmessungen. Es muss daher bei längeren Inhalten mit Seitenumbrüchen gerechnet werden - etwas, was man in der Onlinewelt nicht kennt. Hyperlinks sind auf Papier nicht mehr anklickbar, fehlt also auf dem Papier die zugehörige URL, so geht eine wichtige Information verloren.

3.9.1 Vorbereitungen für den Ausdruck

Die Überschrift trifft dabei den springenden Punkt nicht ganz. Im Detail müssen Sie sich vielmehr entscheiden, ob Sie die Inhalte aller Spaltencontainer oder nur die Inhalte bestimmter oder auch nur eines einzigen Containers ausdrucken möchten.

Dabei gilt es zunächst die Frage zu klären: *Welche Teile des Layout enthalten wichtige Informationen und was ist nur Beiwerk?*

Die Angaben der Fußzeile, Werbung in den Randspalten oder eine Suchmaske sind auf Papier nutzlos. Auch die Navigationselemente der Seite sind auf Papier nicht mehr bedienbar. Es ist daher nicht erforderlich, alle auf dem Bildschirm dargestellten Inhalte auch wirklich auszudrucken. In den Druck-Stylesheets werden daher die Fußzeile, sowie die Hauptnavigation abgeschaltet.

Auswahl der auszudruckenden Spaltencontainer

Innerhalb des YAML-Frameworks ist die Anordnung und damit auch die Nutzung der Spaltencontainer für Inhalte, Navigation oder sonstiges frei wählbar. Demzufolge wurden auch Vorbereitungen getroffen, sodass Sie jede Kombination der drei Spaltencontainer drucken lassen können.

Die Entscheidung darüber fällen Sie durch die Einbindung eines der sieben Print-Stylesheets aus dem Verzeichnis *yaml/print/* in das *zentrale Stylesheet* Ihres Layouts.

Druck-Stylesheet	#col1	#col2	#col3
print_100_draft.css	Ja	-	-
print_020_draft.css	-	Ja	-
print_003_draft.css	-	-	Ja
print_120_draft.css	Ja	Ja	-
print_023_draft.css	-	Ja	Ja
print_103_draft.css	Ja	-	Ja
print_123_draft.css	Ja	Ja	Ja

3.9.2 Aufbau der Print-Stylesheets

Der Aufbau dieser insgesamt 7 Print-Stylesheets ist weitestgehend identisch. Der Großteil der Vorbereitungen für die Druckausgabe einer Webseite hängt nicht von der Wahl der Spalten ab.

Hierbei geht es vielmehr um die Anpassung des Screenlayouts an das Medium Papier, das Abschalten nicht benötigter Layoutelemente oder auch die Auszeichnung von URL's, Abkürzungen oder Akronymen, damit so wenig wie möglich Informationen verloren gehen. Alle diese Dinge sind unabhängig davon, welche Spaltencontainer Sie ausdrucken möchten. Daher werden diese Deklarationen innerhalb von YAML in der Datei *print_base.css* im Verzeichnis *yaml/core/* zusammengefasst und zu Beginn jedes der sieben Print-Stylesheets importiert.

```
/* import print base styles | Basisformatierung für Drucklayout einbinden
*/
@import url(../core/print_base.css);
```

Auf diese Weise stehen diese allgemeinen Anpassungen automatisch jedem Print-Stylesheet zur Verfügung.

In den Print-Stylesheets selbst erfolgen hingegen nur noch das Abschalten der nicht zum Druck ausgewählten Spaltencontainer sowie die Linearisierung der verbleibenden Container.

Linearisierung der Spaltencontainer

Die Spaltencontainer müssen gegenüber dem Screenlayout angepasst werden. Es ist nicht sinnvoll, die einzelnen Spalten wie am Bildschirm nebeneinander liegend auszudrucken. Abhängig von der Menge der Inhalte in den einzelnen Spalten würden unnötig große Freiflächen auf dem Papier entstehen.

Die Spaltencontainer werden daher *linearisiert*, d.h. sie werden in der Reihenfolge ihres Auftretens im Quelltext in voller Breite ausgegeben. Nachfolgend sehen Sie einen Auszug aus dem Print-Stylesheet *print_103_draft.css*. Darin werden die Spaltencontainer *#col1* und *#col3* für die Druckausgabe angepasst, während *#col2* abgeschaltet wird.

```
#col1, #col1_content {float:none; width: 100%; margin: 0; padding: 0;
border: 0}
#col1_content {page-break-after:always}

#col2 {display:none}

#col3, #col3_content {width: 100%; margin:0; padding: 0; border:0}

/* Optionale Spaltenauszeichnung */
/*
    #col1_content:before {content:" [ Linke | Mittlere | Rechte Spalte ]"}
    #col3_content:before {content:" [ Linke | Mittlere | Rechte Spalte ]"}
*/
```

Zusätzlich zur Anpassungen der Spaltenbreiten erhält *#col1* in diesem Beispiel die Eigenschaft *page-break-after:always* womit ein Seitenumbruch erzwungen werden kann. Dieser Seitenumbruch soll der bessern Übersichtlichkeit, insbesondere bei thematischen Unterschieden zwischen den Inhaltsspalten dienen.

Weiterhin finden Sie in den letzten beiden Code-Zeilen eine optionale Benennung der Spaltencontainer. Die darin definierten Texte werden direkt vor den eigentlichen Inhalten ausgegeben. Auf diese Weise haben Sie die Möglichkeit, Zusatzinformationen in die Druckausgabe einfließen zu lassen.

Hinweis: Die optionale Auszeichnung der Spalten ist in allen Print-Stylesheets mit mehr als einer Ausgabespalte bereits vordefiniert, jedoch zur Sicherheit zunächst auskommentiert.

3.9.3 Allgemeine Druckvorbereitung über `print_base.css`

Über die Print-Stylesheets haben Sie die Wahl über die auszudruckenden Inhaltsspalten getroffen. Der Großteil der Layoutanpassungen für den Druck geschieht automatisch über den in den Print-Stylesheets eingebundenen CSS-Baustein `print_base.css` aus dem Verzeichnis `yaml/core/`. Der Inhalt dieses CSS-Bausteins soll im Folgenden näher erläutert werden.

Wichtig: Das Stylesheet `print_base.css` aus dem Verzeichnis `yaml/core/` ist einer der Grundbausteine des YAML-Frameworks. Es stellt allgemeingültige, layoutunabhängige Anpassungen des Basislayouts für die Druckausgabe. Dieses Stylesheet ist in jedem YAML-basierten Layout erforderlich und sollte generell unverändert bleiben!

Allgemeine Anpassungen des Layouts

Die Vorbereitungen beginnen mit dem Abschalten Containern des Basislayouts, die im Ausdruck nicht benötigt werden. Weiterhin wird die Breite des Layouts pauschal auf 100 Prozent gesetzt. Hierbei geht es um eine optimale Ausnutzung des Mediums Papier.

```
/* (en) Preparing base layout for print */
/* (de) Basislayout für Druck aufbereiten */

body, #page_margins, #page, #main {margin:0; padding: 0; border: 0;}
#page_margins, #page {width: 100% !important; min-width: inherit; max-width: none}
#header {height: auto}
#footer {display: none}

/* (en) Hide unneeded container of the screenlayout in print layout */
/* (de) Für den Druck nicht benötigte Container des Layouts abschalten */

#topnav {display: none}
#nav {display:none}
#search {display: none}

/* (en) Linearising subtemplates */
/* (de) Linearisierung der Subtemplates */
.c25l, .c33l, .c38l, .c50l, .c62l, .c66l, .c75l,
.c25r, .c33r, .c38r, .c50r, .c62r, .c66r, .c75r {
  width: 100%; margin:0; float:none; overflow:visible; display:table;
}

.subc, .subcl, .subcr {margin: 0; padding: 0;}
```

Navigationselemente werden generell abgeschaltet. Sie sind ausgedruckt nutzlos. Beachten Sie hier auch den Selektor `#search`. Im Basislayout ist kein vordefinierter Container für den Einbau einer Suchmaske vorgesehen, zu verschieden sind hier die Geschmäcker für dessen Lage im Layout.

Allerdings existiert dieses Element natürlich bei einem Großteil der per CMS betriebenen Webseiten. Daher wurde dieser Selektor an dieser Stelle mit aufgenommen denn auch die Suchfunktion ist natürlich auf dem Papier nicht mehr sinnvoll. Im letzten Schritt folgt schließlich die Linearsierung der Subtemplates.

Umstellung von Zeichensatz und Schriftgrößen

Monitore haben eine deutlich geringere Pixel-Auflösung (72 bis 120 dpi) als das Druckbild (300 bis 1200 dpi). Am Bildschirm sind daher kleine, serifenbehaftete Schriften (z.B. Times) relativ schlecht lesbar. Für die Darstellung am Monitor haben serifenlose Schriftarten (z.B. Verdana oder Arial) deutliche Vorteile.

Auf dem Papier und insbesondere bei längeren Texten ist es genau anders herum - hier werden in der Regel Schriften mit Serifen (z.B. Times) bevorzugt. Aus diesem Grund wird im Drucklayout die Schriftfamilie auf eine serifenbehaftete Schrift umgestellt. Dank der Vererbung in CSS ist das mit wenigen Zeilen CSS-Code erledigt.

```
/* (en) Change font to serif */
/* (de) Zeichensatz auf Serifen umstellen */

body * {font-family: "Times New Roman", Times, serif}
code, pre { font-family:"Courier New", Courier, mono}
body {font-size: 12pt}
```

Auch bei der Angabe von Schriftgrößen gibt es Unterschiede zwischen der Bildschirmdarstellung und sinnvollen Angaben für den Druck. Am Bildschirm ist die Skalierbarkeit der Schriftgröße wichtig, daher werden relative Einheiten wie *EM* oder *Prozent* verwendet. In der Drucktechnik zählen hingegen absolute Größenangaben wie *Punkt* oder *Pica*.

Normaler Text sollte dabei nicht kleiner als **12pt** (12 Punkt) gesetzt werden, um auch auf dem Papier gut lesbar zu erscheinen. Die Umsetzung dieser Vorgabe erfolgt ebenfalls über das `body` Element.

Als nächstes wird über die Eigenschaft `page-break-after:avoid` versucht, den Seitenumbruch unmittelbar nach einer Überschrift zu vermeiden. Auch diese Vorgabe kommt letztlich der Lesbarkeit der Texte auf dem Papier zugute.

```
/* (en) Avoid page breaks right after headings */
/* (de) Vermeidung von Seitenumbrüchen direkt nach einer Überschrift */

h1,h2,h3,h4,h5,h6 { page-break-after:avoid; }
```

Automatische Auszeichnung von URL's, Akronymen und Abkürzungen

Wie bereits zu Anfang erwähnt, ist Papier kein interaktives Medium. Hyperlinks können auf dem Papier nicht angeklickt werden - die URL des Links sollte jedoch erhalten bleiben. Gleiches gilt für erläuternde Texte von Akronymen und Abkürzungen, welche beim Übergang vom Online-Medium zum Papier nicht verloren gehen sollten.

Also muss dafür gesorgt werden, dass externe Link-URLs und erläuternde Hilfstexte mit auf dem Ausdruck erscheinen. Diesmal ist es eine CSS2-Pseudoklasse, mit deren Hilfe auch dieser Stolperstein

aus dem Weg geräumt wird. Die Ausgabe von Hilfstexten erfolgt in runden Klammern, die Ausgabe von Link-URLs in eckigen Klammern, jeweils direkt hinter dem betreffenden Element.

```
/* (en) Disable background graphics of links */
/* (de) Abschalten evlt. vorhandener Hintergrundgrafiken ... */
abbr[title]:after, acronym[title]:after {
    content: '(' attr(title) ')'
}

/* (en) Enable URL output in print layout */
/* (de) Sichtbare Auszeichnung der URLs von Links */
a[href]:after {
    content: " ";
    color: #444;
    background: inherit;
    font-style: italic;
}
```

Wichtig: Die nachfolgenden Anpassungen der Druckausgabe erfordern die Unterstützung von CSS 2.1-Pseudoklassen durch den Browser. Der Internet Explorer erfüllt bis zur aktuellen Version 7.0 diese Anforderungen leider nicht.

Durch diese Deklarationen werden URL's und Erläuterungstexte in Klammern hinter dem jeweils verlinkten oder als Abkürzung gekennzeichnete Text ausgegeben. Auf diese Weise bleibt der Großteil der Informationen der Webseite auch auf dem Papier vorhanden.

Optionale Kennzeichnung der Spalten

Für den Ausdruck mehrspaltiger Webseiten ist eine Linearisierung der Spalten sinnvoll. Durch den Wegfall der links- bzw. rechtsbündigen Ausrichtung erscheinen die Spaltencontainer im Drucklayout jedoch zwangsläufig in der Reihenfolge, in der sie im Quelltext stehen.

Das bedeutet, im Basislayout (Spaltenanordnung 1-3-2) käme die Spalte `#col3`, in der sich im Regelfall die Hauptinhalte befinden, auf Grund ihrer Position im Quelltext an letzter Stelle. Solange ausschließlich diese Spalte ausgedruckt wird, ist dies nicht relevant.

Beim Ausdruck mehrerer Spalten geht für den Anwender infolge der Linearisierung der Spalten die Zuordnung der Inhalte zu deren Lage am Bildschirm - und damit deren Relevanz - unter Umständen verloren. Daher können im Drucklayout die Spalten-Container optional eine Beschriftung erhalten, mit der beispielsweise die Lage am Bildschirm oder deren Inhalt im Ausdruck beschriftet werden kann. Dies lässt sich dank der CSS2-Pseudoklasse `:before` äußerst elegant vorbereiten.

```
/* (en) Preparation for optional column labels */
/* (de) Vorbereitung für optionale Spaltenauszeichnung */

#col1_content:before, #col2_content:before, #col3_content:before {
    content: "";
    color: #888;
    background: inherit;
    display: block;
    font-weight: bold;
    font-size: 1.5em;
}
```

Soll eine Beschriftung ergänzt werden, muss für den jeweiligen Container lediglich noch der Wert für die Eigenschaft `content` nachgereicht werden.

4 Anwendung

4.1 Fünf Regeln ...

Die folgenden Regeln fassen die Grundprinzipien, nach denen YAML entwickelt wurde, zusammen:

Regel 1: YAML ist ein kein Fertiglayout

YAML basiert auf Webstandards und ist ein vielseitiges Werkzeug zur Erstellung flexibler, barrierearmer CSS-Layouts. Grundlage für die effektive Arbeit mit dem Framework ist das Verständnis für den Aufbau und die Arbeitsweise von YAML. Nehmen Sie sich daher unbedingt die Zeit und lesen Sie die Dokumentation bevor Sie mit der Arbeit beginnen.

Regel 2: YAML basiert auf dem TOP-DOWN-Prinzip

YAML stellt flexibles, mehrspaltiges Basislayout mit allen wichtigen Standardelementen sowie funktionale Stylesheets die browserübergreifend korrekte Bildschirmdarstellung und verschiedene Ausgabemedien bereit. Der Nutzer optimiert das fertige Layout durch Löschen nicht benötigter Elemente aus dem Quelltext.

Regel 3: CSS-Grundbausteine

Jedes YAML-basierte Layout benötigt die drei CSS-Grundbausteine *base.css*, *ie hacks.css* und *print_base.css* aus dem Ordner *yaml/core/*. Die ersten beiden Dateien sind verantwortlich für die browserübergreifend korrekte Bildschirmdarstellung, die dritte Datei für eine optimale Ausgabe des Layouts auf Printmedien.

Regel 4: Trennung von YAML- und Nutzer-CSS

Die Dateien im YAML-Ordner sollten unverändert bleiben. Eigene Stylesheets bzw. Anpassungen der von YAML gelieferten CSS-Bausteine gehören in den *css*-Ordner des Nutzers. Damit steht jederzeit eine stabile Basis für die Layoutentwicklung bereit und die Fehlersuche sowie Projektpflege werden erleichtert.

Regel 5: Viel Spaß mit YAML!

4.1.1 Mitgelieferte Beispiele

Zusätzlich zur Dokumentation finden Sie im Ordner *examples* des YAML-Downloadpakets eine große Anzahl an vorgefertigten Beispiellayouts, die Ihnen beim Verständnis zur Arbeit mit dem Framework und als Grundlage für eigene Projekte dienen sollen.

Hinweis: Wenn Sie YAML noch nicht kennen, nehmen Sie sich zunächst die Zeit und lesen Sie die Dokumentation bis zu Ende. Das Kapitel 4 vermittelt den Weg zur praktischen Anwendung, mit dem Sie sich zunächst vertraut machen sollten.

In diesen Beispielen werden die vielfältigen Modifikationsmöglichkeiten des Basislayouts vorgestellt, sowie die Anwendung der verschiedenen CSS-Bausteine des Frameworks demonstriert. Eine Übersicht finden Sie im [Abschnitt 1.5: Der Aufbau des Downloadpakets](#).

4.1.2 Tipps für CSS-Einsteiger

Wenn Sie mit CSS noch nicht vertraut sind, nehmen Sie sich am besten eines der Beispiele, welches Ihren Layoutvorstellungen am weitesten entgegen kommt und spielen Sie mit den einzelnen Stil-Definitionen des Screenlayouts. Probieren Sie, welche Änderungen sich wie auf das Layout auswirken.

Ändern Sie Abstände, Schriftgrößen, Farben und Containerbreiten. So verlieren Sie am schnellsten die Angst vor CSS und lernen gleichzeitig spielend den Umgang mit YAML.

4.2 Empfehlung für die Projektstruktur

Generell gibt es keine zwingenden Vorgaben bei der Arbeit mit YAML. Die hier empfohlene Projektstruktur hat sich jedoch als vorteilhaft erwiesen, da sie die Fehlersuche während der Layouterstellung und die Projektpflege spürbar erleichtert.

4.2.1 Dateien und Verzeichnisse anlegen

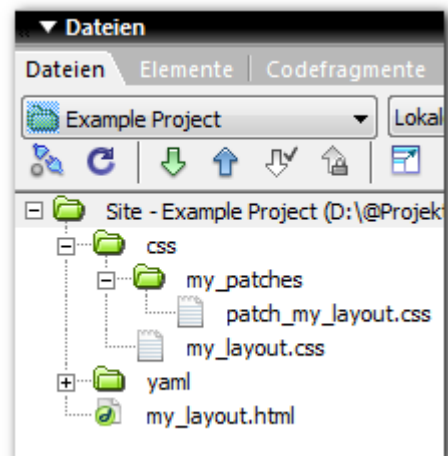
Kopieren Sie zunächst den kompletten Ordner *yaml/* auf Ihren Server und legen Sie auf der gleichen Verzeichnisebene einen *css*-Ordner für die von Ihnen erstellten CSS-Dateien an.

XHTML-Quelltext: Kopieren Sie nun die XHTML-Vorlage *markup_draft.html* aus dem Ordner *yaml/* in Ihr Projektverzeichnis und benennen Sie die Datei um.

Zentrales Stylesheet: Kopieren Sie die Stylesheet-Vorlage *central_draft.css* in Ihren *css*-Ordner und benennen Sie die Datei passend um.

IE-Patches: Kopieren Sie sich die Dateivorlage *patch_layout_draft.css* aus dem Ordner *yaml/patches/* in Ihren Ordner *css/my_patches/* und geben Sie ihr einen zum zentralen Stylesheet passenden Namen (um die Zusammengehörigkeit deutlich zu machen).

Die nebenstehende Grafik verdeutlicht die so entstandene Struktur Ihres neuen Projekts (Auszug aus dem Dreamweaver-Projektfenster).



4.2.2 Anpassung der Dateipfade

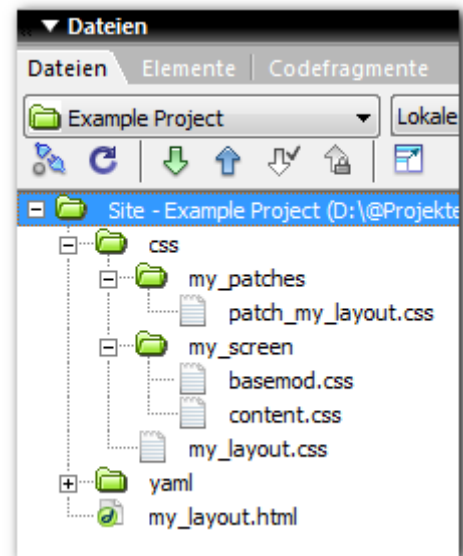
Nach dem Erstellen der Projektstruktur müssen Sie die Dateipfade der CSS-Bausteine kontrollieren. Im XHTML-Quelltext müssen die Pfade zum zentralen Stylesheet und zur Patch-Datei angepasst werden. Im zentralen Stylesheet selbst sowie in der Patch-Datei müssen die Pfade zur *base.css* und *ie hacks.css* kontrolliert werden. Sind diese Arbeiten erledigt, ist das Basislayout einsatzbereit und kann gestaltet werden.

4.2.3 Gestaltung des Layouts

Ab diesem Punkt haben Sie die freie Wahl. Beginnen Sie, eigene Stylesheets für das Screen- und Print-Layout sowie für die Navigation zu erstellen oder greifen Sie auf die Dateivorlagen und vorgefertigten CSS-Bausteine von YAML zurück.

Für das Screenlayout finden Sie im Ordner *yaml/screen/* die Dateivorlagen *basemod_draft.css* für das Seitenlayout und *content_default.css* zur Gestaltung Inhalte.

Kopieren Sie sich diese Vorlagen in Ihren *css*-Ordner und passen Sie sie nach Ihren Wünschen an. Nach dem gleichen Prinzip können Sie mit den Navigationsbausteinen und den Print-Stylesheets verfahren.



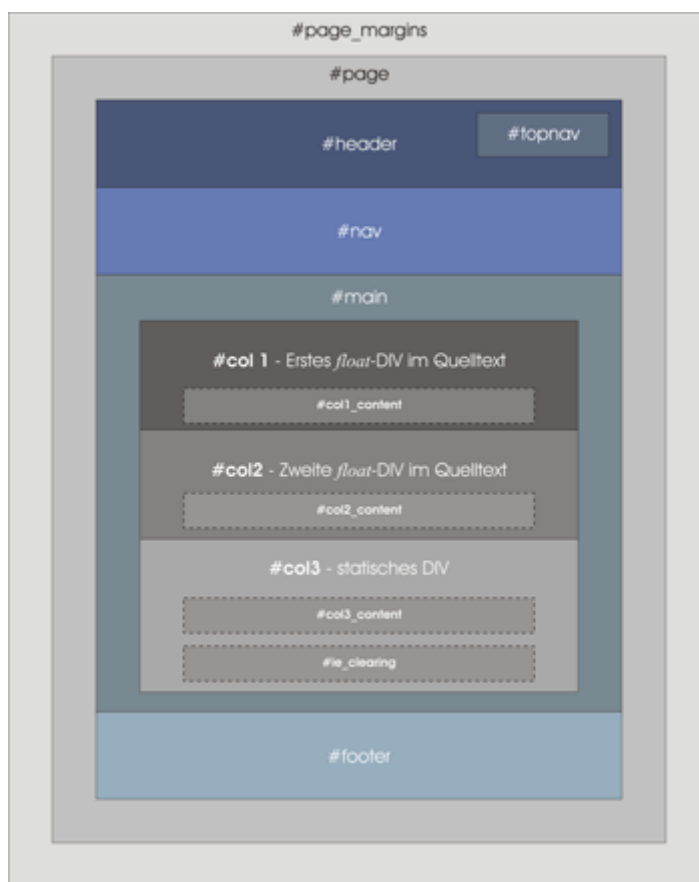
Hinweis: Vergessen Sie nicht, diese zusätzlichen Bausteine in Ihr zentrales Stylesheet aufzunehmen.

4.3 Grundlegende Variationsmöglichkeiten

Sie haben mit YAML zahlreiche Möglichkeiten, das Basis-Layout Ihren eigenen Wünschen anzupassen. Diese Möglichkeiten werde ich in diesem und den folgenden Abschnitten diskutieren. Zunächst werfen wir einen Blick auf die Struktur des (X)HTML-Quelltextes und die Reihenfolge der Spalten darin.

Die Reihenfolge der Spaltencontainer im (X)HTML-Quelltext ist festgelegt und sollte nicht verändert werden. Alle CSS-Bausteine, insbesondere die CSS-Anpassungen für den Internet Explorer, bauen auf dieser Struktur auf.

Mit den Grundvariationen werde ich zunächst Gestaltungsmöglichkeiten ausloten, bei denen die volle Funktionalität des Basislayouts erhalten bleibt. Das betrifft vorrangig das IE-Clearing, was dafür sorgt, dass auch im Internet Explorer die Spalte `#col3` immer zur längsten Spalte wird und damit indirekt auch die grafikfreien Spaltentrennlinien verwendet werden können.

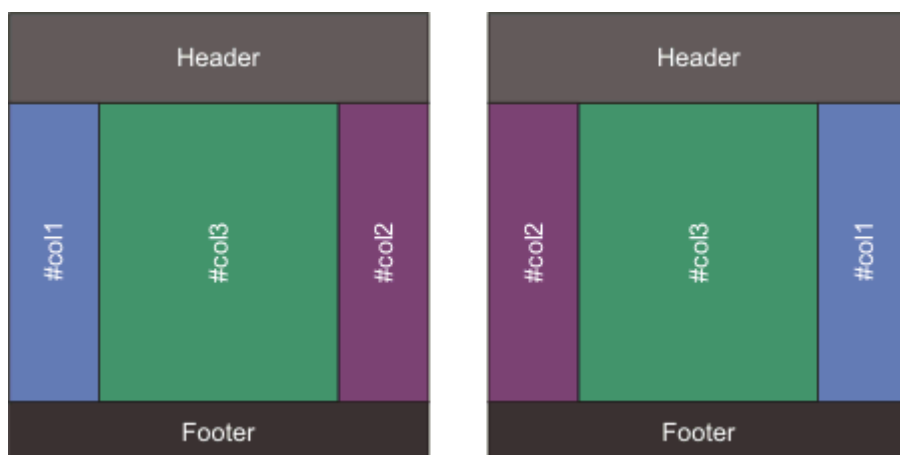


In Bezug auf die Barrierefreiheit eines Layouts steht oft die Forderung im Raum, die eigentlichen Hauptinhalte einer Webseite im Quelltext möglichst weit oben anzuordnen. Das Ziel dieser Regelung ist, den Zugang zu diesen Inhalten über Textbrowser oder Screenreader so einfach wie möglich zu gestalten. Andere Seitenelemente (Sidebar, Werbung, usw.) sollen daher im Quelltext möglichst erst nach den Hauptinhalten folgen.

Hinweis: Für dreispaltige Layouts werde ich im [Abschnitt 4.4](#) ausführlich die Möglichkeiten beschreiben, um diesem Konzept mit YAML uneingeschränkt zu folgen. Dabei geht es um die vollkommen freie Anordnung der einzelnen Spalten auf dem Bildschirm, unabhängig von Ihrer Position im Quelltext.

Der Nachteil der freien Spaltenanordnung ist, dass bei vier der insgesamt sechs möglichen Varianten die grafikfreien Spaltentrennlinien nicht mehr einsetzbar sind, da bei diesen Varianten das IE-Clearing nicht umsetzbar ist.

4.3.1 3-Spalten-Layouts

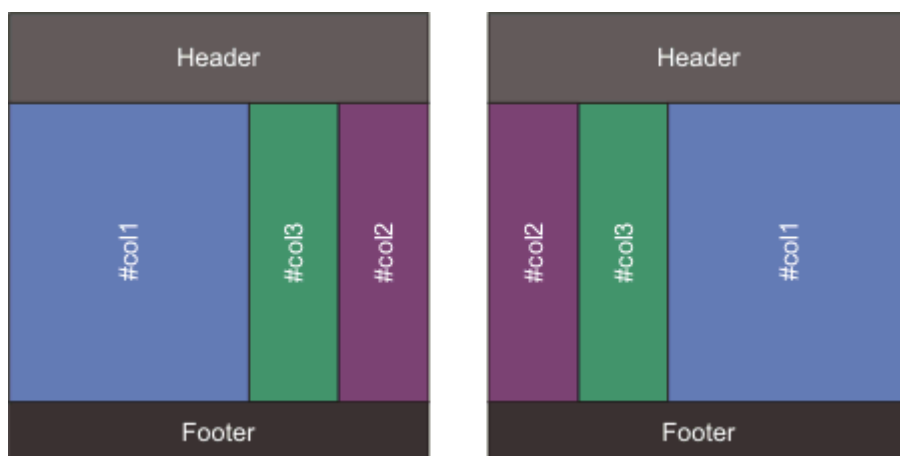


Im Basislayout ist die Spaltenanordnung 1-3-2 umgesetzt. Dabei wird die statische Spalte #col3 von den beiden *float*-Containern #col1 und #col2 umschlossen. Durch einfaches Vertauschen der *float*-Richtung, lässt sich die Spaltenanordnung 2-3-1 erzeugen.

```
#col1 {float:right }
#col2 {float:left }
```

Das Vertauschen der beiden Spalten ermöglicht Ihnen, die Reihenfolge der Inhalte in den Randspalten Ihres Layouts zu beeinflussen. Auf diese Weise können Sie beispielsweise eine Unternavigation, die sich rechts oder links im Layout befindet, im Quelltext direkt unter der Hauptnavigation platzieren. Dazu setzen Sie die Unternavigation in die Spalte #col1 und wählen sich eine der beiden Spaltenanordnungen, um die Unternavigation am linken oder rechten Rand zu positionieren.

In beiden Fällen ist jedoch die Spalte #col3 für die Hauptinhalte vorgesehen und damit an letzter Stelle im Quelltext. Dieser Umstand ist im Sinne der Barrierefreiheit sicherlich ungünstig zu bewerten, er lässt sich jedoch recht einfach durch die standardmäßig im Quelltext eingebauten Skip-Links entschärfen.



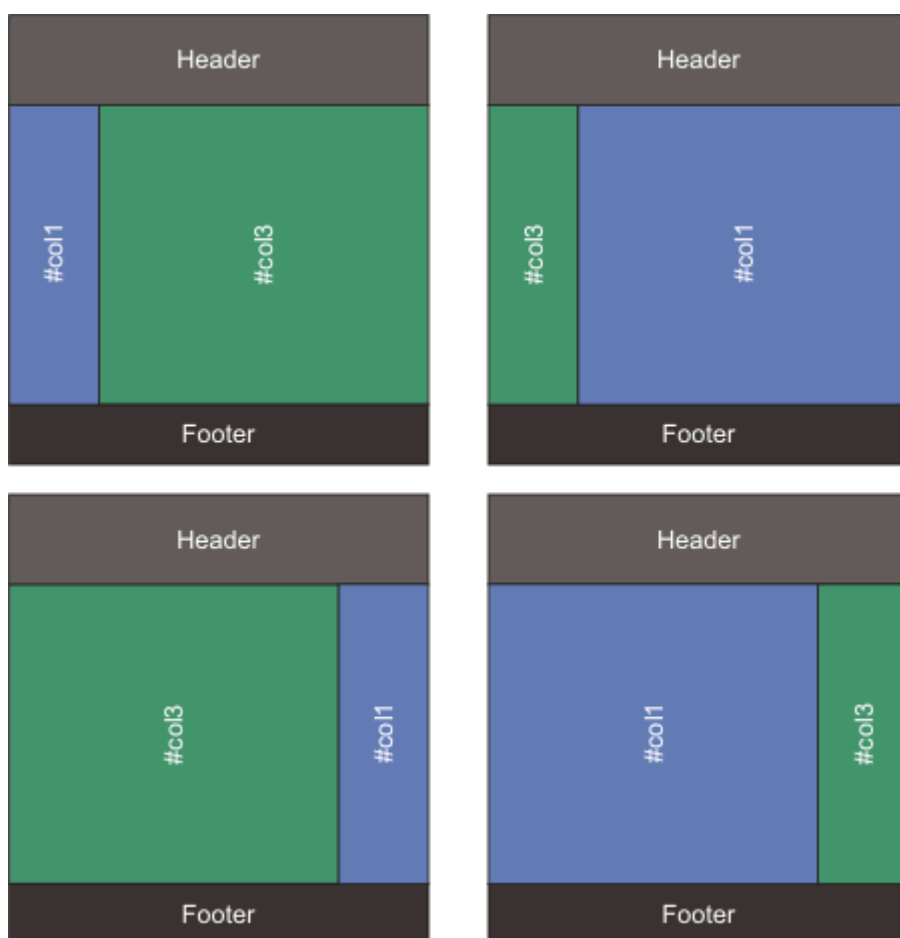
Nun ist dies die gebräuchlichste - jedoch nicht die einzige - Raumaufteilung für ein 3-Spalten-Layout. Eine Alternative bietet sich, indem eine der Randspalten die Hauptinhalte verwendet wird. In diesem

Fall können Navigation, Sidebar und Extras in zwei nebeneinander liegenden, schmalen Spalten platziert werden.

```
#col1 {width: 60%}
#col2 {width: 20%}
#col3 {margin: 0 60% 0 20%}
```

Auch bei dieser Variation bietet es sich an, je nach Lage der Hauptinhalte (links oder rechts) die *float*-Richtung der Spalten `#col1` und `#col2` zu vertauschen. Der Vorteil dieser Lösung liegt darin, dass sich die statische Spalte `#col3` immer noch zwischen den beiden Randspalten befindet und somit der Einsatz der grafikfreien Spaltentrennlinien problemlos möglich ist.

4.3.2 2-Spalten-Layouts



Auch bei Layouts mit zwei Spalten ist es auf einfache Weise möglich, die Platzierung der Hauptinhalte im Quelltext zu optimieren und dennoch volle Gestaltungsfreiheit im Layout zu behalten. Im Regelfall gibt es bei 2-Spaltern eine schmale Spalte für die Navigation und eine breite Spalte für die Inhalte.

Hierbei ist es auf einfache Weise möglich, die Platzierung der Hauptinhalte im Quelltext zu optimieren und dennoch volle Gestaltungsfreiheit im Layout zu behalten. Als Beispiel soll die Navigation am linken Rand erscheinen. Für dieses Beispiel gibt es zwei Wege.

Die nebenstehende Grafik verdeutlicht die möglichen Anordnungen der Spalten. Generell kommt hierbei ausschließlich eine floatender Container (`#col1`) und statischer Container (`#col3`) zum Einsatz.

Bei all diesen Kombinationen lassen sich alle Funktionen des Frameworks uneingeschränkt anwenden (Stichwort: grafikfreie Spaltentrennlinien oder -hintergründe) und gleichzeitig besteht die Möglichkeit, die Inhalte optimal für Suchmaschinen innerhalb des Quelltextes zu platzieren.

Die erforderlichen Eingriffe in das Basislayout sind dabei minimal. Per CSS muss lediglich die Randlage (links/rechts) des Containers `#col1` festgelegt und die zugehörigen Randabstände für `#col3` angepasst werden. Welche Funktion innerhalb des Layouts die beiden Container übernehmen, entscheidet sich dabei lediglich über die Festlegung der Containerbreite.

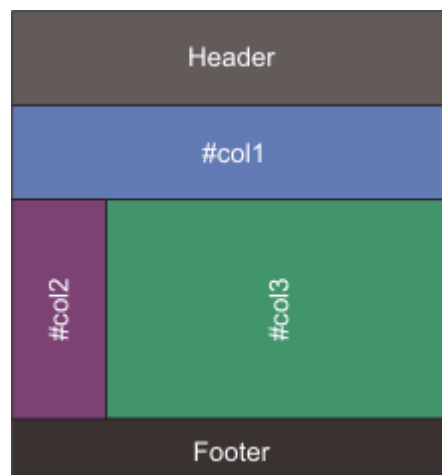
Hinweis: In den Beispielen des Download-Paketes finden Sie vier Varianten für 2-Spalten-Layouts, die jeweils mit den Containern `#col1` und `#col3` umgesetzt wurden. Dabei werden alle möglichen Kombinationen für die Anordnung der Container am Bildschirm vorgestellt.

4.3.3 Weitere Alternativen zur Anordnung der Container

Damit aber noch nicht genug. In den vorangegangenen 2-Spalten-Layouts wurde jeweils eine der beiden *float*-Spalten ausgeblendet. Letztlich kann ein 2-Spalter ebenso gut aus `#col2` und `#col3` erstellt werden. In diesem Fall steht `#col1` für weitere Spielereien zur Verfügung.

Im Standardlayout werden die drei Container zwar als Spalten eines Mehrspaltenlayouts behandelt, jedoch entscheiden letztlich nur Sie über die Verwendung und damit auch die Anordnung der Container.

Im nebenstehenden Beispiel wird zwischen dem Header der Seite und den zweispaltigen Hauptteil noch ein weiterer Container benötigt, der sich über die volle Breite erstreckt. In diesem Fall ist es vorstellbar, den zweispaltigen Hauptteil mit `#col2` und `#col3` zu erstellen und `#col1` darüber zu platzieren.



```
#col1 {float: none; width: auto; }
#col2 {float: left; width: 25%; }
#col3 {margin-left: 25%; margin-right: 0 }
```

Dem Vorgehen bei der Platzierung der Spaltencontainer am Bildschirm sind kaum Grenzen gesetzt. Auf Grund der immer unverändert bleibenden Quelltextstruktur ist es relativ einfach, mögliche störende Einflüsse von CSS-Bugs des Internet Explorers bereits im Vorfeld zu erkennen und abzufangen.

4.4 Freie Anordnung und Verwendung der Content-Spalten

Im [Abschnitt 4.3](#) wurden einige grundlegende Variationsmöglichkeiten des Basislayouts aufgezeigt. Die Bedingung dabei war, dass die volle Funktionalität von YAML (Einsatzmöglichkeit der Border-Definition von `#col3` zur Erzeugung von durchlaufenden Spaltentrennlinien oder Spaltenhintergründen, siehe [Abschnitt 4.6](#)) erhalten bleibt.

Diese Randbedingung ist jedoch ein rein gestalterisches Kriterium und damit nicht zwingend maßgebend bei der Entwicklung eines Layouts. Hinsichtlich der Barrierefreiheit einer Webseite (z.B. der Darstellung der Inhalte in Textbrowsern) sind weitere Kriterien zu berücksichtigen, die ggf. eine Umsortierung der Spaltencontainer gegenüber der Darstellung im Basislayout erfordern.

Oftmals besteht der Wunsch, den Inhalt einer Webseite möglichst weit vorn im Quelltext zu positionieren, während untergeordnete Elemente (Navigation, Sidebar usw.) weiter hinten im Quelltext erscheinen sollen. Zwar ist die Notwendigkeit dieser Maßnahme durchaus diskussionswürdig, aber dies soll nicht hier geschehen. Nachfolgend wird lediglich die grundlegende technische Umsetzung dieser Forderung mit YAML demonstriert.

Hinweis: Die nachfolgenden CSS-Auszüge sind den Beispiellayouts im Verzeichnis *examples/03_layouts_3col/* entnommen. Sie finden die Auszüge in den zugehörigen *basemod_xy.css* Dateien im Ordner *css/screen/*, mit denen die Spaltenreihenfolge des zugrunde liegenden Screenlayouts modifiziert wird.

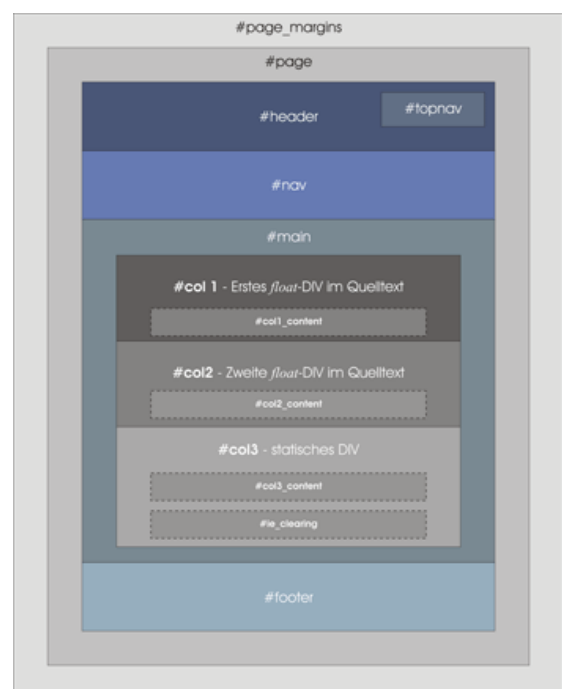
Wichtig: Der Bugfix zur Beseitigung des 3-Pixel-Bugs des Internet Explorers ist in den Patch-Dateien dieser Layoutbeispiele eingebaut, da hier anhand aller möglichen Spaltenanordnungen seine grundsätzliche Anwendung exemplarisch aufgezeigt werden kann.

4.4.1 Spalten in beliebiger Reihenfolge

Der größtmögliche Gestaltungsfreiraum entsteht, wenn die Reihenfolge der Spaltencontainer im Quelltext keinen Einfluss auf die Position der Inhalte am Bildschirm hat. In diesem Fall kann der Webdesigner die Inhalte im Quelltext entsprechend den jeweiligen Anforderungen (Barrierefreiheit, Optimierung für Suchmaschinen usw.) platzieren und hat über die Stylesheets jederzeit volle Kontrolle über die Darstellung auf dem jeweiligen Ausgabemedium.

Wie bereits im [Abschnitt 4.3](#) beschrieben, kann die Reihenfolge der Spalten im Quelltext nicht beliebig vertauscht werden. Aber das ist auch gar nicht notwendig.

Die Lage und Reihenfolge der Spalten am Bildschirm



wird vollständig per CSS geregelt. Sie müssen lediglich Ihre Inhalte an der Stelle des Quelltextes einfügen, an der wir sie haben wollen. Anschließend werden die Container per CSS in Abhängigkeit des Ausgabemediums platziert.

Für drei Spalten mit jeweils unterschiedlichen Inhalten ergeben sich genau sechs mögliche Kombinationen, in der sie am Bildschirm nebeneinander angeordnet werden können. Diese Kombinationen werden im Folgenden kurz vorgestellt und Ihre Einsatzgrenzen umrissen.

Für die drei Spalten des Basis-Layouts ergeben sich insgesamt sechs Kombinationen, in der sie am Bildschirm nebeneinander angeordnet werden können. Bei all diesen Kombinationen wird ein dreispaltiges Layout mit einer Raumaufteilung von 25 | 50 | 25 Prozent erzeugt. Sie finden die Positionierungsbeispiele im Ordner *examples/03_layouts_3col/* des Download-Pakets.

Für jede mögliche Spaltenanordnung sind die wichtigsten Merkmale in einer tabellarischen Übersicht zusammengefasst. Die folgende Legende erläutert die darin verwendeten Abkürzungen:

Kurzbezeichnung	Erläuterung
E-Mix	Die Mischung verschiedener Einheiten fix (Pixel), flexibel (%) und elastisch (EM) zur Festlegung der Spaltenbreiten ist innerhalb eines Layouts möglich.
Prozent	Die Erstellung eines flexiblen Layouts auf Prozent-Basis ist möglich. Alle Spaltenbreiten müssen dabei als Prozentwerte angegeben werden.
Pixel	Die Erstellung eines Layouts mit fixer Breite ist möglich. Alle Spaltenbreiten werden dabei als Pixelwerte festgelegt.
EM	Die Erstellung elastischen Layouts auf Basis der Einheiten EM/EX ist möglich. Alle Spaltenbreiten müssen dabei als EM/EX-Werte angegeben werden.
3P-Fix	Der 3-Pixel-Bug lässt sich vollständig beheben.
SPT	Die Eigenschaft <code>border</code> von <code>#col3</code> kann zur Erstellung von grafikfreien Spaltentrennlinien oder Spaltenhintergründen verwendet werden.
Faux	Die „ <i>Faux Columns</i> “ Technik zur Erstellung von Spaltentrennlinien oder Spaltenhintergründen mittels Hintergrundgrafiken ist einsetzbar.

4.4.2 Spaltenanordnung 1-3-2 und 2-3-1

Layout	E-Mix	Prozent	Pixel	EM	3P-Fix	SPT	Faux
1-3-2	Ja	Ja	Ja	Ja	Ja *)	Ja	Ja
2-3-1	Ja	Ja	Ja	Ja	Ja *)	Ja	Ja

*) Der Einsatz grafikfreier Spaltentrennlinien und die Behebung des 3-Pixel-Bugs über `#col3` schließen sich gegenseitig aus.



Die Spaltenanordnung 1-3-2 entspricht exakt der Standarddefinition, wie sie in der Datei *base.css*, siehe [Abschnitt 3.4](#), verankert ist. Auf beide Varianten bin ich bereits bei den Grundvariationen dreispaltiger Layouts im [Abschnitt 4.3](#) eingegangen.

```
/* #col1 wird zur linken Spalte */
#col1 { width: 25%; }

/* #col2 wird zur rechten Spalte */
#col2 { width: 25%; }

/* #col3 wird zur mittleren Spalte */
#col3 { margin-left: 25%; margin-right: 25%; }
```

Um das Gegenstück 2-3-1 zu erzeugen, müssen wir nicht etwa die Randspalten im Quelltext vertauschen. Viel einfacher ist es, in einer *basemod_xy.css* die *float*-Richtung der beiden Spalten zu ändern.

```
/* #col1 wird zur rechten Spalte */
#col1 { float:right; width: 25%; }

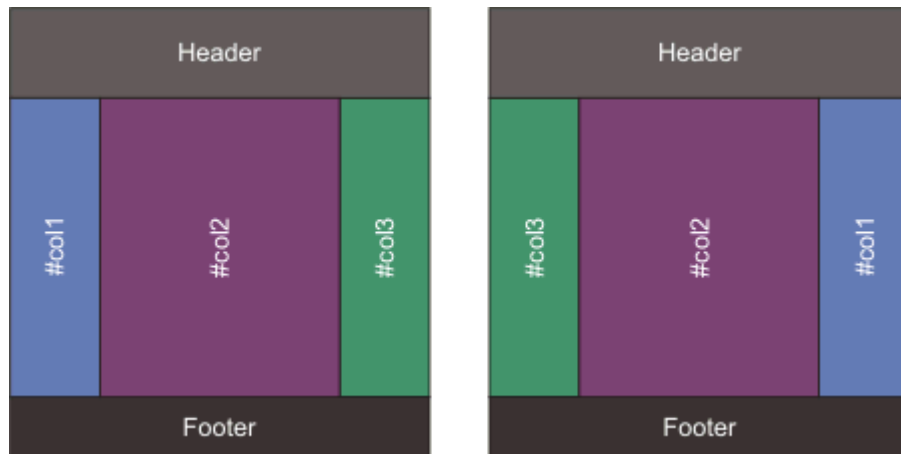
/* #col2 wird zur linken Spalte */
#col2 { float:left; width: 25%; }

/* #col3 wird zur mittleren Spalte */
#col3 { margin-left: 25%; margin-right: 25%; }
```

Das war's schon. Am Bildschirm hätten wir jetzt die Reihenfolge 2-3-1.

4.4.3 Spaltenanordnung 1-2-3 und 3-2-1

Layout	E-Mix	Prozent	Pixel	EM	3P-Fix	SPT	Faux
1-2-3	-	Ja	Ja	Ja	Ja	-	Ja
3-2-1	-	Ja	Ja	Ja	Ja	-	Ja



Die Spalten sollen genau in der Reihenfolge von links nach rechts (1-2-3) oder von rechts nach links (3-2-1) geordnet werden, in der sie auch im Quelltext aufeinander folgen.

Auch diese Darstellungsreihenfolge ist relativ einfach zu bewerkstelligen. Als erstes muss dafür gesorgt werden, dass die beiden *float*-Spalten direkt nebeneinander platziert werden. Dazu lässt man einfach beide Container in die gleiche Richtung floaten. Für die Spaltenanordnung 1-2-3 erhält #col2 daher die Eigenschaft `float:left` bzw. für die Reihenfolge 3-2-1 der Container #col1 die Eigenschaft `float:right`.

Im zweiten Schritt wird dafür gesorgt, dass die Spalte #col3 an den rechten bzw. linken Rand geschoben wird. Dies erledigt man praktischerweise mit einem einseitigen Margin, welcher genau so breit ist, wie die beiden Spalten #col1 und #col2 zusammen. Bei der Spaltenanordnung 1-2-3 erfolgt die Sortierung somit von links nach rechts.

```
/* #col1 wird zur linken Spalte */
#col1 { width: 25%; margin: 0;}

/* #col2 wird zur mittleren Spalte */
#col2 { width: 50%; float:left; margin: 0;}

/* #col3 wird zur rechten Spalte */
#col3 { margin-left: 75%; margin-right: 0%; }
```

Für die Spaltenanordnung 3-2-1 erfolgt die Sortierung in von rechts nach links.

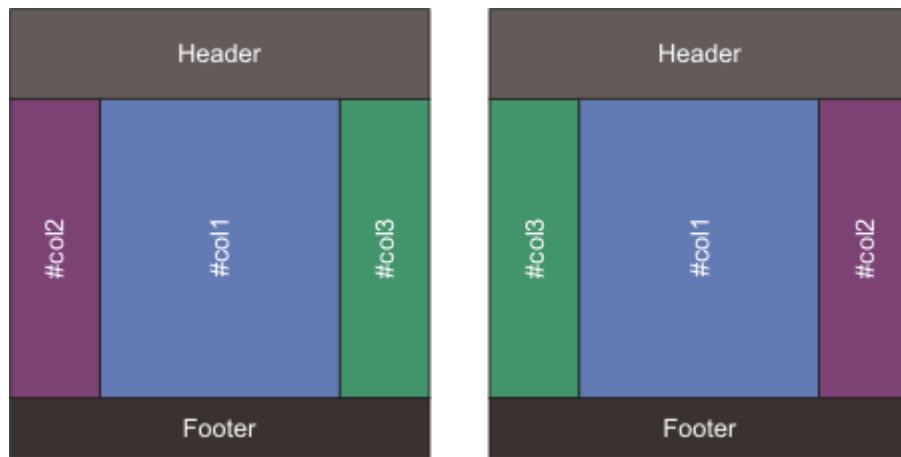
```
/* #col1 wird zur rechten Spalte */
#col1 { width: 25%; float:right; margin: 0;}

/* #col2 wird zur mittleren Spalte */
#col2 { width: 50%; margin: 0;}

/* #col3 wird zur linken Spalten */
#col3 { margin-left: 0; margin-right: 75%; }
```

4.4.4 Spaltenanordnung 2-1-3 und 3-1-2

Layout	E-Mix	Prozent	Pixel	EM	3P-Fix	SPT	Faux
2-1-3	-	Ja	Ja	-	Nicht erforderlich	-	Ja
3-1-2	-	Ja	Ja	-	Nicht erforderlich	-	Ja



Die letzten beiden verbliebenen Kombinationsmöglichkeiten sehen vor, dass die mittlere Spalte am Bildschirm gleichzeitig die erste Spalte im Quelltext ist. Die zuvor beschriebene Spaltenanordnung zeigt, dass bei gleicher float-Richtung die beiden Spalten `#col1` und `#col2` automatisch in der Reihenfolge erscheinen, in der sie im Quelltext stehen. Das müssen wir jetzt irgendwie ändern.

An dieser Stelle sei ein Blick über den Tellerrand gestattet, denn darüber haben sich bereits an anderer Stelle kreative Leute den Kopf zerbrochen und eine wunderbar einfache Lösung entdeckt: Negative Margins. [Alex Robinson](#) verwendet diese Technik auch im Abschnitt *"any order columns"* seines Artikels *"In search of the One True Layout"*. Mittels der negativen Margins können die beiden Spalten so verschoben werden, dass genau der gewünschte optische Effekt entsteht. Das gleiche Prinzip lässt sich auf die beiden *float*-Spalten des YAML-Layouts anwenden.

Im ersten Schritt wird dafür gesorgt, dass die beiden Spalten `#col1` und `#col2` in die gleiche Richtung floaten. Dazu bekommt `#col2` die Eigenschaft `float:left` zugewiesen. Im zweiten Schritt erhält `#col1` einen Außenabstand vom linken Rand (`margin-left`), der exakt der Breite von `#col2` entspricht. Damit befindet sich diese Spalte bereits in ihrer endgültigen Lage. Die Spalte `#col2` floatet in die gleiche Richtung wie `#col1` (dafür haben wir zuvor gesorgt) und wird daher vom Browser normalerweise rechts neben `#col1` platziert. Doch da gehört sie in unserer Spaltenanordnung nicht hin. Sie soll auf die linke Seite von `#col1`.

Hier helfen negativen Margins. Fixpunkt der Spalte `#col2` ist die obere linke Ecke. Um die Spalte nach links neben `#col1` zu verschieben, muss sie um die Breite von `#col1` sowie ihrer eigenen Breite nach links versetzt werden. Als Außenabstand ergibt sich somit ein Wert von -75 Prozent. Im letzten Schritt wird die Spalte `#col3` an den rechten Rand gefloatet.

Da jetzt alle drei Spalten floaten, muss `#main` dazu animiert werden, alle Floats einzuschließen. Dies geschieht im Internet Explorer über die Aktivierung von `hasLayout` durch Vorgabe der Containerbreite zu 100% sowie in allen anderen Browsern durch die Eigenschaft `float:left`.

```

/* floats einschließen in #main */
#main { width: 100%; float:left; }

/* #col1 wird zur mittleren Spalte */
#col1 { width: 50%; float:left; margin-left: 25%; }

/* #col2 wird zur linken Spalte */
#col2 { width: 25%; float:left; margin-left: -75%; }

/* #col3 wird zur rechten Spalte */
#col3 {
  margin-left: -5px;
  margin-right: 0;
  float:right; width: 25%;
}

```

Bei dieser Spaltenanordnung würde der [Doubled Float Margin Bug](#) des Internet Explorers normalerweise gnadenlos zuschlagen, indem er alle vorgegebenen Außenabstände verdoppelt und somit das Layout zerstört. Der entsprechende Bugfix ist jedoch in der Datei *ie hacks.css* bereits integriert und wird auf diese Weise in jedes YAML-basierte Layout eingebunden. Sie brauchen sich also keine Sorgen zu machen.

Das Vorgehen für die Spaltenanordnung 3-1-2 ist ähnlich, nur müssen die *float*-Richtung für `#col1` und `#col2` gewechselt und die Außenabstände in Bezug auf den rechten Rand ermittelt werden. Das Vorgehen unterscheidet sich zudem in Bezug auf die Spalte, welche den negativen Margin zugewiesen bekommt. Im Internet Explorer scheitert in diesem Fall die Festlegung eines nach rechts gerichteten negativen Margins für `#col2`.

Daher erhält zunächst `#col1` die Eigenschaft `float:right` und floatet damit in die gleiche Richtung wie `#col2`. Dann wird `#col1` in die Mitte durch einen negativen Margin verschoben, der wiederum der Summe der Breite von sich selbst und der Breite von `#col2` entspricht (`margin-left: -75%`). Und damit auch die älteren Browserversionen des IE mitspielen, wird zusätzlich auch der Außenabstand zum linken Rand explizit vorgegeben. Durch den Versatz von `#col1` in die Mitte der Seite ist am rechten Rand nun Platz für `#col2`.

Als letzter Schritt erfolgt wiederum die Positionierung von `#col3` an den linken Rand sowie das Einschließen der Spalten in `#main` durch die Aktivierung von *hasLayout* sowie die Vorgabe von `float:left` für `#main`.

```

/* floats einschließen in #main */
#main { width:100%; float:left; }

/* #col1 wird zur mittleren Spalte */
#col1 { width: 50%; float:right; margin-left: -75%; margin-right: 25%; }

/* #col2 wird zur rechten Spalte */
#col2 { width: 25%; float:right; margin-right: 0%; }

/* #col3 wird zur linken Spalte */
#col3 {
  margin-left: 0;
  margin-right: -5px;
  float:left; width: 25%;
}

```

Die Verschiebung der Spalten mittels negativer Margins funktioniert in allen modernen Browsern. Allerdings gibt es nach [Aussage](#) von Alex Robinson im Netscape 6 & 7, sowie mit dem älteren Opera 6 Probleme. Die aktuelle Browserversion Netscape 8.x stellt das Layout jedoch nach eigenen Tests fehlerfrei dar und der Opera 6 darf wohl auch mittlerweile zum *Alten Eisen* gezählt werden.

4.4.5 Fazit

Mit YAML ist die Reihenfolge der Spalteninhalte im Quelltext völlig unabhängig von der Position der Spalten am Bildschirm. Über die Verwendung der drei Spalten `#col1` bis `#col3` für Inhalt, Navigation, Sidebar usw. entscheidet daher einzig und allein der Webdesigner. Die Vor- und Nachteile der einzelnen Lösungen sind bekannt und können damit im Verhältnis zu ihrem Nutzen objektiv bewertet werden.

Hinweis: YAML bietet über seine Print-Stylesheets eine optionale Auszeichnung der Spaltencontainer (Beschriftung) für die Druckausgabe an. Dies kann sinnvoll sein, wenn die linearisierte Darstellung der Spaltencontainer beim Ausdruck die Inhalte mehrerer Spaltencontainer nicht in der logischen Abfolge wiedergeben.

4.5 Subtemplates

Mit dem blanken Layout ist die Webseite natürlich noch nicht fertig. Auch die Inhalte müssen entsprechend formatiert werden. Dabei besteht erstens an vielen Stellen die Notwendigkeit, kurze Inhaltsabschnitte nebeneinander anzuordnen, wobei hier nicht von Tabellendaten die Rede ist. Zweitens erfüllt ein herkömmliches Spaltenlayout nicht in jedem Fall die Anforderungen an die heutige Layoutgestaltung. Ein Beispiel für eine deutlich freiere Gestaltung ist die Startseite von www.yaml.de.

Für diese Zwecke bietet YAML so genannte *Subtemplates*. Das sind XHTML-Codebausteine, welche eine horizontale Gliederung von Inhalten innerhalb beliebiger Container ermöglichen. Diese Bausteine basieren auf ineinander floatende DIV-Boxen.

Hinweis: Alle erforderlichen CSS-Definitionen für die Subtemplates befinden sich in der Datei *base.css*. Die Anpassungen für das korrekte automatische Clearing im Internet Explorer befinden sich in der Datei *ie hacks.css*. Subtemplates sind in die Grundbausteine des Frameworks integriert und stehen somit in allen Layoutvariationen zur Verfügung.

Subtemplates können zudem ineinander geschachtelt werden. Dadurch lässt sich die Raumaufteilung nahezu beliebig variieren.

4.5.1 Struktureller Aufbau

Der Aufbau eines solchen Codebausteins soll exemplarisch an einem einfachen Beispiel erläutert werden. Hier der erforderliche XHTML-Code für eine 50/50 Teilung, also eine Aufteilung in zwei gleich große, nebeneinander liegende Blöcke.

```
<!-- Subtemplate: 2 Spalten mit 50/50 Teilung -->
<div class="subcolumns">
  <div class="c501">
```

```

    <div class="subcl">
      <!-- Inhalt linker Block -->
      ...
    </div>
  </div>

  <div class="c50r">
    <div class="subcr">
      <!-- Inhalt rechter Block -->
      ...
    </div>
  </div>
</div>

```

Soweit der Überblick. Nun der genaue Blick in die Details.

Der Container

Ein *Subtemplate* beginnt immer mit einem DIV-Container der Klasse `.subcolumns`, welcher die einzelnen Container zur Raumaufteilung umschließt.

```

<!-- Subtemplate: 2 Spalten mit 50/50 Teilung -->
<div class="subcolumns">
  ...
</div>

```

In der Datei *base.css* erhält der Container `.subcolumns` folgende CSS-Eigenschaften, die nicht verändert werden sollten.

```

/**
 * @section subtemplates
 * @see    ...
 */

.subcolumns, .subcolumns_oldgecko {
  width: 100%;
  overflow: hidden;
}

.subcolumns_oldgecko { float:left }

```

Die Breite des Containers beträgt per Default 100 Prozent, damit füllt er den horizontal im Layout zur Verfügung stehenden Platz voll aus. Gleichzeitig bewirkt die Definition einer konkreten Breite die Aktivierung von *hasLayout* im Internet Explorer, wodurch dieser die Inhalte automatisch einschließt. Alle anderen Browser benötigen dazu die CSS-Eigenschaft `overflow:hidden` (siehe [Abschnitt 2.3: Markupfreies Clearing](#)).

Hinweis: Netscape-Browser bis einschließlich Version 7.1 bzw. alte Gecko-Browser (bis etwa Juli 2004) haben Probleme bei der Darstellung der Subtemplates aufgrund eines Bugs in Verbindung mit `overflow:hidden`. Ab Version 7.2 werden Subtemplates auch im Netscape korrekt dargestellt.

Wenn diese alten Gecko-basierten Browser ebenfalls unterstützt werden sollen, dann können Sie alternativ die Klasse `.subtemplates_oldgecko` verwenden. Beachten Sie dabei die Hinweise im [Abschnitt 5.3](#) zum *overflow-Bug* des Netscape-Browsers.

Die Raumaufteilung über DIV-Blöcke

Die horizontale Raumaufteilung erfolgt über die DIV-Blöcke mit den CSS-Klassen `c50l` und `c50r`. Das "c" steht dabei für *column* (Spalte), die Zahl "50" für *50 Prozent der verfügbaren Breite* und die Buchstaben "l" und "r" für *links-* bzw. *rechts-*floatende Blöcke.

```
<!-- Subtemplate: 2 Spalten mit 50/50 Teilung -->
<div class="subcolumns">
  <div class="c50l">
    ...
  </div>
  <div class="c50r">
    ...
  </div>
</div>
```

In der Regel bilden zwei Blöcke (ein linker und ein rechter) ein Paar. In der Summe *sollte* die Gesamtbreite aller eingesetzten Blöcke innerhalb eines Subtemplates 100 Prozent betragen. Folgende Teilungsverhältnisse werden von YAML über vordefinierte CSS-Klassen zur Verfügung gestellt:

- 50% / 50% Teilung (Klassen `c50l` und `c50r`)
- 33% / 66% Teilung (Klassen `c33l` und `c66r` sowie `c66l` und `c33r`)
- 25% / 75% Teilung (Klassen `c25l` und `c75r` sowie `c75l` und `c25r`)
- Goldener Schnitt (Klassen `c38l` und `c62r` sowie `c62l` und `c38r`)

Die Klassendefinitionen finden sich in der Datei `base.css`.

```
.c50l, .c25l, .c33l, .c38l, .c66l, .c75l, .c62l { float: left }
.c50r, .c25r, .c33r, .c38r, .c66r, .c75r, .c62r { float: right; margin-left: -5px }

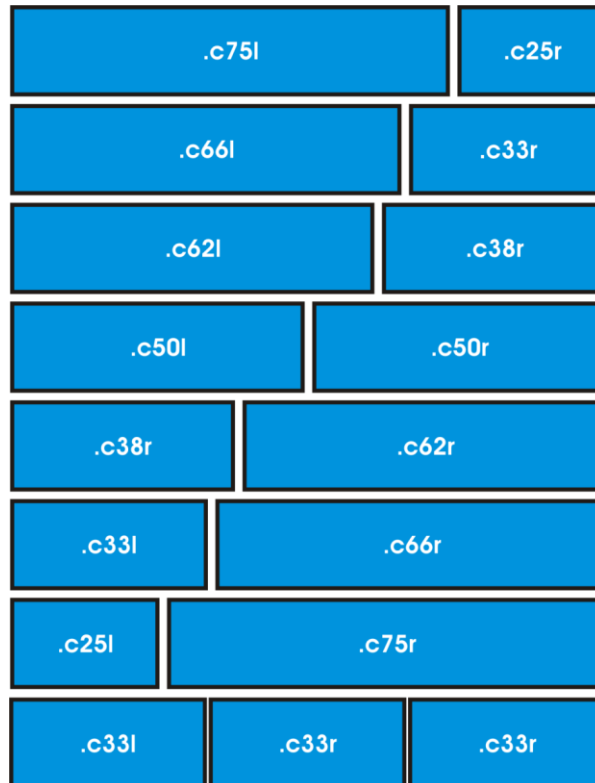
.c25l, .c25r { width: 25% }
.c33l, .c33r { width: 33.333% }
.c50l, .c50r { width: 50% }
.c66l, .c66r { width: 66.666% }
.c75l, .c75r { width: 75% }
.c38l, .c38r { width: 38.2% } /* Goldener Schnitt */
.c62l, .c62r { width: 61.8% } /* Goldener Schnitt */
```

Die reale Breite eines floatenden Containers wird vom Browser direkt vor dem Rendering des Containers ermittelt. Durch das Auf- und Abrunden bei der Umrechnung von Prozentwerten in Pixelmaße kann es (vorwiegend im Internet Explorer) zu Rundungsfehlern in Bezug auf die Gesamtbreite aller DIV-Blöcke innerhalb eines Subtemplates kommen.

Im Ergebnis überschreitet die Summe der Einzelcontainer die Breite des Elterncontainers `.subcolumns`, und es kommt zu Umbrüchen der floatenden DIV-Boxen. Um diesen Effekt zu vermeiden, erhalten alle rechts floatenden DIV-Blöcke zusätzlich die Eigenschaft `margin-left: -5px`. Damit wird erlaubt, dass ein rechts floatender Container ein Element, welches links neben ihm platziert ist, um maximal fünf Pixel überdecken darf. Auf diese Weise können die Rundungsfehler bei der Größenermittlung elegant ausgeglichen werden.

Wichtig: Damit der Ausgleich von Rundungsfehlern fehlerfrei funktioniert, muss sich innerhalb eines Subtemplates immer exakt **ein** rechts-floatender Container befinden.

Mit diesen vordefinierten CSS-Klassen lassen sich - zunächst ohne Verschachtelung der Subtemplates - folgende Raumaufteilungen bilden:



Diese Blöcke können durch Einfügen weiterer Subtemplates beliebig tief ineinander verschachtelt werden. Auf diese Weise lassen sich fast beliebig strukturierte Raumaufteilungen erzeugen.

Die Inhaltscontainer

Wie schon bei den Layoutspalten des YAML-Frameworks übernehmen die äußeren Container (z.B. die DIV-Blöcke `c50l` und `c50r`) die Raumaufteilung, während über die inneren Container `subc`, `subcl` bzw. `subcr` die Abstände und Rahmen der Inhalte (`padding`, `margin`, `border`) geregelt werden.

```
<div class="subcolumns">
  <div class="c50l">
    <div class="subcl">
      <!-- Inhalt linker Block -->
      ...
    </div>
  </div>

  <div class="c50r">
    <div class="subcr">
      <!-- Inhalt rechter Block -->
      ...
    </div>
  </div>
</div>
```

```
.subc { padding: 0 0.5em 0 0.5em; }
.subcl { padding: 0 1em 0 0; }
.subcr { padding: 0 0 0 1em; }
```

Die Endbuchstaben "l" und "r" stehen wiederum für Inhaltsblöcke am linken oder rechten Rand des Subtemplates. Dies hat Einfluss auf die Innenabstände (`padding`). Für Inhaltsblöcke, die sich nicht in Randlage befinden (z.B. der mittlere Block einer 33/33/33 Teilung), steht der Container `subc` zur Verfügung, der ein beidseitiges Padding besitzt.

Damit die Spalten wirklich gleich breit werden, muss bei der Festlegung der Innenabstände (`padding`) für die Inhaltscontainer `subc`, `subcl` und `subcr` darauf geachtet werden, dass die Summe der jeweils vergebenen Paddings identisch ist.

Die Container `subcl` und `subcr` befinden sich planmäßig in Randlage und erhalten daher jeweils ein einseitiges Padding von `1em`. Der Container `subc` benötigt demzufolge ein beidseitiges Padding, welches in der Summe ebenfalls `1em` ergeben, also ein linkes und rechtes Padding von jeweils `0.5em`.

4.5.2 Anpassungen der Subtemplates für den Internet Explorer

Bei den Subtemplates wird intensiver Gebrauch von der CSS-Eigenschaft `float` gemacht. Demzufolge muss natürlich auch mit den entsprechenden IE-Bugs (Escaping Floats Bug, Doubled-Float-Margin Bug) gerechnet werden. Des Weiteren wird speziell bei der Arbeit mit flexiblen Containerbreiten das Expanding Box Problem sehr schnell spürbar zum Problem.

Analog zu den Bugfixes für das YAML-Basislayout, lassen sich die bekannten Bugfixes natürlich auch auf die Subcolumns anwenden.

```
* html .c50l, * html .c25l, * html .c33l, * html .c38l, * html .c66l,
* html .c75l, * html .c62l, * html .c50r, * html .c25r, * html .c33r,
* html .c38r, * html .c66r, * html .c75r, * html .c62r {
  display:inline;
}

* html .subcolumns .subc,
* html .subcolumns .subcl,
* html .subcolumns .subcr {
  word-wrap: break-word;
  overflow:hidden;
}
```

Der Escaping Floats Bug wird bereits durch die Vorgabe von `width:100%` für den Container `.subcolumns` ausgeschaltet (über `hasLayout`). Die Eigenschaft `display:inline` beseitigt den Doubled-Float-Margin Bug und die beiden Eigenschaften `word-wrap:break-word` und `overflow:hidden` sorgen in den älteren IE-Generationen (IE5.x und IE6) dafür, dass übergroße Inhalte zuverlässig abgeschnitten werden und das Layout nicht zerstören.

Hinweis: Für die Druckausgabe wird die Eigenschaft `word-wrap` in der Datei `ie hacks.css` wieder auf ihren Standardwert `word-wrap:normal` zurück gesetzt.

4.5.3 Beispiele für die Anwendung von Subtemplates

Die nachfolgenden Beispiele zeigen exemplarisch die einfache Anwendung sowie die Möglichkeiten zur Verschachtelung von Subtemplates. Schauen Sie sich den Quelltext dieser Beispiele an, um sich mit der Vorgehensweise vertraut zu machen.

50 / 50 Teilung

Block 1: Lorem ipsum dolor sit amet, consectetur adipiscing elit. In ac lectus. Aenean tincidunt metus nec orci. Nulla dapibus mattis tellus. Ut dui nunc, ultrices ut, egestas vitae, feugiat ac, tortor. Nullam velit. Nunc ac urna. Nullam sed quam ac turpis porta porta. Aliquam ut sem ut leo mattis ultricies. Aliquam aliquam ligula ut purus. ...

Block 2: Lorem ipsum dolor sit amet, consectetur adipiscing elit. In ac lectus. Aenean tincidunt metus nec orci. Nulla dapibus mattis tellus. Ut dui nunc, ultrices ut, egestas vitae, feugiat ac, tortor. Nullam velit. Nunc ac urna. Nullam sed quam ac turpis porta porta. Aliquam ut sem ut leo mattis ultricies. Aliquam aliquam ligula ut purus. ...

33 / 33 / 33 Teilung

Block 1: Lorem ipsum dolor sit amet, consectetur adipiscing elit. In ac lectus. Aenean tincidunt metus nec orci. Nulla dapibus mattis tellus. Ut dui nunc, ultrices ut, egestas vitae, feugiat ac, tortor. Nullam velit. Nunc ac urna. ...

Block 2: Lorem ipsum dolor sit amet, consectetur adipiscing elit. In ac lectus. Aenean tincidunt metus nec orci. Nulla dapibus mattis tellus. Ut dui nunc, ultrices ut, egestas vitae, feugiat ac, tortor. Nullam velit. Nunc ac urna. ...

Block 3: Lorem ipsum dolor sit amet, consectetur adipiscing elit. In ac lectus. Aenean tincidunt metus nec orci. Nulla dapibus mattis tellus. Ut dui nunc, ultrices ut, egestas vitae, feugiat ac, tortor. Nullam velit. Nunc ac urna. ...

Teilung nach dem Goldenen Schnitt

Block 1: Lorem ipsum dolor sit amet, consectetur adipiscing elit. In ac lectus. Aenean tincidunt metus nec orci. Nulla dapibus mattis tellus. Ut dui nunc, ultrices ut, egestas vitae, feugiat ac, tortor. Nullam velit. Nunc ac urna. Nullam sed quam ac turpis porta porta. Aliquam ut sem ut leo mattis ultricies. Aliquam aliquam ligula ut purus. ...

Block 2: Lorem ipsum dolor sit amet, consectetur adipiscing elit. In ac lectus. Aenean tincidunt metus nec orci. Nulla dapibus mattis tellus. Ut dui nunc, ultrices ut, egestas vitae, feugiat ac, tortor. Nullam velit. Nunc ac ...

Endlose Vielfalt durch Schachtelung

Subtemplates können beliebig ineinander geschachtelt werden, so dass sich beinahe beliebige Raumaufteilungen ergeben. Einzige Bedingung: Innerhalb einer Schachtelungsebene sollte die Summe der Blöcke immer 100 Prozent ergeben.

Das nachfolgende Beispiel zeigt exemplarisch eine solche Schachtelung. Innerhalb des linken 50-Prozent-Blocks werden in der zweiten Ebene zwei weitere 50-Prozent-Blöcke platziert. Der rechte 50 Prozent breite Block wird in der zweiten Ebene nach dem Goldenen Schnitt geteilt.

Block 1: Lorem ipsum dolor sit amet, consectetur adipiscing elit. In ac

Block 2: Lorem ipsum dolor sit amet, consectetur adipiscing elit. In ac

Block 3: Lorem ipsum dolor sit amet, consectetur

Block 4: Lorem ipsum dolor sit amet, consectetur adipiscing elit. In ac lectus. Aenean tincidunt metus nec

```
lectus.      Aenean lectus.      Aenean adipiscing elit. In orci. Nulla dapibus mattis
tincidunt metus nec tincidunt metus nec ac lectus... tellus. Ut dui nunc,...
orci. Nulla dapibus orci. ...
mattis tellus....
```

4.5.4 Alternatives Layoutkonzept

Dem klassischen Mehrspaltenlayout mit Header und Footer sieht man seine Verwandtschaft - die Tabelle - auf den ersten Blick an. Das Spaltenkonzept ist bewährt und bietet mit der Umsetzung innerhalb des YAML-Frameworks eine hohe Flexibilität.

Doch das Konzept hat seine Grenzen und die globale Raumaufteilung in 2 oder 3 Spalten zusammen mit Kopf- und Fußzeile ist nicht überall praktikabel. In der YAML Version 2.4 wurden daher *Subtemplates* eingeführt. Doch genau genommen steckt dahinter weit mehr. Die *Subtemplates* lassen sich nicht nur innerhalb der Inhaltsspalten einsetzen.

Vielmehr stellen sie neben dem klassischen Mehrspaltenlayout ein extrem vielseitiges Werkzeug für eine viel freiere Layouterstellung dar. Durch die Möglichkeit der Schachtelung der Subtemplates lassen sich beinahe beliebige Raumaufteilungen realisieren.

4.6 Gestaltung der Spalten

Im [Abschnitt 2.7](#) wurde das besondere Clearing am Ende der statischen Spalte `#col3` bereits ausführlich erläutert.

Zwar erreicht man damit noch nicht das große Ziel von echten gleich langen Spalten, jedoch kommt die Lösung des YAML-Frameworks diesem Wunsch sehr nahe. Wie nahe, das sollen die folgenden zwei Beispiele demonstrieren.

Wichtig: Die Spalte `#col3` behält in der Regel den Wert `width:auto`! Andernfalls vergibt der Internet Explorer intern das Merkmal `hasLayout = true` (siehe Artikel: [on having Layout](#)), was dazu führt das das IE-Clearing am Ende von `#col3` wirkungslos wird, denn es würde ebenso eingeschlossen werden.

Hintergrund: Die statische Spalte `#col3` ist der *Lieferant* für die Spaltentrennlinien. Die Spalte würde in diesem Fall nicht mehr nach unten erweitert werden und die an `#col3` gebundenen Spaltentrennlinien reichen nicht mehr in jedem Fall bis zur Fußzeile.

4.6.1 Beispiel 1 - Spaltentrennlinien

Ein oft genutztes Gestaltungselement sind vertikale Trennlinien zwischen den einzelnen Content-Spalten. Damit diese - unabhängig vom Füllstand der Spaltencontainer - immer gleich lang sind, wird hierfür in der Regel die *Faux-Columns-Technik* (Verwendung von Hintergrundgrafiken) eingesetzt.

Bei den Spaltenanordnungen 1-3-2 und 2-3-1 (siehe [Abschnitt 4.4](#)) kann bei YAML auf den Einsatz von Grafikelementen für dieses Gestaltungselement vollständig verzichtet werden. Stattdessen wird die CSS-Eigenschaft `border` der statischen Spalte `#col3` verwendet. Dies ist möglich, da `#col3` bei diesen Spaltenanordnungen immer die längste ist.

Hier ein Beispiel für eine zwei Pixel breite Punktlinie als Trennlinien zwischen den Spalten eines 3-Spalten-Layouts.

```
#col3 {
  border-left: 2px #ddd dotted;
  border-right: 2px #ddd dotted;
}
```

[04 layouts styling/3col_column_dividers.html](#)



4.6.2 Beispiel 2 - Spaltenhintergründe

Eine weitere Möglichkeit zum effektiven Einsatz der CSS-Eigenschaft `border` der statischen Spalte `#col3` ist die Bereitstellung einfarbiger Spaltenhintergründe - vollkommen grafikfrei.

Dabei ersetzt die Eigenschaft `border` die Eigenschaft `margin`, welche im Normalfall dafür sorgt, dass `#col3` korrekt zwischen den *float*-Spalten platziert wird. Hier ein Beispiel, bei dem beide *float*-Spalten eine fixe Breite von 200 Pixeln besitzen.



```
#col1, #col2 { width: 200px; }

#col3 {
  margin: 0; padding:0;
  border-left: 200px #cce solid;
  border-right: 200px #ecc solid;
}
```

[04 layouts styling/3col column backgrounds.html](#)

Das Ergebnis kommt dem Eindruck gleichlanger Spalten sehr nahe. Allerdings ist diese Technik auf einfarbige Hintergründe begrenzt. Weiterhin gilt zu beachten, dass sie nur in Verbindung mit Maßangaben in **Pixel** oder **EM** sinnvoll einsetzbar ist, denn für `border` sind keine Prozentangaben erlaubt.

4.7 Minimale & Maximale Breiten für flexible Layouts

Flexible Layouts passen sich dynamisch an die aktuelle Breite des Browserfensters an. Diese im Normalfall sinnvolle Eigenschaft ist allerdings nicht in jeder Situation wünschenswert. Zum Beispiel macht es keinen Sinn, wenn das Layout aufgrund eines extrem schmalen Browserfensters zerfällt bzw. unleserlich und damit unbenutzbar wird. Hier sollten Sie einen unteren Grenzwert definieren, der sich beispielsweise an einer Desktop-Auflösung von 800x600 Pixeln orientieren könnte und der noch eine fehlerfreie Layoutdarstellung ermöglicht.

Ebenso wichtig ist die Festlegung einer Obergrenze für die Breite des Layouts. Bei einem "zu breiten" Layout entstehen in Fließtexten sehr lange Zeilen. Im Extremfall werden mehrzeiligen Absätzen einzelne Textzeilen. Für das Auge entstehen dadurch beim Lesen weite Wege, was die Konzentration beeinträchtigt und den Inhalt schwer lesbar macht. Auch solche Details stellen letztlich Barrieren dar, die den Zugang zu den Informationen unnötig erschweren. Beide Fälle lassen sich in modernen Browsern mit Hilfe der CSS-Eigenschaften `min-width` und `max-width` sehr einfach abdecken.

Im Basislayout des YAML-Framework sollten alle Breitenangaben für das Layout generell an den Container `#page_margins` vergeben werden. Da dieser alle anderen Elemente umschließt, sind zu automatisch für alle Seitenelemente verbindlich.

```
#page_margins {
  min-width: 760px;
  max-width: 100em;
  ...
}
```

Für die minimale Layoutbreite wird in diesem Beispiel ein Wert von *760 Pixeln* definiert. Dieser orientiert sich an einer Desktopauflösung von 800x600 Pixeln und ermöglicht die Darstellung des Layouts im Vollbildmodus des Browsers, ohne dass horizontale Scrollbalken erforderlich werden.

Für die Festlegung einer maximalen Breite empfiehlt es sich hingegen, den Grenzwert auf die Schriftgröße (Einheit EM) zu beziehen. Ein Wert in der Einheit Pixel würde bei der Vergrößerung der Schriften Probleme bereiten, da sich das Layout dem Textzoom in der Breite nicht anpasst. Ungewollte Zeilenumbrüche von Texten und Bildern im Layout wären die Folge. Durch den Bezug zur Schriftgröße kann dieses Problem vermieden werden. Im Beispiel wurde daher ein Wert von 100em definiert.

4.7.1 Fehlende CSS-Unterstützung im Internet Explorer 5.x und 6.0

Wieder einmal ist es der Internet Explorer, der dem Seitenersteller das Leben schwer macht, denn der IE unterstützt bis zur Version 6.0 weder `min-width` noch `max-width`. Erst mit dem Internet Explorer 7.0 liefert Microsoft die Unterstützung der CSS-Eigenschaften `min-width`, `max-width` sowie `min-height` und `max-height` endlich nach. Neben den zahlreichen beseitigten CSS-Bugs und der Erhöhung der Sicherheit beim Surfen ist diese Neuerung für Webdesigner ein Segen. Man kann daher nur wünschen, dass sich der IE7 schnell verbreitet.



Dennoch sollten die älteren IE-Versionen bei der Layouterstellung nicht vernachlässigt werden, denn der Internet Explorer 6 beherrscht nach wie vor alle Browserstatistiken und wird sicherlich nicht so schnell von der Bühne verschwinden, auch wenn sein Marktanteil durch den IE7 schrumpfen wird.

Ich stelle daher zwei Methoden vor, die per Javascript dem Internet Explorer 5.x und 6.0 die Funktionalität der Eigenschaften `min-width` noch `max-width` für diese Browser nachreicht.

4.7.2 Lösung 1: IE Expressions

Der Internet Explorer ermöglicht dem Webseitenersteller über die proprietäre Eigenschaft `expression()`, einen dynamischen Zugriff auf CSS-Eigenschaften per Javascript. Mit Hilfe von Javascript lässt sich das Verhalten der fehlenden CSS-Eigenschaften recht einfach nachbilden. Einen groben Einblick zur Anwendung bietet Svend Toftes Artikel [max-width in IE](#). Die Beispiele in diesem Artikel verlangen jedoch den *Quirks Mode* ([DocTypes & Darstellungsmodi siehe Abschnitt 2.4](#)). Zu diesem Problem hat sich Jeena Paradies Gedanken gemacht und eine [Code-Variante veröffentlicht](#), die auch im *Standard Mode* des IE funktioniert und die Grundlage für die hier besprochene Lösung bildet.

Wichtig: Gegenüber dem Code früherer YAML-Versionen muss der Internet Explorer bei Anwendung dieser Methode nicht mehr in den *Quirks Mode* versetzt werden.

Der Einbau der JS-Expressions sollte praktischerweise in den Patch-Files erfolgen, sodass wiederum nur der IE den erforderlichen Javascript-Code erhält. Hier ein Auszug, wie er in den Layoutbeispielen des Download-Paketes verwendet wird:

```
/*-----*/
/* min-width / max-width for IE
** IE5.x/Win - x
** IE6      - x
** IE7      - 0
*/

* html #page_margins {
  width: 80em;

  width: expression((document.documentElement &&
document.documentElement.clientHeight) ?
  (document.documentElement.clientWidth < 740) ? "740px" : ((
document.documentElement.clientWidth > (80 *
parseInt(document.documentElement.currentStyle.fontSize))) ? "80em" :
"auto") :

  (document.body.clientWidth < 740) ? "740px" : ((
document.body.clientWidth > (80 *
parseInt(document.body.currentStyle.fontSize))) ? "80em" : "auto")
  );
}
```

Hinweis: Die zweifache Abfrage der aktuellen Viewportgröße ist erforderlich, da im IE6.0 im *Standard Mode* über eine andere Methode auf `.clientWidth` zugegriffen werden muss als im IE 5.x, der sich generell im *Quirks Mode* befindet.

In diesem Beispiel wird eine minimale Layoutbreite von 740 Pixeln umgesetzt. Die maximale Breite wird auf Basis der Schriftgröße festgelegt. Hierzu wird die aktuelle Schriftgröße des `body`-Elements ausgelesen und anschließend mit dem Wert 80em verglichen.

Als Fallback-Lösung bei ausgeschaltetem Javascript wird vor der eigentlichen `expression()`-Anweisung über `width: 80em` eine definierte Breite vorgegeben.

4.7.3 Lösung 2: Externes Javascript »minmax.js«

Dem YAML-Downloadpaket liegt im Verzeichnis `tools/javascript` die Javascript-Datei `minmax.js` von doxdesk.com bei. Diese Datei kann im Header der Webseite eingebunden werden und stellt die volle Funktionalität der Eigenschaften `min-width`, `max-width` sowie `min-height` und `max-height` bereit, indem es die CSS-Anweisungen selbstständig auswertet und die Rendering-Ausgaben des Internet Explorers entsprechend anpasst.

Der Einbau des Scripts sollte praktischerweise innerhalb des *Conditional Comments* erfolgen, um es ausschließlich den Versionen 5.x und 6.0 des Internet Explorers zugänglich zu machen (`<!--[if lte IE 6]>`). Im Internet Explorer 7 wird dieses Script nicht mehr benötigt, da dieser Browser die CSS-Eigenschaften vollständig interpretiert.

```

<head>
...
<!--[if lte IE 7]>
<link href="../../css/patches/patch_3col_standard.css" rel="stylesheet"
type="text/css" />
<![endif]-->

<!--[if lte IE 6]>
<script type="text/javascript" src="minmax.js"></script>
<![endif]-->
</head>

```

Das war's bereits - mehr Arbeit ist nicht nötig. Die Wirkungsweise des Javascripts können Sie sich am besten auf der Testseite *minmax_js.html* ansehen.

[06 layouts minmax for ie/minmax_js.html](#)

Hinweis: Dieses Javascript hat jedoch einen entscheidenden Nachteil. Das Javascript wird erst abgearbeitet, nachdem die Seite vollständig gerendert wurde. Das bedeutet, dass bei einem zu großen oder zu kleinen Browserfenster die Seite zunächst ohne die Wirkung von *min-width* oder *max-width* gerendert wird und das Layout erst nach einigen zehntel Sekunden auf die vorgegebene Mindest- oder Maximalbreite springt. Dieses Springen ist für den Nutzer sichtbar und kann beim Surfen innerhalb einer Website durchaus störend wirken. Sie sollten die Arbeitsweise daher vor dem Einbau testen.

4.8 Entwurf & Fehlersuche

Für die Layouterstellung sowie für das Bugfixing bestehender Layout stellt YAML über das Stylesheet *debug.css* aus dem gleichnamigen Ordner *yaml/debug/* bereit. Es kann optional innerhalb des zentralen Stylesheets an beliebiger Stelle (jedoch nach der *base.css*) eingebunden werden und stellt die nachfolgend beschriebenen Funktionen zur Verfügung.

4.8.1 Kontroll-Automatik für *ie hacks.css*

Die Datei *ie hacks.css* ist einer der Grundbausteine des YAML-Frameworks. Dieses Stylesheet muss in jedes YAML-basierte Layout eingebunden werden (siehe [Abschnitt 3.5: CSS-Anpassungen für den Internet Explorer](#)).

Jedoch wird diese Datei nicht über das zentrale Stylesheet eingebunden sondern innerhalb eines IE-Patchfiles importiert. Das Patchfile selbst wird über den *Conditional Comment* ins Layout integriert, um es ausschließlich dem Internet Explorer zugänglich zu machen. Eine der häufigsten Ursachen für Darstellungsprobleme YAML-basierter Layouts ist daher das Fehlen der *ie hacks.css* aufgrund fehlerhafter Pfadangaben oder Schreibfehlern im *Conditional Comment*.

Innerhalb des *debug.css* wird nun der Container *#ie_clearing* dazu verwendet, eine Warnmeldung auszugeben, wenn die Datei *ie hacks.css* nicht ordnungsgemäß eingebunden ist.

```

/* CSS-Warning, if core stylesheet 'iehacks.css' is missing in the layout
*/
*:first-child+html #ie_clearing { display:block }
* html #ie_clearing { display:block }

#ie_clearing {
  width: 500px;
  font-size: 25px;
  position:absolute;
  top: -2px;
  left:0px;
  background: url("images/warning_iehacks.gif") top left no-repeat;
}

```

Die erste Zeile dieses CSS-Blocks wird NUR vom IE7 verstanden und aktiviert die Darstellung des Containers. Das gleiche erledigt die zweite Zeile, speziell für IE5.x und IE6. Anschließend folgen eine CSS-Vorgaben, um im Falle des Fehlens der *iehacks.css* folgende Meldung im Viewport auszugeben.



Wird die Datei *iehacks.css* hingegen korrekt geladen, so werden die CSS-Deklarationen durch die korrekten Werte überschrieben und die Grafik bleibt für den Nutzer unsichtbar.

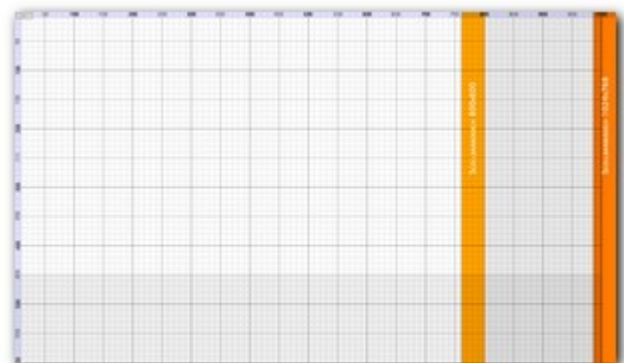
Hinweis: Die automatische Detektion des Fehlens der *iehacks.css* funktioniert nur, wenn der Container `#ie_clearing` im Layout enthalten ist.

Für den Fall, dass Sie Ihr Layout diesen Container nicht enthalten sollte, können Sie direkt in der *iehacks.css* auf einen per Default auskommentierten Block zur Änderung der Hintergrundfarbe des `body` Elements zurückgreifen, um die korrekte Einbindung der Datei zu überprüfen.

4.8.2 Pixelraster für Positions- und Geometriekontrolle

Das Pixelraster beinhaltet ein Grundraster von 10x10 Pixeln. 50- und 100-Pixel-Grenzlinien sind besonders hervorgehoben. Am linken und oberen Rand befindet sich die Legende, welche die die Pixelmaße der Grenzlinien enthält.

Das Pixelraster enthält zwei orangefarbene vertikale Streifen. Diese Streifen markieren den Bereich des Bildschirms, in dem Browser bei einer Bildschirmauflösung von 800x600 Pixeln



bzw. 1024x768 Pixeln gegebenenfalls einen vertikalen Scrollbalken anordnen. Weiterhin sind die Bereiche oberhalb der Breite von 800 Pixeln sowie ab einer Höhe von 450 Pixeln grau hinterlegt. Der darin eingeschlossene weiße Bereich des Rasters entspricht etwa dem Platz, der im Browserfenster bei einer Auflösung von 800x600 Pixeln sichtbar ist (Viewport).

Es kann über die CSS-Klasse `.bg_grid` entweder direkt im Quelltext oder nachträglich über Developer-Toolbars im Browser (z.B. Firebug) direkt an beliebige Elemente als Hintergrundgrafik vergeben werden.

```
/**
 * @section pixel grid
 */

.bg_grid {
  background-image:url(images/grid_pattern.png) !important;
  background-repeat:no-repeat;
  background-position:top left !important;
}
```

Über den Zusatz `!important` wird gewährleistet, dass das Raster nicht durch spätere CSS-Deklarationen wieder abgeschaltet wird.

4.8.3 Hervorhebungen von Elementen

Den letzten Block der *debug.css* bilden CSS-Klassen, um Elemente gezielt einfärben zu können oder sie teilweise transparent erscheinen zu lassen.

```
/**
 * @section transparency
 */

.transOFF { -moz-opacity: 1.0; opacity: 1.0; filter: alpha(Opacity=100);}
.trans50,
.transON { -moz-opacity: 0.5; opacity: 0.5; filter: alpha(Opacity=50);}
.trans25 { -moz-opacity: 0.25; opacity: 0.25; filter: alpha(Opacity=25);}
.trans75 { -moz-opacity: 0.75; opacity: 0.75; filter: alpha(Opacity=75);}

/**
 * @section colors
 */

.bg_red { background-color: #f00 !important;}
.bg_blue { background-color: #00f !important;}
.bg_green { background-color: #0f0 !important;}
```

Hinweis: Dank der Möglichkeit von CSS, Elementen mehr als nur eine Klasse zuzuweisen, können Sie die CSS-Klassen zur Hervorhebung auch beliebig miteinander sowie mit dem Pixelraster überlagern, z.B. `class="bg_grid trans75 bg_red"`.

4.9 Ausgewählte Anwendungsbeispiele

In den folgenden drei Abschnitten werden Beipiellayouts anhand spezieller Anforderungsprofile mit YAML erstellt. Der Aufbau der Beispiele soll helfen, die Handhabung des Frameworks besser zu verstehen und die verschiedenen möglichen Wege zur Gestaltung des Basislayouts aufzuzeigen. Alle mit dem Download-Paket mitgelieferten Beispiele im Ordner *examples/* basieren auf einem einfachen Screenlayout, dessen Erstellung nachfolgend kurz erläutert werden soll.

4.9.1 Das Screenlayout der Layoutbeispiele

Die Grundlage bildet ein flexibles dreispaltiges Layout mit der Spaltenreihenfolge 1-3-2 (Standardaufbau des Basislayouts) und einer Raumaufteilung 25% | 50% | 25% der einzelnen Spalten. Sie finden das Layout im Ordner *examples/01_layouts_basics/*.

01 layouts basics/layout 3col standard.html

Als minimale Breite wird ein Wert von 740 Pixeln festgelegt, der sich an der Desktopauflösung von 800x600 Pixeln orientiert und eine Darstellung der Seite ohne horizontale Scrollbalken ermöglicht. Die maximale Breite des Layouts wird auf 80em begrenzt, was in Verbindung mit der über Datei *content.css* festgelegte Standardschriftgröße von 12 Pixeln einer Breite von 960 Pixeln entspricht.

Das Screenlayout wird über die CSS-Datei *basemod.css* eingebunden, welche sich innerhalb der Themenordner jeweils im Verzeichnis *css/screen/* befindet. Hier der Codeauszug:

```
/* (de) Randbereiche & Seitenhintergrund */
body { background: #9999a0; padding: 10px 0; }

/* (de) Layout: Breite, Hintergrund, Rahmen */
#page_margins {
  min-width: 740px; max-width: 80em;
  margin: 0 auto; border: 1px #889 solid; }
#page{ background: #fff; border: 1px #667 solid; }

/* (de) Zentrierung des Layouts in alten IE-versionen */
body { text-align: center }
#page_margins { text-align:left }

/* (de) Gestaltung der Hauptelemente des Layouts */
#header {
  color: #fff;
  background: #000 url("../../images/bg_gradient.gif") repeat-x bottom
left;
  padding: 45px 2em 1em 20px;
}

#tovnav { color: #aaa; background: transparent; }

#main { background: #fff }

#footer {
  color:#fff;
  background: #336 url("../../images/bg_gradient.gif") repeat-x bottom
left;
  padding: 15px;
}

/* (de) Anpassung der Hauptnavigation */
#nav ul { margin-left: 20px; }
#nav_main {background-color: #336}

/**
 * (de) Formatierung der Inhalts-Container
 *
 * |-----|
 * | #header |
 * |-----|
 * | #col1    | #col3    | #col2    |
 * | 25%      | flexible  | 25%      |
 * |-----|
 * | #footer  |
 * |-----|
 */
```

```
#col1 { width: 25% }
#col1_content { padding: 10px 10px 10px 20px; }

#col2 { width: 25% }
#col2_content { padding: 10px 20px 10px 10px; }

#col3 { margin: 0 25% }
#col3_content { padding: 10px }
```

Hinweis: Der strukturelle Aufbau der Datei basiert auf der Vorlage *basemod_draft.css* aus dem Ordner *yaml/screen/*, die bereits im [Abschnitt 3.6: Erstellung des Screenlayouts](#) erläutert wurde.

Weiterhin wird als horizontale Navigation die Datei *nav_shinybuttons.css* aus dem Ordner *yaml/navigation/* eingebunden und unverändert übernommen. Die einzige Anpassung besteht in der linksbündigen Ausrichtung des ersten Menüpunktes mit einem Wert von 20 Pixeln vom linken Rand des Layouts (*#nav ul { margin-left: 20px }*).

Anpassungen des Screenlayouts für den Internet Explorer

Das Basislayout benötigt im zur fehlerfreien Darstellung im Internet Explorer 5.x und 6.0 noch zwei spezielle Anpassungen. Erstens muss der 3-Pixel-Bug behoben werden und zweitens sollen auch in den älteren IE-Versionen minimale und maximale Layoutbreiten ermöglicht werden. Die Funktionsweise der dafür verwendeten IE-Expressions wird in [Abschnitt 4.6](#) beschrieben.

Die Anpassungen für den Internet Explorer sind in der zum Layout gehörigen Anpassungsdatei *patch_3col_standard.css* im Ordner *css/patches/* abgelegt.

```
/* Bug: 3-Pixel-Jog des Internet Explorers */

* html #col3 { height: 1%; }
* html #col1 {margin-right: -3px;}
* html #col2 {margin-left: -3px;}
* html #col3 { margin-left: 24%; margin-right: 24%; }

/* min-width / max-width for IE

* html #page_margins {
  width: 80em;
  width: expression( ... );
}
```

Damit ist die Grundversion des Screenlayouts einsatzbereit.

4.9.2 Layoutentwurf "2col_left_seo"

An diesen ersten Layoutentwurf mit der Bezeichnung **2col_left_seo** werden folgende Anforderungen gestellt:

- Suchmaschinen-optimiertes Zweispalten-Layout (Navigation links #col3 und Hauptinhalt rechts #col1)
- Flexibles Layout mit flexiblen Spaltenbreiten (25% | 75%)
- Weitere Unterteilung der Hauptinhalts in zwei Spalten nach dem ersten Absatz
- Vertikale 1 Pixel breite Trennlinie zwischen den Spalten mit einem vertikalen Abstand von 1em zu Header und Footer.
- Horizontale Hauptnavigation "Shiny Buttons"
- Drucklayout: Ausgabe der Hauptinhalte aus #col1.



examples/05_layouts_advanced/2col_left_seo.html

Layoutentwurf im Detail

Das zentrale Stylesheet *layout_2col_left_seo.css* beinhaltet folgende CSS-Bausteine:

```
/* import core styles | Basis-Stylesheets einbinden */
@import url(../../yam/core/base.css);

/* import screen layout | Screen-Layout einbinden */
@import url(../../yam/navigation/nav_shinybuttons.css);
@import url(screen/basemod.css);
@import url(screen/basemod_2col_left_seo.css);
@import url(screen/content.css);

/* import print layout | Druck-Layout einbinden */
@import url(../../print/print_001.css);
```

Als erstes folgt die Einbindung des Basisstylesheets *base.css* aus dem Ordner *yaml/core/* sowie die unverändert verwendete Navigation *nav_shinybuttons.css*.

Anschließend wird Grundversion des Screenlayouts *basemod.css* importiert, welche die Grundlage des Layouts bildet. Die Modifikationen, entsprechend des Anforderungsprofils für das gewünschte zweispaltige Layout befinden in der Datei *basemod_2col_left_seo.css*.

```
/* #col1 wird zur Hauptinhaltsspalte */
#col1 { width: 75%; float:right }
#col1_content { padding: 10px 20px 10px 10px; }

/* #col2 abschalten */
#col2 { display:none; }

/* #col1 wird zur linken Spalten */
#col3 { margin-left: 0; margin-right: 75%; }
```

```
#col3_content { padding: 10px 10px 10px 20px; }

/* Grafikfreier Spaltentrenner zw. #col1 und #col3*/
#col3 {border-right: 1px #ddd solid;}
#main {padding: 1em 0}
```

2 Spalten: Mit der ersten Deklaration wird `#col1` 75 Prozent der verfügbaren Breite zugewiesen und die *float*-Richtung nach rechts gesetzt. Damit wird aus `#col1` die Hauptinhaltsspalte. Der Container `#col2` wird nicht benötigt und daher abgeschaltet. Abschließend wird `#col3` über die Anpassungen der Außenabstände (margin) an den linken Rand geschoben.

Spaltentrenner: Zusätzlich wird in diesem Beispiel eine 1 Pixel breite Punktlinie als vertikaler Spaltentrenner gefordert. Dies geschieht mittels der CSS-Eigenschaft `border` für die statische Spalte `#col3`. Der geforderte Abstand der Trennlinie von Header und Footer wird über obere und untere Randabstände des Containers `#main` erreicht.

Anpassungen für den Internet Explorer

Die Anpassungen für den Internet Explorer sind in der Datei `patch_2col_left_seo.css` im Verzeichnis `css/patches` abgelegt. Aufgrund des Einsatzes der grafikfreien Spaltentrenner kann in diesem Layout der 3-Pixel-Bug nicht beseitigt werden.

```
/* LAYOUT-UNABHÄNGIGE ANPASSUNGEN ----- */
@import url(../../../../../yaml/core/ie hacks.css);

/* LAYOUT-ABHÄNGIGE ANPASSUNGEN ----- */
@media screen
{
/* min-width / max-width for IE

* html #page_margins {
  width: 80em;
  width: expression( ... );
}
```

Als erstes wird die globale Anpassungsdatei `ie hacks.css` aus dem Ordner `yaml/core/` eingebunden. Lassen Sie sich bitte von den relativen Pfaden in diesem Beispiel nicht irritieren. Diese sind nur der Verzeichnisstruktur des Beispielordners geschuldet.

Im Anschluss daran finden Sie wiederum die Einbindung der IE-Expressions zur Simulation von `min-width` und `max-width` im IE 5.x und 6.

Hinweis: Wenn Sie sich das Beispiel im IE5.01 anschauen, werden Sie bemerken, dass einige Randabstände auf Null fallen. Im IE5.01 kollabieren einige paddings. Die entsprechenden Korrekturen sind in diesem Beispiel jedoch nicht eingearbeitet, da Sie für das Verständnis der Anwendung von YAML nicht erforderlich sind.

4.9.3 Layoutentwurf "3col_fixed_seo"

In diesem Layoutentwurf mit der Bezeichnung **3col_fixed_seo** werden folgende Anforderungen gestellt:

- Suchmaschinen-optimiertes 3-Spalten-Layout (Spaltenanordnung 2-1-3)
- Gesamtbreite 960 Pixel (240 | 480 | 240 Aufteilung)
- Weitere Unterteilung der Hauptinhalte in zwei Spalten nach dem ersten Absatz
- Spaltenhintergrund links: Hintergrundgrafik mit "Faux Columns" Technik.
- Horizontale Hauptnavigation "Shiny Buttons"
- Drucklayout: Ausgabe der Hauptinhalte aus #col1
- Grundlage für das Screenlayout bildet das 3-spaltige Basislayout, des YAML-Frameworks



examples/05_layouts_advanced/3col_fixed_seo.html

Layoutentwurf im Detail

Das zentrale Stylesheet *layout_3col_fixed_seo.css* beinhaltet folgende CSS-Bausteine:

```
/* import core styles | Basis-Stylesheets einbinden */
@import url(../../yam1/core/base.css);

/* import screen layout | Screen-Layout einbinden */
@import url(../../yam1/navigation/nav_shinybuttons.css);
@import url(screen/basemod.css);
@import url(screen/basemod_3col_fixed_seo.css);
@import url(screen/content.css);

/* import print layout | Druck-Layout einbinden */
@import url(../../print/print_001_draft.css);
```

Als erstes folgt die Einbindung des Basisstylesheets *base.css* aus dem Ordner *yam1/core/* sowie die unverändert verwendete Navigation *nav_shinybuttons.css*.

Anschließend wird Grundversion des Screenlayouts *basemod.css* importiert, welche die Grundlage des Layouts bildet. Die Modifikationen, entsprechend des Anforderungsprofils für das gewünschte zweispaltige Layout befinden in der Datei *basemod_3col_fixed_seo.css*.

Fixe Breite und Zentrierung des Layouts: Die fixe Layoutbreite von 960 Pixeln wird an den äußersten Container `#page_margins` übergeben. Anschließend kann dieser Container zentriert werden, indem die seitlichen Margins auf den Wert `auto` gesetzt werden. Die minimalen und maximalen Breiten werden abgeschaltet, da sie in einem fixen Layout wirkungslos sind.

```
/* Festlegung der Layoutbreite und Zentrierung */
#page_margins {
```

```
width: 960px;
min-width:inherit;
max-width:none
}
```

Spaltenanordnung 2-1-3: Die Technik für die Umsortierung der Spalten habe ich bereits im [Abschnitt 4.4: Freie Spaltenanordnung](#) erklärt. Jetzt wird sie angewendet, um eine optimale Zuordnung der Inhalte im Quelltext entsprechend ihrer Relevanz zu erhalten.

```
/* #col1 wird zur mittleren Spalte */
#main {width:100%; float:left;}

#col1 { width: 480px; float:left; margin-left: 240px; }
#col1_content {padding-left: 10px; padding-right: 10px}

/* #col2 wird zur linken Spalte */
#col2 { width: 240px; float:left; margin-left: -720px; }
#col2_content {padding-left: 20px; padding-right: 10px}

/* #col3 wird zur rechten Spalte */
#col3 { margin-left: -5px; margin-right: 0; width: 240px; float:right; }
#col3_content {padding-left: 10px; padding-right: 20px}
```

Achten Sie auf die Deklaration von #col3. Der Container wird hierbei zum floatenden Container. Mit diesem Trick kann der 3-Pixel-Bug des IE 5.x und IE6 von vorn herein vermieden werden. Auch werden hierdurch die Freiheiten des Webdesigners nicht eingeschränkt, da die Spaltenanordnung 2-1-3 von Natur aus nur in Verbindung mit reinen pixel- oder prozentbasierten Layouts umsetzbar ist, siehe [Abschnitt 4.4](#).

Faux Columns Hintergrund: Die am linken Rand angeordnete Floatspalte #col2 soll einen durchgehenden Spaltenhintergrund erhalten. Dazu bietet sich die Faux Columns Technik an. Dazu erhält der Container #main die Grafik als linksbündig platziertes und vertikal zu wiederholendes Hintergrundbild.

```
/* Hintergrundgrafik für linke Spalte - Grafikbreite 240 Pixel */
#main {
background-color: transparent;
background-image: url(../../images/bg_pattern.png);
background-repeat:repeat-y;
background-position:left;
}
```

Damit ist das Layout soweit vollständig. Fehlt nur noch der Internet Explorer.

Anpassungen für den Internet Explorer

Die Anpassungen für den Internet Explorer sind in der Datei *ie hacks_3col_fixed_seo.css* im Verzeichnis *css/patches/* abgelegt. Im ersten Schritt wird die globale Anpassungsdatei *ie hacks.css* eingebunden.

```
/* Layout-unabhängige Anpassungen ----- */
@import url(../../../yaml/core/ie hacks.css);

/* Layout-abhängige Anpassungen ----- */
@media screen
```

```

{
    /*
    ** IE5.x/Win - x
    ** IE6      - 0
    ** IE7      - 0
    */

    * html #col3 {margin-left: -3px}
    * html #col3 {ma\rigin-left: 0px}
}

```

Dank der Vorsorge über die *ie hacks.css* hält sich der IE bei diesem recht komplexen Layout erfreulicherweise mit weiteren Ärgernissen sehr zurück. Lediglich der veraltete IE 5.x kann es nicht lassen, zu nerven.

Mit dem negativen Margin werden Positionierungsfehler im IE5.x ausgeglichen. Obwohl #col3 nach rechts floatet und damit der 3-Pixel-Bug nicht aktiv wird, ist hier trotzdem ein Eingreifen erforderlich. Der negative Margin ermöglicht ein Überlappen der seitlichen Container, was jedoch aufgrund des pixelgenauen Layouts ebenfalls nicht eintritt. Die Deklaration hat demzufolge keine optischen Auswirkungen.

Alternative Lösung zur Zentrierung fixer Layouts (IE5.x tauglich)

Die in diesem Layoutentwurf verwendete Methode zur Zentrierung funktioniert in allen modernen Browsern, egal ob es sich um ein fixes oder flexibles Layout handelt. Im Internet Explorer 5.x wird das Layout jedoch linksbündig angezeigt.

Für fixe Layouts bietet sich folgende alternative Möglichkeit der Zentrierung, die auch im veralteten Internet Explorer 5.x funktioniert.

```

body { padding: 0em; }

#page_margins {
    width: 960px;
    min-width:inherit;
    max-width:none

    position:absolute;
    top: 0;
    left: 50%;
    margin-left: -480px;
}

#page { width: 960px; margin: 1em; }

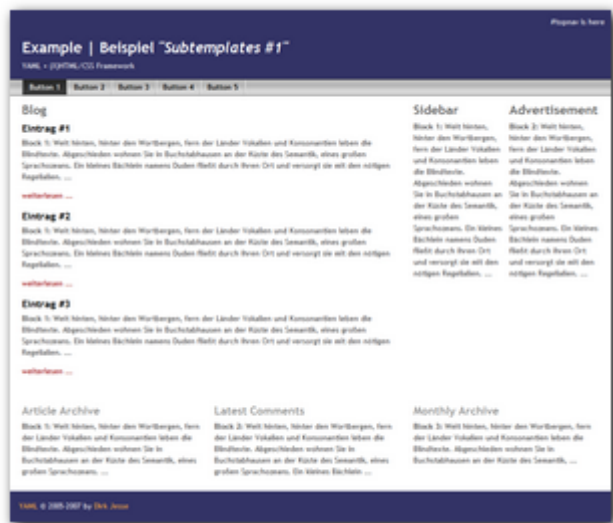
```

Hinweis: Dabei wird die Webseite durch Angabe eines negativen Margins zentriert. Diese Variante ist jedoch für flexible Layouts nicht einsetzbar.

4.9.4 Layoutentwurf "Flexible Grids"

Nicht immer erfüllt ein "normales" Spaltenlayout die Anforderungen an aktuelle Designvorschläge. Immer öfter werden flexiblere Systeme benötigt, um eine Webseite in Bereiche zu unterteilen. Hierfür hat sich die Bezeichnung "Grids" durchgesetzt, welche die Layoutgestaltung über vorgegebene Raster meint.

Mit YAML lässt sich dieses Konzept sehr einfach und komfortabel unter Verwendung der Subtemplates realisieren. Die Subtemplates ermöglichen eine flexible Raumaufteilung in Prozentwerten und ermöglichen gleichzeitig die Schachtelung von Subtemplates ineinander. Das Layoutbeispiel "flexible_grids" soll hierfür einen Ausblick der flexiblen Gestaltungsmöglichkeiten solcher rasterbasierten Layouts geben.



examples/05_layouts_advanced/flexible_grids.html

Layoutentwurf im Detail

Das zentrale Stylesheet *layout_grids.css* beinhaltet folgende CSS-Bausteine:

```
/* import core styles | Basis-Stylesheets einbinden */
@import url(../../yam/core/base.css);

/* import screen layout | Screen-Layout einbinden */
@import url(../../yam/navigation/nav_shinybuttons.css);
@import url(screen/basemod.css);
@import url(screen/content.css);

/* import print layout | Druck-Layout einbinden */
@import url(../../yam/core/print_base.css);
```

Als erstes folgt die Einbindung des Basisstylesheets *base.css* aus dem Ordner *yam/core/* sowie die unverändert verwendete Navigation *nav_shinybuttons.css*.

Anschließend wird Grundversion des Screenlayouts *basemod.css* importiert, welche auch hier die Grundlage des Layouts bildet. Für die Druckausgabe wird in diesem Fall lediglich die *print_base.css* aus dem Ordner *yam/core/* eingebunden.

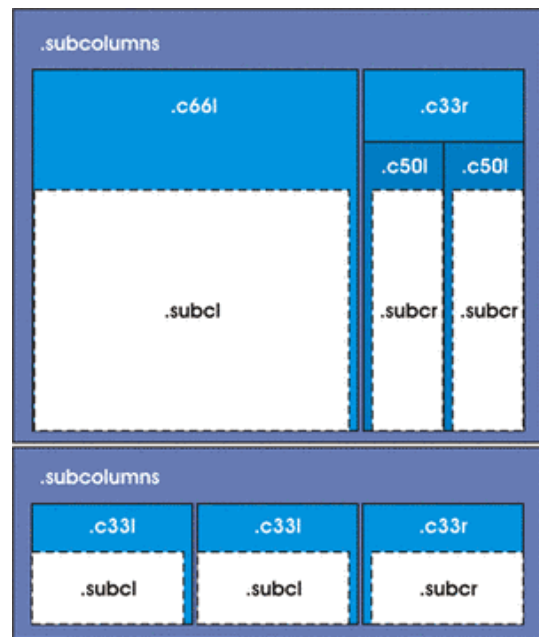
Wichtig: Subtemplates werden im Druck standardmäßig nicht linearisiert, da ihr Verwendungszweck zu vielfältig und nicht die Layouerstellung begrenzt ist. Für die praktische Anwendung zur Layoutgestaltung muss die Linearisierung der Subtemplates der im Printlayout benötigten Container durch den Seitenersteller individuell geregelt werden.

Umsetzung des Rasterkonzepts

Die Umsetzung dieses Entwurfs geschieht in erster Linie über den Einbau der notwendigen Subtemplates-Container in den HTML-Quelltext. Wie Sie der nebenstehenden Abbildung entnehmen können, werden die Content-Spalten des Basislayouts offensichtlich überhaupt nicht mehr benötigt.

Die Bezeichnung "Subtemplates" impliziert eigentlich, dass diese Bausteine in erster Linie innerhalb der Content-Spalten zum Einsatz kommen. Mit diesem Ziel wurden sie auch entwickelt. Doch die Möglichkeit der Schachtelung macht sie für die die Layoutentwicklung besonders interessant.

Ich breche daher an dieser Stelle mit der starren Quelltext-Struktur des YAML-Framework und ersetze die Content-Spalten `#col1` bis `#col3` vollständig.



Aufbau des oberen 66/33 Blocks

Als erstes muss dafür gesorgt werden, dass der obere und untere Block gleich ausgerichtet sind. Daher wird für den oberen Block die Teilung 66% | 33% verwendet, während der untere Block eine 33% | 33% | 33% Teilung erhält. Somit sind die jeweils rechten Container beider Blöcke immer gleich breit. Im zweiten Schritt wird der 33%-Container des oberen Block durch ein weiteres Subtemplate nochmals in zwei gleichbreite Bereiche unterteilt. Die Content-Container werden so eingefügt, dass sich vertikal jeweils bündige Abschlüsse im oberen und unteren Block ergeben.

Aufbau des unteren 33/33/33 Blocks

Hierbei handelt es sich um ein einfaches Subtemplate mit der Teilung 33% | 33% | 33%. Die einzige Besonderheit gegenüber dem Standardaufbau ist die linksbündige Ausrichtung des mittleren Content-Blocks (`.subcl`). Der Grund hierfür ist einfach: Die Textbreite soll bündig mit dem darüber liegenden 66%-Container abschließen. Bei einer Zentrierung des Contentblocks würden unterschiedliche Randabstände entstehen.

```
<!-- #main: Beginn Inhaltsbereich -->
<div id="main">
  <a id="content" name="content"></a><!--Skiplink:Content -->

  <!-- Subtemplate: 2 Spalten mit 66/33 Teilung -->
  <div class="subcolumns">
    <div class="c66l">
      <div class="subcl">
        <h2>Blog</h2>
        ...
      </div>
    </div>
    <div class="c33r">
      <div class="subcolumns">
        <div class="c50l">
          <div class="subcr">
            <h2>Sidebar</h2>
            ...
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

```

        </div>
    </div>
    <div class="c50r">
        <div class="subcr">
            <h2>Advertisement</h2>
            ...
        </div>
    </div>
</div>
</div>
</div>
</div>

<!-- Subtemplate: 3 Spalten mit 33/33/33 Teilung -->
<div class="subcolumns">
    <div class="c33l">
        <div class="subcl">
            <h3>Article Archive </h3>
            ...
        </div>
    </div>
    <div class="c33l">
        <div class="subcl">
            <h3>Latest Comments </h3>
            ...
        </div>
    </div>
    <div class="c33r">
        <div class="subcr">
            <h3>Monthly Archive </h3>
            ...
        </div>
    </div>
</div>
</div>
<!-- #main: Ende -->

```

Die Anzahl der für dieses Layout erforderlichen DIV-Ebenen ist relativ groß. Allerdings ist basiert es vollständig auf flexiblen Breiten und passt sich somit optimal an die Bildschirmverhältnisse an. Die Gesamtbreite erhält wiederum der Container `#page_margins`. Die Raumaufteilung innerhalb von `#main` übernehmen die Subtemplates automatisch. Sicherlich lässt sich dieses DIV-Konstrukt bei Verwendung fixer Breiten vereinfachen. Aber eben nur dann.

Anpassungen für den Internet Explorer

Die Anpassungen für den Internet Explorer sind in der Datei `ie hacks_subtemplates.css` im Verzeichnis `css/patches` abgelegt.

```

/* LAYOUT-UNABHÄNGIGE ANPASSUNGEN ----- */
@import url(../../../../../yaml/core/ie hacks.css);

/* LAYOUT-ABHÄNGIGE ANPASSUNGEN ----- */
@media screen
{
    /* Keine layoutabhängigen Anpassungen erforderlich */
}

```

Spezielle Anpassungen für den Internet Explorer sind in diesem Fall nicht erforderlich, da die Subtemplates ein fester Bestandteil des Frameworks sind. Die Anpassungen beschränken sich daher auf die Einbindung der *ie hacks.css*.

5 Hinweise

5.1 Hilfsmittel

Neben dem YAML-Framework und den Beispiellayouts liegen dem Download-Paket im Ordner *tools/* weitere kleine Hilfsmittel bei, die die Arbeit der Layouterstellung erleichtern können.

5.1.1 Dynamisch generierte Blindtexte

/tools/javascript/ftod.js

Dieses kleine Javascript-Tool generiert Blindtexte (Lorem ipsum ...) innerhalb beliebiger DIV-Container. Dabei können über zwei dynamisch eingefügte Textlinks Blindtext-Absätze hinzugefügt oder entfernt werden.

Das Script kommt bei verschiedenen Layoutbeispielen im Ordner *examples/* des Download-Paketes zum Einsatz. Die Anwendung des Scripts ist einfach. Die Einbindung erfolgt im Header der Webseite:

```
<script type="text/javascript" src="dein_pfad/ftod.js"> </script>
```

Direkt im Anschluss daran erfolgt die Konfiguration — die Festlegung in welchen Bereichen der Webseite die Blindtexte generiert werden sollen. Dazu muss das betreffende HTML-Element über eine eindeutig zuordenbare ID identifizierbar sein.

In den Layoutbeispielen sind dies die Container `#col1_content` bis `#col3_content` der drei Content-Spalten des Basislayouts.

```
<script type="text/javascript">
window.onload=function(){ AddFillerLink( "col1_content", "col2_content",
"col3_content"); }
</script>
```

5.1.2 Bearbeitungshilfen für Dreamweaver

Der *Dreamweaver* ist eines der meist verbreiteten Softwaretools zur Erstellung von Webseiten. Allerdings hat er auch noch in der Version 7 einige Schwierigkeiten, YAML-basierte Layouts in der WYSIWYG-Ansicht seines Editors darzustellen.

Dreamweaver MX 2004 (V7.0)

/tools/dreamweaver_7/base_dw7.css

Der *Dreamweaver* bietet für die Bearbeitung von Webseiten im WYSIWYG-Mode des Editors die Möglichkeit, alternative Stylesheets für diese sogenannte "Entwurfsphase" zu verwenden. Diese Stylesheets werden also ausschließlich für die WYSIWYG-Darstellung im Editor verwendet und stellen somit eine Möglichkeit dar, die Anzeigeprobleme des Dreamweavers bei komplexen CSS-Layouts zu umgehen.

Für den Dreamweaver MX 2004 finden Sie im Verzeichnis *tools/dreamweaver_7/* ein alternative Basis-Stylesheet *base_dw7.css*, in welchem die für den Dreamweaver problematischen Deklarationen auskommentiert sind. Die Einbindung ist in der beiliegenden *lies_mich.txt* beschrieben.

Dreamweaver 8

In der aktuellen Version hat sich die Darstellungsqualität im WYSIWYG-Mode des Editors weiter verbessert, sodass in der Regel keine speziellen Anpassungen der CSS-Bausteine von YAML erforderlich sind. Das derzeit einzige bekannte Problem beruht auf einem Bug der Verarbeitung der @media-Regeln im Dreamweaver. Dadurch kann es passieren, dass der Dreamweaver Regeln der Screen-Ansicht mit Regeln für den Druck überlagert.

Sollte dies der Fall sein, reicht es in den meisten Fällen aus, die Print-Stylesheets in der Entwurfsansicht auszublenden. In der Datei *lies_mich.txt* im Ordner *tools/dreamweaver_8/* sind die notwendigen Arbeitsschritte dafür beschrieben.

5.2 Tipps zum Entwurf flexibler Layouts

Zum Abschluss noch einige Hinweise, die bei der Erstellung flexibler Layouts beachtet werden sollten.

5.2.1 Umgang mit großen Elementen

Es ist wichtig, die Funktionsweise eines Spaltenlayouts auf Basis von *float*-Umgebungen zu verstehen. Die statische Spalte *#col3* "umfließt" die beiden *float*-Spalten *#col1* und *#col2* (auch wenn das im Layout nicht sichtbar wird).

Hintergrund: Der Internet Explorer ist in diesem Punkt als einziger Browser fehleranfällig bei der Handhabung von Elementen, die zu breit für die statische Spalte *#col3* sind. In diesem Fall wird die gesamte Spalte *#col3* vertikal unter die *float*-Spalten geschoben bzw. sogar ausgeblendet. Das Layout wird dadurch zerstört — die Funktion der Webseite ist beeinträchtigt. Sie finden hierzu im [Abschnitt 3.5.2](#) verschiedene Lösungsansätze.

Alle anderen modernen Browser lassen hingegen die zu breiten Elemente in die Nachbarspalten hineingleiten und brechen die Spalte nicht um. Dadurch bleibt das Layout intakt. Generell sollte bei flexiblen Layouts dieser Problematik erhöhte Beachtung geschenkt werden denn auch bei minimaler Layoutbreite sollten die Inhalte immernoch ausreichend Platz in Ihren Spaltencontainern finden.

5.2.2 Kleine Bildschirmgrößen

Flexible Layouts passen sich der zur Verfügung stehenden Breite an. Die Formatierung (Abstände, Größen) der Content-Elemente sollte daher auf eine sinnvolle Mindestbreite hin abgestimmt werden.

Eine übliche Untergrenze für die Bildschirmdarstellung stellt die SVGA-Auflösung mit 800x600 Pixel dar. Bei dieser Auflösung verbleibt ein maximal nutzbarer Viewport im Browserfenster von ca. 760 Pixeln, da Fensterrahmen und ggf. erforderlicher vertikale Scrollbalken des Betriebssystems die verfügbare Breite reduzieren. Die Beachtung dieses Umstandes ist wichtig, da horizontale Scrollbalken unter allen Umständen vermieden werden sollten.

Alle Content-Elemente (Überschriften, Tabellen, Formulare, Grafiken) sollten auf diese Mindestbreite abgestimmt sein, so dass das Layout fehlerfrei und ohne Überlappung von Elementen dargestellt wird.

Für noch kleinere Auflösungen, wie sie beispielsweise auf PDA's üblich sind, sollte über die CSS-Regel `@media handheld` ähnlich wie bei der Druckvorbereitung ggf. ein spezielles Mini-Layout definiert werden. Für die Darstellung auf mobilen Geräten empfiehlt sich die Linearisierung des Spaltenlayouts, sodass diese wie im Drucklayout untereinander angezeigt werden.

5.2.3 Flexible Randspalten

Die Breite der statischen Spalte `#col3` ergibt sich innerhalb eines flexiblen Layouts in der Regel automatisch aus der Gesamtbreite des Browserfensters abzüglich der Breite der dargestellten *float*-Spalten. Sollen auch die *float*-Spalten `#col1` oder `#col2` eine flexible Breite erhalten, so sind dabei Maßeinheiten *EM* oder *Prozent*-Angaben möglich.

Bei Verwendung der Maßeinheit *EM* für die *float*-Spalten ist jedoch folgender Umstand zu beachten: Die *float*-Spalten dehnen sich immer in Richtung der statischen Spalte `#col3` aus. Mit steigendem Textzoom wird somit die Spalte `#col3` übermäßig gestaucht, denn erstens wächst die Schriftgröße in allen Containern und zweitens wächst die Breite der *float*-Spalten proportional zur Schriftgröße aufgrund des Bezugs auf *EM*. Für die `#col3` bleibt an Breite nur, was die *float*-Spalten an Platz "übrig lassen".

Für flexible *float*-Spalten empfehle ich daher Maßangaben in Prozentwerten. Damit bleiben die Proportionen zwischen den Breiten der einzelnen Spalten unabhängig von Schrift- und Fenstergröße konstant.

5.3 Bekannte Probleme

5.3.1 Internet-Explorer 5.x: Kollabierender Margin bei `#col3`

	IE 5.x/Win	IE 5.x/Mac	IE 6	IE 7
Bug aktiv	Ja	Unbekannt	Ja *)	Ja *)

*) Der Bug ist prinzipiell auch in diesen Browserversionen vorhanden, wird jedoch durch das spezielle IE-Clearing (siehe [Abschnitt 2.7: Das Clearing der Spalte #col3](#)) unterdrückt.

Beschreibung: Die Spalte `#col3` besitzt die Eigenschaft `width:auto`. Der Internet Explorer vergibt an diesen Container daher das Merkmal `hasLayout = false`.

Im speziellen Fall dass im 3-Spalten-Layout die linke Spalte die kürzeste und die rechte Spalte gleichzeitig die längste der drei Spalten ist, kollabiert im Internet Explorer der linke Außenabstand (Margin) von `#col3`.

Dies führt dazu, dass ein eventuell vorhandener Rahmen an `#col3` (grafikfreie Spaltentrenner) zwischen `#col1` und `#col3` an den linken Seitenrand rutscht. Ein eventuell definierter Hintergrund von `#col3` wird ebenfalls bis zum linken Seitenrand ausgedehnt. Auf die eigentlichen Inhalte des DIVs (Texte, Grafiken usw.) hat diese Verbreiterung im YAML-Layout keinen Einfluss, da `#col3` durch

die vorgegebene Sortierung via `z-index` hinter den Randspalten liegt. Der Bug kann auf der folgenden Testseite nachvollzogen werden.

Testseite: [ie_bug.html](#) (Bug ist nur im IE5.x sichtbar!)

Workaround 1: Die sichtbaren Folgen des Bugs können vermieden werden, in dem als linken Spaltentrenner eine Grafik verwendet und diese als Hintergrundgrafik z.B. in `#main` definiert wird. Des Weiteren sollte `#col3` kein Hintergrund (Grafik oder Farbe) zugewiesen werden (siehe Abschnitt 4.9: Layoutentwurf "3col_fixed"). Dies kann bei Bedarf ebenfalls in `#main` oder `#page` erfolgen.

Workaround 2: Alternativ hilft hier die Aktivierung von `hasLayout = true` für `#col3` innerhalb der Anpassungsdatei für den Internet Explorer das Problem beseitigen:

```
...
#col3 {height: 1%;}
...
```

Mit dem Einsatz dieses CSS-Hacks sind jedoch die grafikfreien Spaltentrenner nicht mehr einsetzbar. Daher muss auch bei diesem Workaround generell auf den Einsatz von Hintergrundgrafiken ([Faux-Columns](#) Technik) zurückgegriffen werden.

Hinweis: Mit der YAML Version 2.5 wurde dieser Bug im IE6 und IE7 beseitigt. Auf Grund der nur noch sehr geringen Verbreitung des IE 5.x in der heutigen Zeit stellt er kein großes Problem dar. Der Bug behindert zudem nicht die Zugänglichkeit der Webseite.

5.3.2 Mozilla & Firefox

Mozilla-Browser bis Version 1.7.0: In der Render-Engine des Mozilla-Browsers war bis zur Version 1.7.0 (ebenso im Firefox-Browser bis zur V1.0) ein [Float Clearing Bug](#) enthalten. Dieser führte dazu, dass an `#col3` gebundene Spaltentrenner nicht bis zur Fußzeile geführt wurden falls eine der Randspalten länger als die mittlere Spalte wird. Auf grafische Elemente, die als Hintergrundbilder eingefügt werden, hatte dies keinen Einfluss.

Bugfix: Der Bug ist seit V1.7.1 vom Juli 2004 nicht mehr existent und kann daher mittlerweile vernachlässigt werden.

5.3.3 Netscape

Netscape 6 & 7: Die Browserversionen 6.x basieren auf unfertigen Beta-Versionen des Mozilla-Browsers und sind daher extrem fehlerbehaftet. Obwohl die Version 7 offiziell die Rendering-Engine des Firefox 1.0.1 bzw. 1.0.2 beinhaltet, gibt es auch hier an verschiedenen Stellen CSS Kompatibilitätsprobleme, insbesondere mit den Versionen 7.0 und 7.1.

YAML unterstützt den Netscape-Browser ab der Version 8 offiziell. In dieser Version wurden keine Darstellungsfehler YAML-basierter Layouts festgestellt.

Netscape 7: overflow-Bug

Das markupfreie Clearing mittels `overflow:hidden` führt bis zum Netscape 7.1 zu Darstellungsfehlern bei der Anwendung auf statischer Boxen. Aufgrund dessen werden z.B. die

Inhalte der Subtemplates nicht dargestellt. Der nachfolgende Workaround ermöglicht die Beseitigung dieses Bugs.

Workaround: Allgemein ist es ausreichend, den betreffenden Container floaten zu lassen. In diesem Fall müssen Sie jedoch dafür sorgen, dass der Container die volle verfügbare Breite einnimmt, um störende Nebeneffekte in Ihrem Layout zu vermeiden.

Falls Sie mit Subtemplates arbeiten und Browser mit diesen alten Gecko-Engines unterstützen müssen, ist es ausreichend mit der CSS-Klasse `.subcolumns_oldgecko` zu arbeiten anstatt mit `.subcolumns`. In dieser alternativen CSS-Klasse ist der eben beschriebene Float-Hack bereits integriert.

Hinweis: Im Falle, dass bei Verwendung von `.subcolumns_oldgecko` nachfolgende Inhalte neben dem Subtemplate gerendert werden statt unterhalb (ist bisher nur bei Tabellen bekannt), sollte dem betreffenden Elementen die Eigenschaft `display:inline` zugewiesen werden.

5.3.4 Opera

Opera 9.01 Bug: Die Opera-Version 9.01 beinhaltet einen Hover-Bug, der dazu führt das Margins zwischen einem clearenden Element und dem nachfolgenden Element kollabieren. In der aktuellen Version 9.02 ist der Bug nicht mehr enthalten.

Workaround: Statt des Einsatzes von Margin können Abstände mit Padding oder Bordern erzeugt werden. Auf diese Weise wird das Problem vermieden.

Opera 6: Obwohl der Opera 6 prinzipiell in der Lage ist, auf YAML basierende Designs korrekt darzustellen, kann es unter bestimmten Umständen zu unvorhergesehenen Phänomenen kommen (Bereiche, die sich nicht anklicken lassen und Ähnliches). Diese Browserbugs des Opera 6 lassen sich nicht mit vertretbarem Aufwand abstellen. Die Browser-Version kann heutzutage jedoch getrost ignoriert werden.

6 Changelog

6.1 Changelog 3.x

6.1.1 Änderungen in Version 3.0.4 [27.11.07]

Änderungen und Korrekturen

Core-Files [base.css]

- **Bessere Lösung für das Erzwingen vertikaler Scrollbalken im Firefox**
Die neue Lösung arbeitet mit `html { height: 100%; margin-bottom: 1px; }` und erzeugt lediglich einen 1 Pixel hohen Überstand, wodurch der Scrollbalken weniger störend wirkt.
- **Änderungen im Reset-CSS Block**
Das Element `cite` wurde aus dem Block entfernt. Für `blockquote` werden die Eigenschaften `font-size` und `width` nicht mehr vorgegeben. Diese Eigenschaften können vom Nutzer über die `content.css` definiert werden.
- **Container #header**
Der Container `#header` erhält die Eigenschaft `clear:both`. Dadurch wird das Vertauschen von `#header` und `#nav` im Quelltext möglich, ohne dass floatende Navigationselemente Darstellung beeinflussen.
- **Generische Klassen zur Layoutumschaltung**
Die Benennung der Klassen `.hideleft` und `.hideright` entsprach keiner sinnvollen Semantik, da die Spaltencontainer im Layout grundsätzlich frei platzierbar sind. Sie wurden daher umbenannt in `.hidecol1` und `.hidecol2`, was eine eindeutige Zuordnung erlaubt. Die Klasse `.hidenone` ist obsolete und wurde entfernt.

Core-Files [jehacks.css]

- **Verbesserung der Robustheit**
Stabilitätserhöhung für flexible Spaltenbreiten im IE5.x + IE6 durch `#main {position:relative}` bei Verwendung der JS-Expressions. Fehlpositionierungen der Spaltencontainer bei der Skalierung der Layoutbreite werden vermieden.
- **Bugfix für List-Numbering-Bug ergänzt**
Betrifft IE-Versionen 5.01 - 7.0: Wird bei Listenelementen von geordneten Listen das Merkmal `hasLayout` aktiviert, so erfolgt eine fehlerhafte Nummerierung der Elemente.

Screenlayout Vorlage [content_default.css]

- **Fix für Gecko-Probleme beim Reset von monospaced Schriften ergänzt**
Elemente mit monospaced Schriften (`textarea`, `tt`, `pre`, `code`) erhalten in Gecko-Browsern beim Reset der Schriftgrößen einen Standardwert von 13px statt 16px. Ein entsprechender Bugfix wurde ergänzt.

Layoutbeispiele [Ordner /examples/]

- **[3col_2-1-3]**
Verbesserter CSS-Aufbau ohne 3-Pixel-Bug, eingebundenes Print-Stylesheet zu `print_100_draft.css` korrigiert.

- **[3col_3-1-2]**
Verbesserter CSS-Aufbau ohne 3-Pixel-Bug, eingebundenes Print-Stylesheet zu *print_100_draft.css* korrigiert.
- **[3col_fixed_seo]**
Bugfix für `min-width` Verhalten des Safari 3 eingebaut, verbesserter CSS-Aufbau ohne 3-Pixel-Bug
- **[3col_gfxborder]**
Bezeichnungen der Eckgrafiken und -container vereinheitlicht
- **Namensraum-Kennung im <html> Element der Beispiele ergänzt**
- **Beispiele für 3-Spalten-Layouts (03_3col_layouts)**
Seitliche Paddings der Content-Container entsprechend der Position der Spalten im Layout angepasst.

6.1.2 Änderungen in Version 3.0.3 [18.08.07]

Änderungen und Korrekturen

Core-Files

- **[ie hacks.css] Bugfix für input-Elemente im IE6**
Der in V3.0 implementierte Bugfix für den Italics-Bug hatte den Nebeneffekt, Input-Elemente im IE6 willkürlich zu verlängern. Dieser Fehler ist nun korrigiert und die *slim_ie hacks.css* wurde angepasst.

6.1.3 Änderungen in Version 3.0.2 [01.08.07]

Download-Paket & Dokumentation

- [Doku en/de] URL's einiger Links innerhalb der Dokumentation korrigiert.
- [Doku de] Fehler in der Gliederung korrigiert
- [Doku de] Abschnitt 1.4: Weitere Artikel ergänzt
- [CSSDoc-Kommentare] Einrückungen innerhalb der Quelltexte korrigiert

Änderungen und Korrekturen

Core-Files

- **[base.css] Fix für fehlenden Scrollbalken in Opera 9.x**
Die negativen Abstände der Klassen `.skip`, `.hideme` und `.print` wurden auf `-1000em` verkleinert, sodass der Bug nicht mehr auftritt.
- **[ie hacks.css] Bugfixes medienabhängig geregelt**
Die Bugfixes für den *Doubled Float Margin Bug* und das *Expanding Box Problem* werden über `@media screen` auf die Ausgabe am Bildschirm beschränkt.
- **[print_base.css] Drucklayout im IE6 & Linearisierung der Subtemplates**
Subtemplates werden in der Druckausgabe nun per default linearisiert. Robustheit der Drucklayouts für IE6 verbessert.

Navigationselemente

- Anpassung der Hintergrundfarben der Listenelemente bei *nav_slidingdoor.css* und *nav_shinybuttons.css*.

Sonstiges

- Kleinere Schönheitskorrekturen in den Layoutbeispielen (Seitentitel vereinheitlicht)

6.1.4 Änderungen in Version 3.0.1 [15.07.07]

Änderungen und Korrekturen

Core-Files

- **[gefixt] Rundungsfehler in Subtemplates**
In v3.0 hatte sich eine fehlerhafte Breite bei den 33- und 66% Containern der Subtemplates eingeschlichen.

6.1.5 Änderungen in Version 3.0 [09.07.07]

Download-Paket & Dokumentation

- **Zweisprachige Dokumentation (derzeit nur online, nicht freigeschaltet)**
Die Dokumentation sowie alle Kommentare in den CSS-Dateien des Frameworks stehen in deutscher und englischer Sprache zur Verfügung.
- **Umfassende Neustrukturierung des Download-Pakets**
Im Download-Paket gibt es nun eine klare Trennung zwischen dem eigentlichen Framework, der Dokumentation sowie Layoutbeispielen und weiteren Tools. Gleichzeitig erfolgte eine Überarbeitung der Struktur des Frameworks.
- **Bessere Unterstützung von Dreamweaver 7 und 8**
Für den Dreamweaver 7 (MX 2004) liegt ab sofort ein alternatives Basis-Stylesheet bei, welches eine weitgehend fehlerfreie Darstellung YAML-basierter Layouts in der Entwurfsansicht ermöglicht. Sowohl für die Version 7 als auch für die Version 8 liegt eine *lies_mich.txt* bei, die alle erforderlichen Anpassung zur Arbeit mit YAML erläutert.
- **Umstellung aller Dateien auf Zeichenkodierung "UTF-8"**
Alle Dateien des Frameworks wurden auf die UTF-8 Zeichenkodierung umgestellt. Im Zuge der Einführung mehrsprachiger Kommentare in die Dateien des Frameworks war dieser Schritt sowohl logisch als auch unumgänglich.
- **CSS-Kommentare nach CSSDOC-Standard**
Der CSSDOC-Standard bietet eine für Menschen gut lesbare und für Maschinen auswertbare Formulierung von Kommentaren innerhalb von CSS-Dateien.
- **Zahlreiche neue Beispiellayouts**
Die Anzahl der mitgelieferten Beispiellayouts wurde deutlich erhöht. Alle Layoutbeispiele basieren auf einem neuen anprechenden Design.

Änderungen und Korrekturen

Markup

- **[geändert] Vereinfachung des (X)HTML-Quelltextes**
Die Klasse `.hold_floats` muss im Basislayout nicht mehr explizit an `#page` vergeben werden. Der Bugfix wird standardmäßig über die Datei *ie hacks.css* aktiviert.

Core-Files

- **[neu] Optimierte Stylesheets für den produktiven Einsatz**
Von den Stylesheets im *core/* Ordner des YAML-Frameworks gibt es in einer optimierten Form (geringe Dateigröße). Diese Dateiversionen sind kommentarfrei und folgen einem Kompromiss aus Lesbarkeit und geringstmöglicher Dateigröße. Im produktiven Einsatz spart der Einsatz dieser Dateiversionen wertvolle Bandbreite.
- **[neu] Alternatives Spaltenkonzept auf Klassenbasis**
Vier generische CSS-Klassen erlauben eine noch einfachere Auswahl der darzustellenden Spalten des Basis-Layouts.
- **[neu] Generische CSS-Klassen für versteckte Inhalte**
Mit den CSS-Klassen *.hideme* und *.print* stehen ab sofort zwei Optionen zur Verfügung, um Inhalte barrierefrei innerhalb des Screenlayouts zu verstecken und gleichzeitig für Screenreader und Textbrowser sichtbar zu halten. Die Klassen sind in der *base.css* definiert und stehen damit immer zur Verfügung.
- **[neu] Handhabung übergroßer Elemente im IE**
Mithilfe der CSS-Klasse *.slidebox*, definiert in der *ie hacks.css*, ist es ab sofort auch im IE5.x und 6.0 möglich, dass übergroße Elemente einfach über benachbarte Layoutbereiche hinweggleiten, ohne das Seitenlayout zu zerstören.
- **[neu] Neuer Bugfix für Italics-Bug des Internet Explorers**
Ein neuer universeller Bugfix in der *base.css* löst das Problem mit Italics-Schriften im IE 5.x und 6.0 umfassend. Bisher musste diesem Bug weitgehend inhaltsbezogen nachgegangen werden.
- **[neu] IE7 Bugfix für den Druckausgabe**
Der IE7 hat Probleme beim Ausdruck des Containers *#col3* aufgrund der fehlenden Eigenschaft 'hasLayout' und erzwingt dadurch Seitenumbrüche. In die Datei *ie hacks.css* wurde ein passender Bugfix aufgenommen.
- **[neu] Bugfix für Firefox2 overflow:hidden Bug bei Druckausgabe**
Der Firefox hat in Version 2.x Probleme im Umgang mit der Eigenschaft *overflow:hidden* bei der Druckausgabe. Für die generische Klasse *.floatbox* wurde ein passender Bugfix in die Datei *print_base.css* aufgenommen.
- **[geändert] Min-/max-width Unterstützung für IE 5.x und IE6**
Die Scriptlösung über Expressions wurde überarbeitet, sodass der IE ab sofort nicht mehr in den Quirks Mode versetzt werden muss und zudem mit EM-basierten Werte gearbeitet werden kann.
- **[geändert] Subtemplates**
Das CSS der Block- und Content-Container wurden vereinfacht. Der Einschluss der Inhalte erfolgt über die *float*-Eigenschaft der Block-Container. Dadurch werden übergroße Inhalte nicht mehr abgeschnitten. Des weiteren wurde der Ausgleich von Rundungsfehlern überarbeitet, sodass *.subcolumns* kein übergroßer Container (> 100%) mehr ist. Die alternative Klasse *.subcolumns_oldgecko* ermöglicht die Unterstützung alter Gecko-Browser (z.B. Netscape < Version 7.2).
- **[geändert] Skip-Link-Navigation**
Die Skip-Links werden ab sofort bei Aktivierung über die Tabnavigation im Browser sichtbar. Dieses Verhalten in Bezug auf Barrierefreiheit des Layouts gefordert.
- **[geändert] Überarbeitete Print-Stylesheets**

Alle layoutunabhängigen Anpassungen für den Druck wurden in den eigenständigen CSS-Baustein *print_base.css* ausgelagert, der über die Print-Stylesheets geladen wird. Damit verbessert sich die Übersicht und individuelle Anpassungen werden erleichtert.

- **[geändert] Hovereffekte für Links im IE7**
Hover-Effekte werden ab sofort im IE7 nicht mehr pauschal über die *ie hacks.css* blockiert.
- **[entfernt] Altes IE-Clearing (bis V2.4) wird nicht mehr unterstützt**
Die CSS-Deklarationen für die alte CSS-Klasse `clear_columns` wurde aus der *base.css* entfernt.
- **[entfernt] Hacks für IE-Mac aus dem Projekt entfernt**
Der IE/Mac erhält aufgrund der Verwendung von Conditional Comments und der `@media`-Regel generell weder normale Stilanweisungen noch die IE-Anpassungen. Die betreffenden Mac-Hacks (spezielle Kommentare) in der *ie hacks.css* waren daher eher verwirrend und wurden entfernt. YAML unterstützt diesen veralteten Browser nach wie vor nur, indem die Inhalte ohne jegliche CSS-Formatierungen dargestellt werden.

Navigationselemente

- **[neu] Navigationselemente allgemein**
Alle mitgelieferten Navigationslisten unterstützen ab sofort die Tabnavigation korrekt, einschließlich der Hervorhebung des aktuell anvisierten Menüpunktes.
- **[neu] Navigationselemente allgemein**
Der aktive Menüpunkt aller Navigationselemente kann neben der bisherigen Vorgehensweise über die ID `#current` alternativ auch mittels `strong` gesetzt werden.
- **[neu] Erweiterung der vlist-Navigation**
- Die vlist-Navigation verfügt nun über 4 statt den bisherigen 2 Gliederungsebenen.
- **[entfernt] Die Navigation "Sliding Door I" entfernt**
- Die Version "Sliding Door II" ist weiterhin vorhanden und in *nav_sliding_door.css* umbenannt.

Content-Gestaltung

- **[neu] Neuer CSS-Baustein `content_default.css`**
- Die Datei *content_defauft.css* befindet sich im Verzeichnis *yaml/screen/* und stellt Basisformatierungen für alle gebräuchlichen Inhaltselemente bereit und kann optional eingebunden werden.
- **[neu] Generische CSS-Klassen für Content-Gestaltung**
Ab sofort stehen über den Baustein *content_defauft.css* die drei CSS-Klassen `.note`, `.important`, `.warning` zur Hervorhebung von Inhalten bereit.

Sonstiges

- **[neu] Debug-Stylesheet**
Ein neues optionales Stylesheet *debug.css* erleichtert das Debugging des Layouts (siehe [Abschnitt 4.8: Entwurf und Fehlersuche](#)). Vordefinierte CSS-Klassen zur Darstellung von Pixelrastern, Transparenzen oder Hintergrundfarben ermöglichen eine einfache Hervorhebung/Kontrolle von Layoutelementen. Weiterhin warnt das Stylesheet automatisch den Nutzer, falls das Core-Stylesheet *ie hacks.css* nicht korrekt eingebunden ist.