Documentation

Imprint
YAML PDF-Documentation
based on YAML 3.3.1
Last change: 18.06.2011

Table of Content

1 Introduction

1.1 What is YAML?

YAML was conceived as a basis for developing flexible layouts, with a strong emphasis on meeting the challenges that result from working with elastic containers and varying units.

The most important features:

- A flexible, accessible layout concept for designing column-based and grid-based CSS layouts,
- Extensive Cross-browser compatiblity (IE 5.0/Win+) for bullet-proof layout rendering in every situation,
- The fewest possible restrictions for the designer (fixed or flexible layouts, variable column widths, etc.),
- User-defined order of content columns in the source code ("any order columns"),
- Stylesheet templates ordered by function to work efficently,
- Column separators and backgrounds all generated without images and continuous down to the footer,
- flexible grid-system via subtemplates for almost all purposes

This system allows for the rapid development of designs with one to three columns, with fixed or variable widths. With *subtemplates* (flexible grids), the column system can be endlessly nested and expanded. The YAML basis layout can be extended with special containers which help set the layout width or can create a border around the layout. But why so many containers?

There are two basic methods for creating a layout:

The Bottom-Up Principle

> The programmer starts with a blank page. The containers must all be created, positioned, and styled with CSS. A basic layout does not yet exist. While programming the layout, the designer must discover all relevant browser bugs and either avoid or hack them.

The Top-Down Principle

> Here, the programmer begins with a cross-browser-compatible, functional, modular skeleton layout, which contains all the most often-used page elements. The web designer then modifies this basic layout as he wishes and finally optimizes the XHTML and CSS code by removing unnecessary elements from the layout.

YAML was built for those working according to the second principle, and is best described with the terms "building block system" or "framework".

1.2 What is YAML not?

YAML is *not* a prepackaged layout. That would contradict the main idea behind the *Top-Down principle*. Without optimization for the demands of a particular design, the unnecessary elements (HTML / CSS) are just extra ballast.

Author's note: the YAML framework provides a cross-browser compatible basic layout as well as many helpful CSS components, allowing programmers to devote more time and energy to creative design.

Nothing is further from my intentions than a translation of the monotony of row houses into the area of web design by the repeated use of YAML as a finished layout.

Of course it is not forbidden to use YAML and all its components as a "ready to use" layout. Yet, while adjusting the code to the site's individual design, you should always keep the code as simple and clean as possible. Maintenance and bugfixing in the code will be that much easier. Unnecessary elements in the XHTML source code or the CSS files should thus be removed once the layout is final.

1.3 Advantages of the Framework

YAML is more than just a simple multicolumn layout. It is an entire layout framework, highly flexible, and tested under real-world conditions. YAML supplies diverse modules and ensures that they work together flawlessly. Here are a few advantages of the YAML framework:

Browser Compatibility

> YAML's components are all fully tested to ensure identical layouts in all browsers. All necessary hacks are already built in, minimizing the usual layout testing time for the various programs.

Building Block Principle

> The modular design allows particularly efficient layout design using the provided code.

> The basic components combine to form a basic but fully functioning layout. Additional components complete or modify this basis. These CSS components are universally usable: once written and tested, they can be built in as needed and are available for future projects.

> Examples include the simple layout variations with the *basemod* files as well as the print stylesheets.

Flexibility in layout design

> The framework design provides for much more than just a simple three-column layout. The flexible basis allows columns to be placed anywhere on the screen. The dynamic character of the floats allows even one- or two-column layouts in just a few clicks. Column and layout widths can be defined in any unit of measurement. Units can even be mixed among different column widths.

Robust Code

The XHTML and CSS construction of the individual components guarantees almost complete independence from the structure of the actual content.

The nesting of the main elements of the page in separate DIV containers ensures the correct positioning of the elements on the screen, irrespective of the later use of any particular container.

1.3.1 Updates

The YAML Framework is constantly updated. All changes in and additions to each new version are summarized in the changelog and when necessary, documented more extensively.

Updates of the framework basis are possible anytime, thanks to the organization of the CSS components and the separation of YAML and user CSS. Relaunches and redesigns are excellent opportunities for reworking YAML-based websites – or when the extended functionality of a new YAML version becomes necessary.

Important: YAML has always been built with robust and stable components. However, existing websites need not be updated with every new version. A perfectly functioning CSS layout does not need monthly security patches!

An update of the framework basis is recommended when known CSS problems can be solved with a new YAML version.

1.4 Accessibility & Web standards

The definitions of various levels of accessibility cannot be discussed here, and a thorough treatment of the advantages of using web standards goes beyond the scope of this site. Here are some highlights of YAML's usefulness in and practicability in both these areas.

Valid XHTML code and valid stylesheets

A valid skeleton structure is the basis of any website for all target audiences, regardless of any handicaps. Validity guarantees a high degree of uniformity in the presentation of the website in various browsers. The individual components of the YAML framework all found on valid XHTML and CSS code.

Extensive browser support

YAML aims to ensure a uniform presentation of a website in all browsers. The problems of the sometimes highly variable support of CSS standards, in particular the many CSS bugs in Internet Explorer, are well known. Still, as Internet Explorer is clearly the worldwide market leader, it is completely supported. It is simply not sensible to optimize a CSS framework only for supposedly standard-conform browsers.

Internet Explorer's current market share is estimated at about 90% worldwide. The percentage of IE 5.x users has fallen to below 10%. This number is close to the numbers of surfers using alternative browsers like Opera, Mozilla, or Safari. Firefox alone has won more

than 5% of internet users. Support for IE 5.x is thus just as sensible and justified as the support of modern browsers.

Doing without layout tables

Opposing opinions on layout tables are easily found online. While generally agreed that nested tables are outdated, user-unfriendly, and difficult to update, controversy still reigns over the use of tables themselves: if their (non-nested!) application is ever justifiable. The following presents a few advantages resulting from YAML's non-table layout:

- **Free choice in column order**

  The order of the column containers in the source code is completely independent of the columns' position on the screen. The accessibility of the content for text browsers and screen readers is greatly improved, as they present content linearly. Search engine placement can also benefit from this flexibility.

- **Flexibility in layout and printing**

  Individual columns can be removed from the screen layout via CSS (for one- or two-column layouts). Specific features like the navigation, sidebars, etc., can be turned off for printing purposes with the print stylesheet. In addition, column containers are easily linearized for printing: set to full page width and presented in source order.

- **Rendering speed in the browser**

  Tables are only rendered by the browser when all sections of the table have completely loaded. When using DIV containers, the browser starts rendering as soon as the first container has loaded. Pages with table layouts thus make users wait longer for content. Even today, many users still connect via modem and ISDN. Longer load times are particularly noticeable and annoying for these readers.

Applying variable size units

A further important milestone on the road to accessible websites is the use of relative units of measurement (for example in layout widths or font sizes). Accessibility problems occur for all of us, not just for those with disabilities, when fixed layouts and tiny type make reading difficult, or when web pages cannot be legibly printed. The flexible setup of all design elements (column sizes, margins, font sizes) was one of the main principles behind the development of the YAML framework.

Semantic Code

The semantically correct markup of content contributes to simpler code, easier reading in alternative browsers, and greater compatibility with future products. The YAML framework provides the design skeleton for a website, which must function *regardless* of the nature of the later content. The involvement of content elements in the layout design, which, when carefully done, could lead to fewer DIV elements for the basis layout, cannot be anticipated by YAML's framer. The optimizing of the XHTML markup and the stylesheets must lie in the hands of the web designer after the end layout is final.

Skip Link Navigation

> In addition to the possibilities of the variable column order, which allows for optimum linearization of content for text browsers and screen readers, the skip link navigation improves maneuverability on a web page equipped with links to important content elements (navigation, content area) – particularly important for screen readers.

The YAML framework provides a flexible skeleton structure, oriented to the demands of barrier-free web design and exploiting the advantages of web standards. In this context, I am proud to mention the Redesign 2006 of the website "Einfach für Alle" ("Easy for Everyone"). The website is an initiative of "Aktion Mensch" ("Action Human") and has promoted barrier-free web design for many years. The current flexible multiple-column layout from 2006 is based in great part on YAML.

Accessibility and standards could only be treated briefly here; I recommend the following online articles for those interested in more.

Further Links (in German)
BITV für Alle
Barrierefrei zum Mitnehmen
Retro-Coding: Semantischer Code ist der Anfang von gutem Design
Semantischer Code - Definitionen, Methoden, Zweifel

1.5 The Structure of the Download Package

The following describes the structure of the download package, available directly on the homepage. The package contains not only the files for the framework itself, but the complete documentation, several application examples, and a few helpful tools for developing layouts.

1.5.1 The Download Package

File/Folder	Description
documentation/	The documentation of the framework in English and in the original German, as PDF files. This is a complete copy of the online documentation from yaml.de. Read the documentation carefully and take the bold tips into account when using the framework.
examples/	This folder contains many application examples of the YAML framework with complete layout examples. The samples are organized according to various themes. The documentation explains selected examples in great detail.
js/	This folder contains a small script for dumy texts as well as the JavaScript library jQuery.
tools/	This folder contains several tools for developing layouts. The files in this folder are not necessary for the framework's functionality and need not be placed on the live server.
yaml/	This folder contains all the framework files. These are the finished, out-of-the-box CSS components as well as templates for the actual layout design. The relevance of each individual component is thoroughly explained in the documentation. Tips for using the framework in actual practice are in Chapter 4.

1.5.2 The Framework Files

The YAML framework consists of a predefined XHTML structure as well as a series of CSS files with various functions. These CSS files are in the yaml folder. In addition to the actual CSS components, this folder also contains "drafts", which you can use to design your own layout. These templates are meant to speed your implementation of YAML and simplify the first basic steps.

File/Folder	Description
/yaml/	
central_draft.css markup_draft.html	This is the trunk folder of the YAML framework. It contains the file *central_draft.css:* a so-called *central stylesheet* (see section 3.3). Via this central stylesheet, YAML embeds all the necessary CSS components in the (X)HTML source code of the website -- with the `@import` rule. The file *markup_draft.html* is also here, which contains the source code structure for the YAML framework.

/yaml/add-ons/	
accessible-tabs ***microformats*** ***rtl-support*** ***syncheight***	This folder contains several add-ons to the YAML layout frameworks. These are optional elements that aren't necessary for your work but they provide meaningful extensions to the framework core. • Accessible Tabs (jQuery plugin) • Mircoformats • RTL-Language-Support • SyncHeight (jQuery plugin)

/yaml/core/	
js/ ***js/yaml-focusfix.js*** ***base.css*** ***iehacks.css*** ***slim_base.css*** ***slim_iehacks.css***	This folder, as the name implies, contains the core CSS components for YAML. Used together with the predefined XHTML markup and the file *base.css*, these files produce a robust three-column basic layout with header and footer (see Section 3.4: The Base Stylesheet). The file *iehacks.css* contains all the CSS adjustments that are necessary for Internet Explorer (Versionen 5.x - 7.0) and are independent of the layout and structure (see Section 3.5: CSS Adjustments for Internet Explorer). It is a core component and required for every YAML-based layout. Both these basic files together ensure the browser-independent uniform display of the basic layout. Each of these stylesheets has its own *slim* version: intended for the live site, they are optimized for size. The folder *js* contains the JS-file *yaml-focusfix.js* that solves a focus problem of IE8 and Webkit-based browsers (see section 2.8)

/yaml/screen/	
basemod_draft.css ***content_default.css*** ***forms.css***	CSS components for the screen design are in this folder. *basemod_draft.css* is a template for the screen layout. It can be copied into different projects and the predefined containers within can be changed or added to with additional elements. Every YAML-based layout will incorporate one or more such basic modification (*basemod*) files via the *central stylesheet* (see section 3.6: Creating the Screen Layout as well as Chapter 4). The second file in this folder is *content_default.css*. Often-used content elements have been predefined here. This file too can be copied into any project and adjusted accordingly. More information is available in Section 3.8. *forms.css* is the CSS-part of YAMLs form construction kit. See section 3.10 for more information.

/yaml/navigation/	
images/ ***nav_shinybuttons.css*** ***nav_slidingdoorl.css*** ***nav_vlist.css***	This subfolder contains the components for the navigation. Various list navigations - horizontal as well as vertical - are provided within the YAML framework. • *nav_shinybuttons* (horizontal navigation) • *nav_slidingdoor* (horizontal navigation)

- *nav_vlist* (vertical navigation)

More information is available in Section 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* *print_draft.css*	This folder contains the CSS files for printing YAML-based layouts. These files modify the screen layout for paper. More information on print layouts is in Section 3.9: Adjusting the Layout for Print.

/yaml/patches/	
patch_layout_draft.css *patch_nav_vlist.css*	This folder contains the adjustment files for Internet Explorer. The file *patch_layout_draft.css* is a draft for such a file. Such stylesheets contain all the necessary CSS hacks for the layout in Internet Explorer and are incorporated into the website with a so-called *conditional comment* (see Section 3.5: CSS Adjustments for Internet Explorer). The second file is *patch_nav_vlist.css*, which belongs to the navigation file *nav_vlist.css* and adjusts those CSS commands for Internet Explorer. More information is available in Section 3.7.

1.5.3 Included Layout Samples

The layout examples described in the following are intended to provide a glimpse of the many varied possibilities for the application of the framework. Several of the samples are described more thoroughly in the documentation, others are meant as inspiration for solving frequently encountered design problems. The necessary YAML CSS components for each example are inside the given folders in the subfolder css. The file and folder names are intentionally rather long in order to make clear the meaning of the individual CSS components.

Datei/Verzeichnis	Beschreibung
/examples/01_layout_basics/	
3col_standard.html *building_forms.html* *multicolumnar_forms.html* *styling_content.html*	This folder contains two very simple examples. The sample *3col_standard.html* contains the YAML basis layout: a simple, flexible, 3-column layout with a horizontal navigation. Both examples *building_forms.html* and *multicolumnar_forms.html* demonstrate the use of the Form Construction Kit. For more information, please see: Section 3.10 The example *styling_content.html* demonstrates all the content elements which are influenced by the integration of the default CSS file*content_default.css*, see Section 3.8.

/examples/02_layouts_2col/	
2col_left_13.html *2col_left_31.html* *2col_right_13.html* *2col_right_31.html*	All the important combinations for 2-column YAML layouts are in this folder.
/examples/03_layouts_3col/	
3col_1-2-3.html *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*	Here are all variations of the 3-column YAML layout. More information is available in Section 4.4: Variable Column Order.
/examples/04_layouts_styling/	
3col_column_dividers.html *3col_faux_columns.html* *3col_liquid_faux_columns.html* *3col_gfxborder.html* *dynamic_layout_switching.html*	Many examples with some far-reaching graphical layout adjustments. Samples of graphic column separators and backgrounds as well as graphical borders. The section *dynamic_layout_switching.html* demonstrates the use of CSS classes to switch among various layouts, see Section 4.3.
/examples/05_navigation/	
menu_shiny_buttons.html *menu_sliding_door.html* *menu_vertical_listnav.html*	Three examples that show the use of the included navigation components. More information is available in Section 3.7.
/examples/06_layouts_advanced/	
2col_left_seo.html *3col_fixed_seo.html* *flexible_grids.html* *flexible_grids2.html* *equal_height_boxes.html*	This folder contains a few more complex layouts, oriented to the usual practical demands. In these examples, various functions of YAML are integrated and the use of subtemplates for subdividing content and for layout purposes is demonstrated. The example "*equal height boxes*" is quite a specialty. It demonstrates the creation of content containers that are all the same height by using subtemplates: see Section 4.5.
/examples/07_layouts_advanced_2/	
fullpage_3col.html *fullpage_grids.html*	This folder presents two layout examples demonstrating the multiple use of the CSS classes `.page_margins` and `.page`, using a new hierarchy for the containers.
/examples/08_special_interest/	
3col_fullheight.html *minmax_js.html*	*3col_fullheight.html* demonstrates a special case were the minimal layout height gets expanded to the full height of the viewport, even without any content. *minmax_js.html* demonstrates an alternative method to simulate CSS properties min-width and max-width in Internet Explorer 5.x and 6.0 without using JS-Expressions. More information is available in Section 4.7.

/examples/09_add-ons/	
accessible_tabs.html *rtl_support.html* *styled_microformats.html*	These layout examples demonstrate the use of the provided add-ons, that come along with the download package in folder *yaml/add-ons/*.

1.5.4 Tools for Layout Development

As already mentioned in the introduction, this folder contains a few tools for developing layouts. The files here are not necessary for YAML's function and are not part of the framework.

Datei/Verzeichnis	Beschreibung
/js/	
ftod.js	This little script creates dummy text on the fly. It is used in the application samples.
minmax.js	This script allows the use of the CSS properties `min-width` and `max-width` in Internet Explorer. More information is available in Section 4.7: Minimum & Maximum Widths.
/js/lib/	
jquery_1.3.js *jquery_1.3.min.js*	Here you'll find the latest release (december 2008) of the JavaScript library jQuery.
/tools/dreamweaver/	
base_dw7.css	Dreamweaver is a versatile and popular editor for creating web pages. Its WYSIWYG capabilities for CSS layouts are, however, somewhat restricted. The file in this folder simplifies the use of Dreamweaver's Design mode with YAML layouts. More information is available in Section 5.1.

1.6 Browser Support

- **Windows**
    - o Internet Explorer 5.x
    - o Internet Explorer 6.0
    - o Internet Explorer 7.0
    - o Internet Explorer 8.0
- **Macintosh OS**
    - o Camino 0.6 +
- **Linux**
    - o Konqueror 3.3 +
    - o Galeon 1.3 +
    - o Epiphany 1.4.8 +
    - o Lynx (Textbrowser)
- **All operating systems**
    - o Firefox 1.0 +
    - o Mozilla Suite 1.7.1 +
    - o Apple Safari 1.0.3 +
    - o Google Chrome 1.0+
    - o SeaMonkey 1.0 +
    - o Netscape 8.0 +
    - o Opera 6 +

The browsers listed here are completely supported: YAML-based layouts will be consistent in all of them. A plus sign (+) after the version number means that all later versions should work just as well with YAML.

1.7 IE 5/Mac, Netscape 4 & Co.

Internet Explorer 5 for the Macintosh and Netscape 4 — as well as all other outdated browsers — have their own special place in YAML's support.

Outdated browsers have great difficulty displaying modern CSS layouts. It makes sense to keep this actual CSS completely hidden from these browsers — as it would only confuse them — and thus still allow the user access to the actual content.

YAML's CSS building blocks use the rules of `@import` or `@media` to deal with this problem. Internet Explorer 5/Mac, Netscape 4x, and many other outdated browsers are incapable of interpreting one or the other of these rules, and so are automatically shunted away from the modern CSS declarations. Users see the complete content: unformatted, but legible.

Certain versions of Netscape are known to crash at the sight of a mere floated picture. The consistent use of the shunting principle for aged browsers allows all users access to the content.

In short: outdated browsers are supported by YAML in such a way as to allow users to read the content without being hampered by incomplete CSS interpretation. Content is visible in the browser's standard design, similar in appearance to text browser interpretations (i.e. Lynx).

1.8 Thanks

"Yet Another Multicolumn Layout" (YAML) is a one-man project, begun in spring 2005 when I needed a flexible and all-purpose basic layout for my own small website projects. It began as a hobby project, as I am a civil engineer in my day job, and am only involved with web design on the side. Jens Grochtdreis encouraged me to publish Version 1.0 in October 2005, after having already supported me in developing the framework.

Since then, the project has grown into a comprehensive and stable CSS framework and I became a book author. Public interest grows with each new version, and more and more comments and emails arrive in my inbox. This feedback is particularly important and helpful for me as a developer: I would like to thank the users for all their support.

I would like to especially thank...

- Nils Pooker for his extensive user feedback on YAML and the YAML Builder,
- Ansgar Hein for the legwork and great support on the Form Construction Kit,
- Alexander Haß for more legwork and the help with the RTL Support Add-on,
- Michael Jendryschik for the assistance with the Microformats Add-on,
- Tomas Caspers for his tips on accessibility,
- Michael Preuß for the help with the blog,
- Peter Müller for the time to discuss so many ideas by phone,
- Jens Grochtdreis for long years of advice and support,
- Dirk Ginader for the support with several jQuery ideas,
- Folkert Groeneveld for the YAML logo,
- Genevieve Cory for the translation of the documentation

Dresden, 18.Oct.2009
Dirk Jesse

2 Basics

2.1 A Comprehensive Concept

As the introductory chapter demonstrated, YAML's construction founds on many various considerations, which are most easily explained using the XHTML source code structure. YAML's high flexibility requires a certain amount of complexity, but fear ye not. This and the following chapters explain YAML's basic concept using many examples and source code snippets.

CSS can only be learned and used effectively and precisely when one knows the traps along the way. As in real life, working with CSS is not always easy peasy. Internet Explorer is outstanding in its field as far as the number of CSS bugs it contains — creating headaches for both beginners and professionals. But no pain, no gain — in spite of these bugs, you'll see that even Internet Explorer can be maneuvered into displaying accurate, modern, accessible CSS layouts.

This documentation will not merely explain YAML's use in standard-conform browsers, but when necessary includes explanations of Internet Explorer's particular problems and their possible solutions. That's my idea of a comprehensive concept.

Let us begin...

2.2 The Basics: Floats

If an element (a picture or a table) is declared to be a floated object, it is released from the normal text flow and the following elements flow around it, as if it were an obstacle in a stream. This type of positioning only requires the left-aligning or right-aligning of the element (`with float:left;` or `float:right;`) within the available space. The browser places the rest of the content around the floated object.

> **Note:** you are advised to read up on the theory of *floats* in order to better understand their functioning. I highly recommend the article "Float: The Theory" by Big John of positioniseverything.net. "Floats: Die Theorie" is the German translation by Andreas Kalt and Jens Grochtdreis.

Flexible layouts and columns with flexible widths are particularly amenable to floated objects embedded in the text, as the browser can then optimally place line breaks and content within the column.

The text flow is stopped with the CSS property `clear:value;` (Description in German). Unfortunately, as the W3C has currently defined text flow, it cannot be automatically stopped at the end of the current paragraph or the next subheadline.

Stopping the text flow thus usually requires additional and optically visible HTML code. The use of empty p or hr tags is widespread, but this is certainly not practical.

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

This is particularly disadvantageous for layouts, as those additional code elements are still displayed by the browser as unintentional vertical space.

The precise use of CSS lets us avoid this problem and makes floated environments practical for layout design. In Spring 2005, several web designers devoted themselves to this topic and published interesting ideas.

Two of these markup-free *clearing* methods are used in YAML. Both methods are explained in the following section (the right column).

2.3 Markup-Free Clearing

Effective use of *floats* was always very complicated, as extra code / markup was necessary to end the flow of text -- often in the form of inline CSS. *Floats* were thus primarily used for only the simplest layout tasks, such as arranging images.

The expanded capability of CSS 2 and CSS 2.1 and the current good browser support, the applications for *floats* are endless. The key is the *markup-free clearing* via CSS.

2.3.1 Method 1: Clearfix

The *Clearfix* Method is from Big John's article "How To Clear Floats Without Structural Markup", which thoroughly explains Tony Aslett's [csscreator.com] *clearing* method. A German translation of this tutorial is available here.

```
/* 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 */
```

IE 7 requires a minor adjustment, which is explained in the article "New clearing method needed for IE7?"

2.3.2 Method 2: Floatbox

As an alternative to Clearfix method, it is sometimes necessary to contain float objects without using the CSS property `clear`. The reason for this is that this property doesn't always work without side effects. See more details in <u>section 2.6</u>.

One such method is the CSS class `.floatbox` which is implemented in the YAML framework. In this case the CSS properties `display:table; width:100%;` does the job of containing floats. This method has proven to be robust in all modern browsers. In contrast to `overflow:hidden;` be content with this method at the element boundaries not clipped, which is of growing importance related to the many new design possibilities with CSS3.

As `display:table;` needs an explicit value for width for crossbrowser compatibility, at the same time this works as a fix for older versions of Internet Explorer (5.x to 7.0), that don't know `display:table;`, by triggering *hasLayout*, which has the same effect of containing floats.

```
/* (en) alternative solution to contain floats */
.floatbox { display:table; width:100%; }
```

More information on this topic in <u>Section 2.6: How *Floats* Work</u>.

2.4 Structure of the XHTML Source Code

The goal of the YAML framework is to deliver a universally applicable, cross-browser consistent and fully functional layout with all the necessary XHTML structures, independent of any content. In particular, page creators have been given the freedom to choose fixed or flexible layouts and column widths. Furthermore, a certain level of comfort is provided with the predefined commonly needed elements and the usual design requirements built into the structure. The result is a universal source code structure, which offers a multitude of easy modifications via CSS without changing the basic markup. The source code structure is in the download package as an empty HTML file.

<u>/yaml/markup_draft.html</u>

2.4.1 Doctype Choice

The doctype *XHTML 1.0 Transitional* was chosen for the source code structure. You may certainly change it if you wish: *Strict XHTML* or perhaps *HTML 4.01* are completely compatible with the framework should your content require them.

Standard Mode

> In this mode, the browser interprets (X)HTML as it is defined by the W3C. Mistakes in the (X)HTML code can cause major errors in presentation. However, this mode offers the greatest possible assurance that a website will be consistent in all browsers.

Quirks Mode

> This mode lets the browser tolerate much more invalid code and will always attempt to produce a usable web page. This mode is used automatically, when the HTML document specifies no Doctype, an outdated Doctype – or a misspelled one. Internet Explorer 5.x can use no other mode than this.

The chosen Doctype's presentation mode is thus crucial for a correct display of the layout -- particularly in Internet Explorer. All the YAML CSS components, including the CSS hacks for Internet Explorer, are based on the browser's using the standard-conform **Standard Mode**.

2.4.2 The Structure in Detail

Time to look at the fundaments of the YAML framework. Here is an excerpt from the file *markup_draft.html*:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de" lang="de">
<head> ... </head>

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

    <!-- begin mainpart -->
    <div id="main">

      <!-- left column -->
      <div id="col1">
        <div id="col1_content" class="clearfix">
        ...
        </div>
      </div>

      <!-- right column -->
      <div id="col2">
        <div id="col2_content" class="clearfix">
        ...
        </div>
      </div>
```

```
    <!-- middle column -->
    <div id="col3">
      <div id="col3_content" class="clearfix">
      ...
      </div>
      <!-- IE Column Clearing -->
      <div id="ie_clearing"> </div>
    </div>

  <!-- end mainpart -->
  </div>

  <!-- footer -->
  <div id="footer"> ... </div>
  </div>
</div>
</body>
</html>
```

The outermost DIV container .page_margins controls the width of the layout as a whole. It contains all the following containers, and its parameters determine the maximum and minimum widths of a flexible layout as well as the width of a fixed design.

In addition, this container together with the container .page can be used to create border visuals for the layout - more on that later. Both containers are given the IE proprietary property *hasLayout*, in order to avoid various CSS bugs, such as the Escaping Floats Bug when using horizontal menus on a *float* basis. For more information, see Section 3.5: CSS Adjustments for Internet Explorer

Next are the containers for the #header, the main navigation #nav, as well as the main content area #main with its three columns. The end of the file is: the #footer.

The red section of code labeled *IE Column Clearings* is one of YAML's special features. The meaning and function of this container is thoroughly explained in Section 2.7: The Clearing of #col3.

The following layout example demonstrates the resulting 3-column-layout:

examples/01_layouts_basics/3col_standard.html

Important: starting with version 3.1 of the YAML frameworks, .page_margins became a CSS class in order to allow its use more than once on a page. All earlier versions used the id #page_margins, which could only be used one time.

2.4.3 Variations of the Markup

As demonstrated, the two CSS classes .page_margins and .page control the basic appearance of the layout, by setting the total width as well as margins, borders, or backgrounds if necessary.

In the source code described above, the outermost DIV container .page_margins surrounds all the other elements of the webpage. This results in a layout which in its simplest form is a rectangular box against the background of the <body> element. This effect is not always desirable.

Alternatively, the classes .page_margins and .page can be used multiple times within a layout to allow a background color or image to stretch horizontally across the entire viewport with only the

actual content within a rectangle of defined width. The following layout example demonstrates the effect:

examples/07_layouts_advanced_2/fullpage_3col.html

To create this design, the nesting hierarchy of the code had to be changed. Here is an excerpt of the header markup:

```
<div id="header">
  <div class="page_margins">
    <div class="page">
      ...
    </div>
  </div>
</div>
```

While in the standard version the container `.page_margins` contains all other elements, this example changes the hierarchy so that `#header` becomes the outermost container and `.page_margins` merely contains its contents – its width definitions thus only influence its contents. The container `#header` itself, if not specifically restricted, will automatically stretch to fill the entire viewport width and can be used to design the header area.

You can use this modification on any basic element of your layout (header, navigation, main, footer) or just on specific areas, simply by having a common parent container of `.page_margins` surround all the other elements, as in the standard layout. And of course you need only set the width for `.page_margins` to set the width for the entire layout. As you have probably noticed, the (X)HTML framework is extremely flexible.

2.4.4 Design Freedom with the Combination Model

The Box Model, which has existed since CSS 1, is clearly intended for use when working with fixed measurements (i.e. pixels). The total width of a container is determined by the addition of the individual components of the model: `width`, `padding`, and `border`.

When mixing units of measurement within a container (for example `width:25%; padding: 0 10px;`), it is no longer possible to calculate the total width of the container in advance. Design freedom in composing flexible layouts is thus severely reduced.

Internet Explorer has a further problem with flexible column widths. When using *Quirks Mode*, it interprets the CSS Box Model incorrectly. IE 6 can be set to present content in a standards-conform manner with the use of the correct *Doctype*. However, YAML has always been designed to completely support version 5.x of Internet Explorer, which only works in *quirks mode*.

In order to yet persuade IE to present the correct width, the Box Model Hack was developed -- along with countless other variations of the hack. All variations have in common that they exploit the parser bug to give IE a separate width, taking into account the false calculation, which then results in a column with the correct width. Unfortunately, this method cannot correct for mixed units of measurements, because of the problems described above. It is thus a further restriction on design freedom.

The solution for all these problems lies in YAML's combination model for the basic layout - with two nested DIV containers in each column.

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

The total width of the column is assigned to the outer container `#colx`. The `padding` and optional `border` go to the inner container `colx_content`, which has no defined width, but only `width:auto`.

This means that the total width of the container `#colx` can always be determined. Any number of combinations of various units of measurements are possible, which frees the design to flexible layouts and simultaneously entirely avoids the IE box model bug.

2.5 Column Order in Source Code

Both columns `#col1` and `#col2` are *floats*. The third column, `#col3`, is a static container. The order in which these three containers appear in the source code is not variable. The *float* objects (`#col1` and `#col2`) must always come **before** the static object (the container `#col3`).

The CSS declarations of the *float* columns are in the file *yaml/core/base.css*:

```
#col1 {
    float: left;
    width: 200px; /* Standard value */
}

...

#col2 {
    float: right;
    width: 200px; /* Standard value */
}
```

The basic layout *floats* the two column containers `#col1` and `#col2` to the left and right edges, respectively, leaving `#col3` to appear in the middle of this three-column layout.

As you can see in the XHTML structure, the individual columns are not nested in additional containers (often called `wrapper`). All three column containers are within `#main` in the same structural level, but both floated columns `#col1` and `#col2` are completely cut out of the normal element flow. The static container `#col3` then takes up the entire available width between them.

CSS must still specify a few more things so that the content in `#col3` will not conflict with that in the two *float* columns. The *float* columns are set to a standard width of 200 pixels. A 200 pixel wide outer `margin` on `#col3` in combination with its `width:auto;` forces its content into the alley between the content of `#col1` and `#col2`. The CSS declarations described here are in the file: *yaml/core/base.css*.

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

Important: the order of containers `#col1`, `#col2`, and `#col3` should remain unchanged in the (X)HTML source code. Sort your content into column containers in the desired order. Their sequence is completely independent of their display on the web page. Details are available in Section 4.4: Variable Order and Use of Content Columns.

Now we've got the three containers `#col1`, `#col2` and `#col3` set up in our source code and positioned with CSS. Only one question left: why are these three columns **not nested** inside `#main`?

The answer is in Section 2.7: The Clearing of Column #col3. Before we get to that, a small detour along the way to visit float functionality.

2.6 How Floats Work

When using *floats*, it is imortant to remember that when used in static elements, the CSS property `clear: left | right | both` does not only affect its own location within the surrounding element, but works **globally** - on all the *floated* elements on the page that share the same level in the nesting hierarchy. This is easier demonstrated than explained: please see the file *global_clear.html*.

global_clear.html

Warning: Internet Explorer 5.x and 6.0 will have problems displaying this file. The IE float bugs have not been fixed here. Please try another browser (Firefox, Safari, Opera ...).

2.6.1 Layout Preparation

First we must ensure that *floated* objects can be used freely within the columns. For this, the eventual content must be completely contained within the static DIVs `#col1_content`, `#col2_content` and `#col3_content`.

For this purpose, these three containers are given the CSS class `.clearfix`. The Clearfix hack guarantees that all content (static and / or floats) is automatically enclosed. The definition of this class is found in the file *base.css*.

```
/* Clearfix Method for clearing the floats */
.clearfix:after {
   content: ".";
   display: block;
   height: 0;
   clear: both;
   visibility: hidden;
}

/* This declaration is necessary for Safari!! */
.clearfix { display: block; }
```

Important: although the class `.clearfix` is only used on block-level elements in the YAML framework, the Safari browser still needs the explicit declaration of `display:block;`. Otherwise the container `#col3_content` becomes much too narrow. This value is redundant for all other modern browsers, like Firefox or Opera.

As you can see, even while using the Clearfix Hack we're still using `clear:both;`. Within the *float* columns `#col1` and `#col2`, the `clear` property only works locally - just how we want it. Within the static container `#col3` however, `clear:both` works globally and ensures that the container `#col3_content` lengthens to reach the lower edge of the longest *float* column. This behavior is exactly what the YAML framework requires.

Unfortunately, Internet Explorer up to version 6 cannot deal correctly with the CSS pseudo-class `:after`. The clearfix method is not completely ineffective in Internet Explorer. It is used within the two containers `#col1_content` and `#col2_content` to enclose the content. A couple of adjustments are necessary for IE. These hacks are centrally maintained in the file *yaml/core/iehacks.css*. For more, see [Section 3.5: CSS Adjustments for 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; }
/*--------------------------------------------------------*/
```

As Internet Explorer cannot interpret the CSS pseudo-class `:after`, it ignores the property `clear:both;` and does not clear globally within `#col3`. A special DIV container (`#ie_clearing`) at the end of `#col3` is necessary to force IE to clear. A detailed explanation is in the following [Section 2.7: The Clearing of #col3](#).

Note: a further source of information - especially regarding the technical functioning of *floats* and in dealing with various browsers - please see the very thorough article "[Grundlagen für Spaltenlayouts mit CSS](#)" by Mathias Schäfer on the [SelfHTML-Weblog](#).

2.6.2 Preparing the Content

For the content, yet to come, we need a way to control the text flow within the static container `#col3` without triggering the global behavior of `clear:both;`. Within the floating columns `#col1` and `#col2`, the use of this property is simple, as the clearing here generally only works locally within the columns. Within `#col3`, as discussed, the effect is global and would cause large vertical gaps. Unless you can prevent it.

The solution is the overflow method, which also makes the encompassing of floats possible. The overflow method works with the property `display:table`, so no conflicts with the clearing of the columns arise. For the preparation of the content, YAML provides the CSS class `.floatbox`: its use is explained in the following two examples.

The definition of the CSS class `.floatbox` is in the file *base.css*.

```
.floatbox { display:table; width:100%; }
```

The columns can now work with any floated objects. It may be useful to restrict the text flow to a particular area, perhaps to the next section headline. This can prevent graphics from flowing into a following but separate section.

For that, we need to nest the flowing content area, again using the CSS class `.floatbox` (based on the overflow method). Two examples:

Example 1: a paragraph text should flow around a picture. The surrounding `p` tag is given the class `.floatbox`. The text flow is then restricted to this particular paragraph -- no more HTML code is necessary to stop the flow.

```
<p class="floatbox">
  <img src="picture.jpg" style="float:left;" alt="" />
  This is the text of the paragraph which flows around the picture...
</p>

<p>Here the text flow has ended. This paragraph always begins below the
picture.</p>
```

Example 2: the text of several paragraphs should flow around a picture. The flow should stop before the next subheading.

The corresponding section is nested in a special DIV container with the `class="floatbox"`. Within this DIV container, objects can be placed at will with `float:left` or `float:right`:

```
...
<h2>Subheading 1</h2>
<div class="floatbox">
  <img src="picture.jpg" style="float:left;" alt="" />
  <p> ... a paragraph ...</p>
  <p> ... a second paragraph ...</p>
  <p> ... and another paragraph in the flow of text. </p>
</div>

<h2>Subheading 2</h2>
...
```

The flow of text is restricted to the DIV container by the nesting, and needs no extra HTML code with `clear:both;` .

2.7 The Clearing of #col3

The previous section explained the global behavior of `clear:both;` and its effects within the static container `#col3`. Though this effect would be counterproductive for the position of content within `#col3`, YAML specifically exploits this effect to consistently make `#col3` the longest column in the layout -- independent of the amount of content in the other columns.

The goal of these efforts to use the CSS `border` property of `#col3` to create vertical column separators (solid, dashed, or dotted lines) or even solid color column backgrounds for the *float* columns without using graphics. Because of the global clearing, these will always reach to the `#footer`. This profides an alternative method of designing the graphic layout, which is also extremely easy to edit.

2.7.1 Global Clearing Makes #col3 the Longest Column

How does `#col3` become the longest column? In all modern browsers (Mozilla, Firefox, Opera etc.), this happens without any further ado. As `#col3` is a static container, the clearing of `#col3_content` via the `clearfix` class works globally and forces `#col3` to stretch to the lowest end of the longest *float* column. More on the functioning of the `clearfix` class in Section 2.6: How Floats Work.

2.7.2 Special Clearing Solution for Internet Explorer

The global clearing via clearfix does not work in IE, as it does not recognize the CSS pseudo-class `:after`, which contains the property `clear:both;` . Additional HTML must be added to the end of `#col3` to contain it again: this is done with an invisible `DIV`.

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

Let us take a close look at this *invisible* DIV. As mentioned, this container is only required for IE. For modern browsers, it is turned off completely. The necessary declarations are in the file *base.css* in the folder *yaml/core/*:

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

The adjustment in the properties of this particular clearing DIV for IE are in the file *iehacks.css* in the *yaml/core/* folder:

```
#ie_clearing {
    display:block; /* DIV made visible */
    \clear:both; /* Normal clearing for IE5.x/Win */
```

```
    width: 100%; /* IE Clearing with 100% DIV for IE 6 */
    font-size:0;
    margin: -2px 0 -1em 1px; /* IE clearing with extra-large DIV for IE7 */
}

* html { margin: -2px 0 -1em 0; }
#col3_content { margin-bottom: -2px; }

/* (en) avoid horizontal scrollbars in IE7 ... */
html { margin-right: 1px; }
* html { margin-right: 0; }

/* (en) Bugfix: Essential for IE7 */
#col3 { position:relative; }
```

IE Clearing in Internet Explorer 5.x

`display:block` turns the DIV on. Then the actual clearing begins: with `\clear:both`. The leading backslash exploits the IE 5.x and 6.0 parser bug, which ensures that the property will only be understood by Internet Explorer 5.x.

Important: this is the standard method for clearing float environments. Unfortunately a particularly tricky bug turns up in Internet Explorer v5.x to v7, which under certain circumstances can lead to the collapsing of the left margin of `#col3`. More information on this in Section 5.3: Known Problems - Internet Explorer. This bug cannot be fixed in IE 5.x, so the regular clearing is still used for this browser version.

For Internet Explorer 6 and 7, we use a special clearing method, which prevents the bug from appearing.

IE Clearing in Internet Explorer 6.0

The clearing solution bases on the fact that within `#col3`, Internet Explorer will break too-large elements beneath the *float* columns. The DIV container `#ie_clearing` is defined with `width: 100%` in IE6 to force this. As the *float* columns will still restrict the space remaining to less than 100 percent, the container must break under the *float* columns.

IE Clearing in Internet Explorer 7.0

IE7 needs a box with a width of over 100 percent. The container, therefore, also needs an additional left margin of 1 pixel (`margin: -2px 0 -1em` **1px**). But Internet Explorer 7 has a bug that makes this overlapping pixel - which has no significance for the layout - to cause horizontal scrollbars when used with whole-page layouts (`body`, `.page_margins` and `.page` at 100% width and no border). To catch this case, the HTML element `html` receives a 1 pixel wide margin on the right side.

```
/* Avoiding horizontal scrollbars for layouts with too-large content in
IE7*/
html {margin-right: 1px}
* html {margin-right: 0}
```

This trick prevents horizontal scrollbars and the extra 1 pixel wide edge next to the vertical scrollbar in IE7 is usually not even noticed.

Now one more helpful hint for Internet Explorer 7. The container `#col3` has to be assigned the property `position:relative`. Without it, Internet Explorer 7 would ignore the container `#ie_clearing`.

Hiding Clearing Containers in Layouts

The margins in the other directions `margin:` **-2px** `0` **-1em** `1px` are only to make the container optically invisible in all IE versions. To make it definitively invisible, the font size was set at 0. The height of the container then shrinks to 2 pixels. These last two pixels are then canceled out by a further negative margin in `#col3_content`. Now the DIV container is not visible in the layout, and yet still fulfils its functions.

One last adjustment is necessary. The IE clearing only works as long as the column `#col3` is **not** given the proprietary property hasLayout. bekommt. Yet exactly that is called for when, for example, removing the 3 pixel bug (see Section 3.5: CSS Adjustments for Internet Explorer). In this case, the column dividers cannot be used. Still, the columns must be cleared correctly, in order to place the footer beneath them. This is done easily by also giving the container `#footer` in the file *base.css* the property `clear:both;`.

2.7.3 Graphic-Free Column Divider

Done: now we can use the CSS property `border` of `#col3` for vertical column dividers and / or solid color column backgrounds for `#col1` and `#col2`, which go all the way down to the footer. All without a single graphic. As an example, we can construct a vertical dotted line:

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

You want proof that it works? Here you go:

/examples/04_layouts_styling/3col_column_dividers.html

Detailed descriptions of examples of these techniques are in Section 4.2: Designing the Columns.

> **Note:** the use of this technique is only recommended in combination with a column setup with `#col3` in the middle, i.e. 1-3-2 and 2-3-1 or when using two column layouts. More information on variable column order is found in Section 4.4: Variable Column Order.
>
> When using the column order 1-2-3 / 3-2-1 or 2-1-3 / 3-1-2, this technique is not so useful, as IE will not stretch `#col3` to the height of the longest *float* column. With these layout variants, please use the "Faux Columns" technique for defining vertical column dividers.

This closes the explanation of the structure of YAML's XHTML source code and the functions of the IE clearings. The foundation is set. The last bit of the source code structure is the skip links, which are explained in the following section.

2.8 Skip Link Navigation

Skip-links improve the usability of a website most of all for those users who are dependent upon a screen reader. Screen readers linearize the content of a website and read it aloud from beginning to end. Skip-links should be as close to the beginning of the source code as possible and provide links to the most important areas within the web page (navigation, content, etc.).

This of course invites the discussion of wheter it is not better to simply place the content of the website as close to the beginning of the source code as possible -- and place the navigation further down. This would let the user arrive at the content more quickly, without having to listen to the navigation links be read aloud on every single page.

But - what if the user does not want to read the content? The user might well merely want to visit a further subarea of the navigation. It would then be quite frustrating to have to go through the entire content before getting to the navigation. Clearly, there is no perfect placement of the content in the source code. More practically, we need to help the users get quickly to the kind of content they need. Skip-links are a very simple and effective tool.

2.8.1 Skip Link Navigation in the YAML Framework

The skip links in YAML's source code are located just after the opening BODY tag. In general, they should be placed before all other content. Two skip targets are defined: The first skiplink passes further content in `#topnav` and `#header` and leads directly to the main navigation in `#nav`. The second skip link leads directly to the main content of the website, in this case to `#col3`.

Here is the associated markup. Skip-links are created as an unordered list with the ID `#skiplinks`. This class is defined in the *base.css* (see Section 3.3) and controls the visual properties of the list. Furthermore, each link in the list get's the CSS class `.skip`.

```
<!-- skip link navigation -->
<ul id="skiplinks">
  <li><a class="skip" href="#nav">Skip to navigation (Press
Enter).</a></li>
  <li><a class="skip" href="#col3">Skip to main content (Press
Enter).</a></li>
</ul>
```

Alternatively, you can integrate the skip links directly into their layout. The unordered list is optional. To ensure the functionality, only the CSS class `.skip` is required for skip anchors. The following example shows the placement of the skip links before the meta navigation in `#topnav`.

```
<div id="header">
  <div id="topnav">
    <a class="skip" href="#nav">Skip to navigation (Press Enter).</a>
    <a class="skip" href="#col3">Skip to content (Press Enter).</a>
    ...
  </div>
  ...
</div>
```

Existing IDs within the layout serve as skip targets in both of these examples. You can use any ID as an anchor in the href attribute of your skip links. Alternatively - although not often done today - you can use named anchors as skip targets. In this case, the skip target must have identical values for both the `id` and `name` attributes to ensure backwards compatibility (i.e. `<a id="content" name="content">`).

2.8.2 Invisible and Accessible

Users without disabilities who use the internet with standard browsers generally do not need this navigational help. For this reason, the skip links are invisible in the print and normal screen views.

However, the links cannot be hidden with the css property `display:none;` that is interpreted by screen readers and such content is not read aloud. The skip links would then be unusable. Furthermore, skip links have to become visible on focus to provide visual feedback for those users navigating with a keyboard.

The CSS definitions required are located in the CSS file *base.css* (see section 3.3: Base Stylesheet) and are always available via the ID `#skiplinks` (invisible list) and `.skip` (skip anchor).

```
/**
 * @section hidden elements | Versteckte Elemente
 * @see     www.yaml.de/en/documentation/basics/skip-links.html
 *
 * (en) skip links and hidden content
 */

/* (en) classes for invisible elements in the base layout */
.skip, .hideme, .print {
  position: absolute;
  top: -32768px;
  left: -32768px; /* LTR */
}

/* (en) make skip links visible when using tab navigation */
.skip:focus,
.skip:active {
  position: static;
  top: 0;
  left: 0;
}

/* skiplinks: technical setup */
#skiplinks {
  position: absolute;
  top: 0px;
  left: -32768px;
  z-index: 1000;
  width:100%;
  margin: 0;
  padding: 0;
  list-style-type: none;
}

  #skiplinks a.skip:focus,
```

```
#skiplinks a.skip:active {
  left: 32768px;
  outline: 0 none;
  position: absolute;
  width:100%;
}
```

After the general definition, the visual presentation of the links must be determined by the webdesigner. The layout examples included are shaped by the rules in the *basemod.css*.

```
/**
 * Skiplinks
 *
 * (en) Visual styling for skiplink navigation
 * (de) Visuelle Gestaltung der Skiplink-Navigation
 *
 * @section content-skiplinks
 */

#skiplinks a.skip:focus,
#skiplinks a.skip:active {
  color:#fff;
  background:#333;
  border-bottom:1px #000 solid;
  padding:10px 0;
  text-decoration:none;
}
```

Further reading: Jim Thatcher's "[Skip Navigation](#)" article gives a very good overview of various methods of skip link presentation.

This helpful navigational method can be expanded through the use of further skip targets. Such expansion is entirely up to the individual webdesigner and should be undertaken only after thorough consideration.

2.8.3 Correcting Focus Problems

Although there are no problems with the visual presentation of skip links, Internet Explorer 8 on Windows 7 as well as Webkit-based browsers such as Safari and Google Chrome require a focus bugfix. In these browsers, the skip target is reached (the target is displayed in the visual area of the viewport), but the tab focus remains on the skip link and **not** on the target. If the user tabs again, the focus jumps back to the beginning of the document, to the anchor or element which immediately follows the activated skip link.

YAML provides a script which finds all the skip links in a document, analyzes the targets, and automatically sets the focus to the target anytime one of the links is clicked.

The JavaScript file *yaml-focusfix.js* is in the folder */yaml/core/js/*.

```
<!-- full skip link functionality in ie8 & webkit browsers -->
<script src="./yaml/core/js/yaml-focusfix.js"
type="text/javascript"></script>
</body>
```

To prevent possible performance problems in high-traffic sites, the script is included just before the closing BODY tag in the layout samples and in the file *markup_draft.html*.

3 CSS Components

3.1 The CSS Concept

YAML's CSS concept is modular and cascading. The CSS definitions of the basic layout are divided according to function into several separate CSS components (files):

- Positioning of the main areas of the web page (header, footer, columns)
- Screen layout: design of the main areas
- Formatting of the content
- Design of the navigational elements
- Print templates

The finished layout always comprises several of these components. The separation according to function makes editing and organizing easier.

Furthermore, regular CSS is strictly separated from those files necessary for Internet Explorer hacks (bugfixes for CSS bugs). Many of these bugfixes exploit other IE parser bugs, which let IE accept invalid or incorrect CSS declarations.

Only in rare cases can regular CSS be mixed with the IE bugfixes and still validate. The hacks also interfere with the legibility of the stylesheets. A summary of these hacks in one single file allows better comprehension regarding the various IE browser versions, which themselves sometimes need varying hacks.

3.1.1 Cascading

In addition to the thematic organization of the style listings in various CSS components, YAML uses the cascading of CSS quite intensively.

Cascading lets the browser decide, which CSS properties are relevant for the display of any particular element. This cascade is divided into four steps:

- Step 1: origin of the declarations
  (browser, author, or user stylesheet).
- Step 2: sorting by origin and weight
- Step 3: sorting by selector specificity
- Step 4: sorting by order of appearance

With the CSS basic components (*base.css* and *iehacks.css*), the page creator is presented with a three-column basic layout as a basis. These stylesheets are integrated into each YAML-based layout and are never changed.

The basic layout can then be modified by overwriting specific style declarations and expanding other properties. All the page creator's changes should be made in separate stylesheets: only then can YAML remain the stable basis at the lowest level.

3.2 Naming Conventions

Certain terms are used again and again within the documentation, as well as in the naming of files and folders of the framework. A short definition of these:

3.2.1 Basic components (core files)

The *core files* comprise the core or the foundation of the YAML framework and are in the folder *yaml/core/*.

They provide the basic functionality of the framework and are necessary for the cross-browser uniform layout presentation. They are necessary for every YAML-based layout.

3.2.2 Complementary components

YAML is based on the cascade principle. The actual layout design is created by *modifying* YAML's basic layout. In addition, YAML provides several more finished CSS components as well as templates for often-required elements. These modules are organized by function:

- Screen layout - *screen/*
- Print layout - *print/*
- Navigation - *navigation/*

Should these files be used unchanged, they need only be copied directly into the layout from the folder *yaml/*. Separate, new stylesheets or modification of these components should be maintained in a new *css* folder.

3.2.3 Patches

A *patch* file contains all the necessary CSS adjustments for Internet Explorer together in one CSS file. This is integrated into the (X)HTML source code with a *conditional comment* and ensures a homogeneous layout.

3.2.4 File templates

YAML offers file templates for oft-required components. These templates have names ending in **_draft.**.

To use them with YAML, copy those you need into your *css* folder and rename them.

3.3 The Central Stylesheet

YAML's CSS concept is based, as discussed previously, on modules as well as on the cascade principle. The CSS components are composed according to function (positions of the layout elements, formatting of content, etc.)

The following diagram shows the functions and meaning of the YAML framework's individual components.
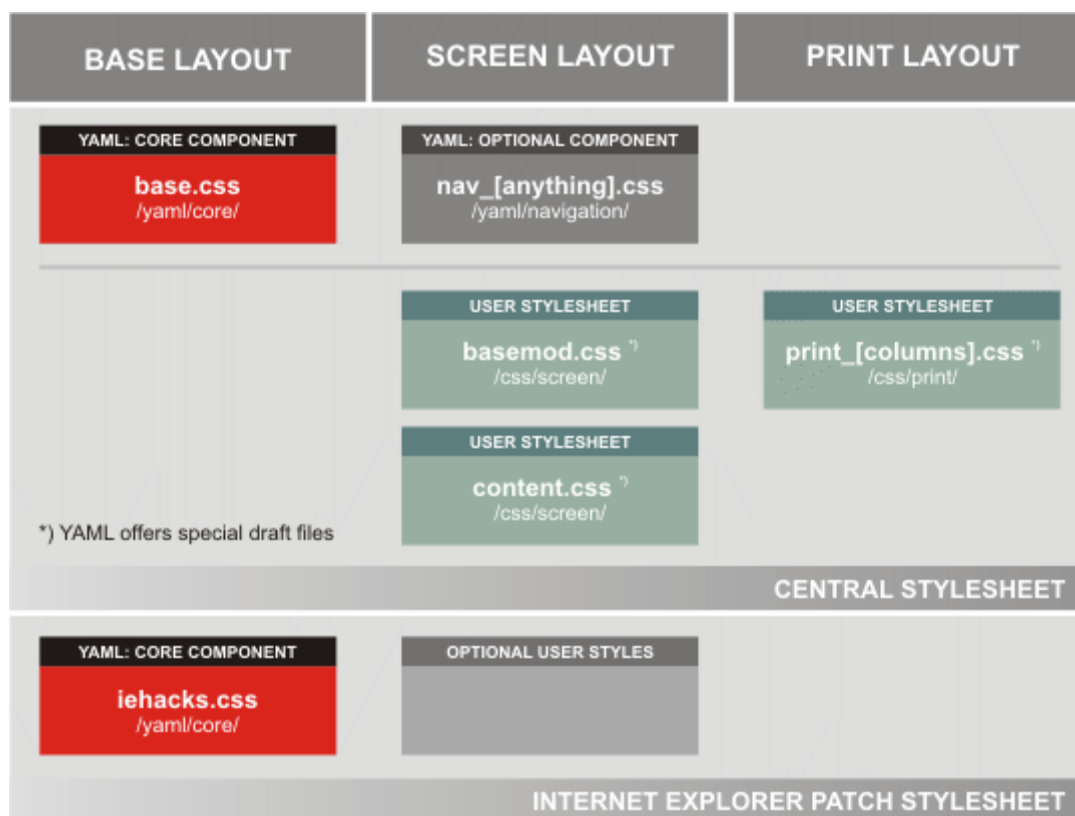
Every YAML-based layout contains such a central stylesheet, which integrates all the required components for that layout (basic components, screen layout, navigation, print styles). A complete layout always comprises several of these components. This separation according to function makes editing and comprehension easier.

How do we start actually working with YAML?

3.3.1 Integration & Import of the CSS Components

The structure of the central stylesheet -- and thus the use of YAML in your own projects -- is easiest explained through examples.

> **Note:** copy the folder *yaml* from the download package onto your server, on the same level as your *css* folder. This separation between your own css files and the files of the framework is necessary to let you update YAML at any time.

The layout is integrated into the (X)HTML source code via a so-called *central stylesheet*, which is usually reached with the `link` element in the HTML head of each web page.

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

This *central stylesheet* contains your layout and should be placed in your css folder. Within this stylesheet, the other necessary CSS components are integrated with the `@import` rule.

```
/* 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);
```

As you can see, the first stylesheet is the most basic of the YAML framework: *base.css*. It is loaded directly from the *yaml/core/* folder.

In the second step, the screen layout is put together. A stylesheet for the navigation is loaded: *nav_shinybuttons.css*. This should remain unchanged, so again, the link is directly to the *yaml* folder. The screen layout and the content design is up to you: those files should be saved to your own *css* folder.

The third and last step connects the print layout, also available as YAML prefabricated components. In this example, one of these files (*print_003_draft.css*) is directly linked from the *yaml/print/* folder, without customization.

Important: the basic principle of separation of your custom CSS files from the YAML files has many advantages, borne out by practical use.

If you want to make changes to any files of the framework or use any of the file templates, copy the file to your *css* folder and do not work with the original.

Unchanged original components should be imported directly from the *yaml* folder into your layout. When updating the framework, you need then only overwrite the *yaml* folder.

3.3.2 Adjustments for Internet Explorer

All modern browsers (Firefox, Safari, Opera, etc.) have their CSS needs met with the *central stylesheet* linked from the (X)HTML source code. Only Internet Explorer needs extra CSS adjustments to be able to display YAML-based CSS layouts. These are integrated into the framework with a so-called *conditional comment*.

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

This is a special comment, which is only understood and interpreted by Internet Explorer. It allows IE to access a specially created stylesheet which no other browser will see. In the example above, this is the file *patch_3col_standard.css*, which contains all CSS modifications for IE.

More on these functions in Section 3.5: CSS Adjustments for Internet Explorer. For all other browsers, this is a normal HTML comment, and they ignore its content.

3.4 The Base Stylesheet *base.css*

Important: The stylesheet base.css in the folder yaml/core/ is one of the basic components of the YAML framework. It sets up the foundation (browser reset, clearing, subtemplates etc.). This stylesheet is required for every YAML-based layout and should not be changed lightly!

3.4.1 Browser Reset - Uniform Starting Point for All Browsers

Note: the components of the browser reset are required for all media. The following CSS rules are thus included with the media rule @media all.

YAML's purpose is to guarantee a uniform and cross-browser compatible layout. A uniform starting point is necessary. This is not a given: every browser sets its own particular standard formats for displaying unformatted content.

Let us examine the first lines of the base stylesheet *base.css*:

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

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

* html body * { overflow:visible }

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

div { outline: 0 none; }

article,aside,canvas,details,figcaption,figure,
footer,header,hgroup,menu,nav,section,summary {
    display:block;
}

fieldset, img { border:0 solid; }

ul, ol, dl { margin: 0 0 1em 1em; }
li {
  margin-left: 0.8em;
  line-height: 1.5em;
}

dt { font-weight: bold; }
dd { margin: 0 0 1em 0.8em; }

blockquote { margin: 0 0 1em 0.8em; }
```

```
blockquote:before, blockquote:after,
q:before, q:after { content: ""; }
```

Eliminating margins and paddings

Setting `* { margin:0; padding:0; }` eliminates the inner and outer spacing of all HTML elements via the asterisk selector. This method takes care of all HTML elements at one go.

For `select` elements, however, this creates a small problem. The above instruction of course sets the padding of the `option` element (the choices in the selectbox) to zero, causing it (in Windows) to hide the last letter of the content. This problem is solved by setting its standard value explicitly: `option {padding-left: 0.4em}`.

Note: the entry can be completed with the CSS property `* {border: 0;}`. However, this also removes the preformatting of form elements -- textareas and submit buttons.

In this case, these elements must be formatted with the standard values in the CSS file *content.css* (see [Section 3.8: Designing the Content](#)), or they will be quite difficult to see on the screen.

To avoid a permanently visible outline of DIV elements div {outline: 0 none;}. This occurs when its ID's are used as targets for skip links. The rule doesn't affect other browsers due to their identical default behavior.

The border for the HTML elements `fieldset` and `img` are also set to zero (`fieldset, img { border:0 solid; }`).

Avoiding the italics bug in IE

This bugfix for Internet Explorer 5.x and 6.0 is an exception. While all YAML's further CSS hacks for IE are collected in special stylesheets, this bugfix must appear before all layout-specific CSS declarations to work properly.

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

An exact description of this bugfix is in [Section 3.5: CSS Adjustments for Internet Explorer](#).

Font size and rounding errors

The declaration `body { font-size: 100.01% }` compensates rounding errors, in particular in older versions of Opera and Safari. Both would otherwise display fonts that are too small.

Note: in earlier YAML versions, this font size correction was also included for the elements `select`, `input`, and `textarea` for Safari 1.x. This led to problems in the current Firefox 2.x and will not be used after YAML version 3.0. Safari 1.x is also seldom used today.

Block Formatting Context für HTML5 Elemente

With the upcoming HTML5 standard several new elements with semantic background are introduced, which can already be used in the current browser generation (except Internet Explorer up to version 8). Unfortunately, until today the browser manufacturers didn't provide a standard

formatting as inline or block elements for these elements. This standardization is therefore made at here as block level elements:

```
article,aside,canvas,details,figcaption,figure,
footer,header,hgroup,menu,nav,section,summary {
    display:block;
}
```

Standard values for lists and quotations

HTML lists as well as elements for designating quotations (`blockquote, ol, ul, dl`) need line heights and margins in order to be consistent in all browsers. The browser's own interpretations were overwritten with the declaration `* {margin:0; padding:0 }` along with the other browser resetting options.

```
ul, ol, dl { margin: 0 0 1em 1em; }
li {
  margin-left: 0.8em;
  line-height: 1.5em;
}

dt { font-weight: bold; }
dd { margin: 0 0 1em 0.8em; }

blockquote { margin: 0 0 1em 0.8em; }

blockquote:before, blockquote:after,
q:before, q:after { content: ""; }
```

3.4.2 Standard CSS Classes

Note: the components of the standard classes are necessary for all media. The following CSS rules are thus included with the media rule `@media all`.

YAML provides many standardized CSS classes to be used when required. These classes are discussed briefly.

Clearing Methods

YAML provides two general methods for clearing floats without additional markup. These are the clearfix and the overflow methods. Their technical details were discussed in section 2.3. Practical use scenarios for each method were described in section 2.6.

```
/**
 * @section clearing methods
 * @see    yaml.de/en/documentation/basics/general.html
 */

/* (en) clearfix method for clearing floats */
/* (de) Clearfix-Methode zum Clearen der Float-Umgebungen */
.clearfix:after {
  clear: both;
  content: ".";
  display: block;
  font-size: 0;
```

```
  height: 0;
  visibility: hidden;
}

/* (en) essential for Safari browser !! */
/* (de) Diese Angabe benötigt der Safari-Browser zwingend !! */
.clearfix { display: block; }

/* (en) overflow method for clearing floats */
/* (de) Overflow-Methode zum Clearen der Float-Umgebungen */
.floatbox { overflow: hidden; }
```

Section 2.7 described how within YAML the container `#col3` can consistently remain the longest column by using the clearfix method. Internet Explorer requires many adjustments for this function, which are provided by the container `#ie_clearing`. The adjustments for IE 5.x – IE 7.0 are included via the *iehacks.css*. The container `#ie_clearing` is generally hidden in modern browsers, as they do not require it.

```
/* (en) IE-Clearing: Only used in Internet Explorer ... */

#ie_clearing { display: none; }
```

Skip Links and Invisible Content
In order to provide the most easily accessible webpages, YAML offers predefined CSS classes in the `base.css` to hide content from the visual screen and yet make it available to print versions and alternative media such as screen readers.

```
/**
 * @section hidden elements | Versteckte Elemente
 * @see      www.yaml.de/en/documentation/basics/skip-links.html
 *
 * (en) skip links and invisible content
 */

/* (en) classes for invisible elements in the base layout */
.skip, .hideme, .print {
  position: absolute;
  top: -32768px;
  left: -32768px; /* LTR */
}

/* (en) make skip links visible when using tab navigation */
.skip:focus, .skip:active {
  position: static;
  top: 0;
  left: 0;
}

/* skiplinks: technical setup */
#skiplinks {
  position: absolute;
  top: 0px;
  left: -32768px;
  z-index: 1000;
  width:100%;
  margin: 0;
  padding: 0;
```

```
    list-style-type: none;
}

#skiplinks a.skip:focus,
#skiplinks a.skip:active {
  left: 32768px;
  outline: 0 none;
  position: absolute;
  width:100%;
}
```

The definition of the CSS class `.skip` for providing predefined skip links was explained in [section 2.8: Skip-Link Navigation](). Further standard classes were developed to display content only to particular media (for example, to provide additional information for screen readers).

The class `.hideme` hides content from all visual media. The CSS class `.print` allows content to be hidden from screens and yet printed onto paper. Both classes are fully accessible to screen readers.

3.4.3 Building Blocks for the Screenlayout

YAML works on the top-down principle, providing a basic markup with often-used standard elements of common layouts.

- `.page_margins` serves as a *wrapper* for a particular layout area. Usually the layout width or the minimum and maximum widths of a flexible layout are set via `.page_margins`.
- `.page` defines the borders of and space around flexible layouts in relation to .page_margins (see the combination model, [Section 2.4]()).
- `#header`, `#nav`, `#main` und `#footer` are typical standard elements for all layouts.
- In addition to the main navigation, many pages include a "metanavigation", which contains links to the sitemap, the privacy policy, etc. The container `#topnav` is intended for this.
- Within `#main` there are three column containers:
  - `#col1` & `#col1_content`
  - `#col2` & `#col2_content`
  - `#col3` & `#col3_content` + `#ie_clearing`
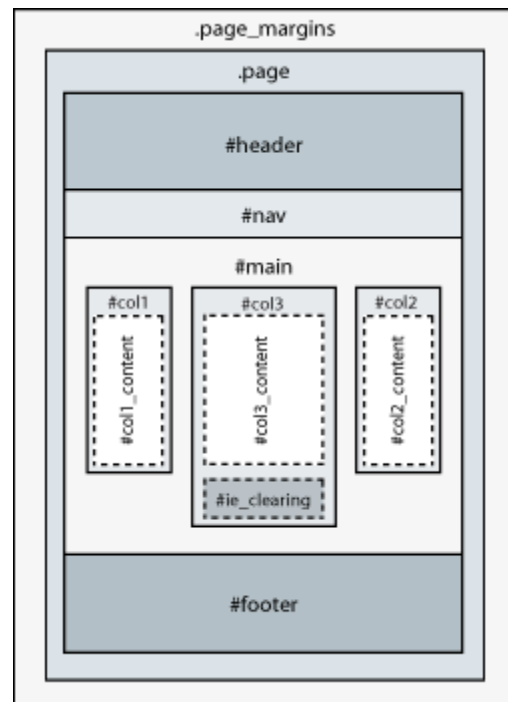
  The column containers are each divided into two, in order to allow for the greatest possible flexibility and freedom in designing flexible layouts with the combination model.

Basic Layout - Universal Fallback
The predefined elements in the markup can always be freely redesigned, but basic values are set within the *base.css* to provide a fallback to a simple three-column layout. In this basic layout, the side columns `#col1` and `#col2` are each defined as 20% wide.

The `position:relative` property for the content container `#colx_content` is a preparatory step to allow absolute positioning within the columns. It also ensures that Internet Explorer's column contents appear without selection or changing the size of the window, and they are prevented from suddenly disappearing.

```
/* (en) Ensuring correct positioning */
#header, #nav, #main, #footer { clear:both; }

#col1 { float: left; width: 20%; }
#col2 { float:right; width: 20%; }
#col3 { width:auto; margin: 0 20%; }

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

This fallback means that you never face designing your new screen layout with a completely blank slate: even in the worst case scenario, you can always call up a fully-functioning basic layout.

Flexible Grid Blocks

The *base.css* also contains the class definitions for the flexible grid blocks, YAML's subtemplates. The functions and structure of the subtemplates is covered thoroughly in Section 4.5.

3.4.4 Specifications for the Print Version

Note: the following CSS rules for the print version are grouped within the media rule `@media print`.

Float Clearing in the Print Version

Both Firefox and Internet Explorer still have some problems in their current versions with accurate printing of floated elements, or when using the `overflow:auto` or `overflow:hidden` properties. The `display:table` property avoids these kind of problems and ensures that floating content remains within the surrounding parent containers.

```
/**
 * (en) float clearing for subtemplates.
 */

.subcolumns,
.subcolumns > div {
  display: table;
  overflow:visible;
}
```

In addition to `.subcolumns`, these properties are also applied to the selector `.subcolumns > div`. This is a preventive measure to ensure that the content of the subtemplates remains within its surrounding column container `.cxxl` or `.cxxr`.

Adjusting the print version for specific content

The CSS class `.print` is defined along with the skip link classes (see Section 2.8) in order to hide elements on screen while leaving them accessible for screen readers and in the print version. The following rule makes elements with the `.print` class visible.

```
/* (en) make .print class visible */
/* (de) .print-Klasse sichtbar schalten */
.print { position: static; left: 0; }
```

In general, you should hide elements from your print version with your print stylesheet and CSS. Yet there is no rule without exceptions, and not every element in a complex, dynamic layout can be reliably reached with CSS selectors. For such cases, we have the CSS class `.noprint`. This class hides content from the print version, but must be specifically assigned to such content in the markup.

```
/* (en) generic class to hide elements for print */
/* (de) Allgemeine CSS Klasse, um Elemente in der Druckausgabe auszublenden */
.noprint { display:none !important; }
```

3.5 CSS Adjustments for Internet Explorer

Internet Explorer since Version 5 has integrated broad support for CSS 1 and good support for CSS 2. Unfortunately the CSS 2 support in particular is riddled with mistakes, which ignored quickly lead to display errors in CSS layouts.

The source code structure of the YAML basic layout is set up to allow many variations via CSS without changing the HTML code. To ensure this flexibility, we must iron out the numerous IE CSS bugs.

The CSS bugs in IE occur in connection with specific source code constructs relating to combinations of floating, positioned, and static elements. As the YAML framework's code is fixed and its variations are known, most of the bugs are predictable and thus manageable.

The bugs are categorized according to their manifestation, and are dealt with separately:

Structure- and layout-independent adjustments

> Most of the CSS bugs are easily managed from within the XHTML source code. When such bugfixes are compatible with all possible modifications and column orders, it qualifies as *structure- or layout-independent*.

> All these are managed in one stylesheet, *iehacks.css* in the folder *yaml/core/*, which should not be modified.

Layout dependend adjustments

> Some CSS bugs are only triggered by particular layouts. These problems cannot be dealt with by the standard structure, but must be handled individually by the site's designer and are categorized as *structure- or layout-dependent* - especially as they are often triggered by the display of particular content elements.

> Every YAML-based layout should include a hack file *patch_[layout].css* for Internet Explorer, replacing the placeholder *[layout]* in the filename to match the relevant central stylesheet. A template for such a hack stylesheet (*patch_layout_draft.css*) is in the *yaml/patches/* folder.

Structure of the CSS Patch File for Internet Explorer

As described above, every YAML-based layout (or every central stylesheet, see Section 3.3) requires an IE patch file *patch_[layout].css*. The structure of such stylesheets is described below, using the example of the template file *patch_layout_draft.css* from the *yaml/patches/* folder.

```
/* Layout independent adjustments ----------- */
@import url(/yaml/core/iehacks.css);

/* Layout dependent adjustments ------------ */
@media screen, projection
{
   /* add your adjustments here | Fügen Sie Ihre Anpassungen hier ein */

   ...

}
```

As you can see, this file includes both layout-dependent and -independent adjustments. You need then only integrate one additional CSS file into your layout.

The first section imports the file *iehacks.css* from the *core/* folder of the YAML framework. As previously mentioned, this file contains all the structure- and layout-independent bugfixes and can thus be integrated unchanged into every YAML-based layout.

The second part contains an empty `@media` rule. After this you can integrate further IE stylesheets (the navigation component *nav_vlist*, for example). Furthermore, this is the place to add the *structure- or layout-dependent* bugfixes or bugfixes for the correct display of layout elements.

This IE adjustment stylesheet then takes care of similar issues as the central stylesheet: all CSS hacks are collected and presented to Internet Explorer.

Integration of the CSS Adjustments in YAML's Layout

Many bugfixes exploit IE's numerous parser bugs - particularly those in older IE versions. The resulting CSS code is therefore not always valid and should thus only be made accessible to IE. This is possible with the use of conditional comments within the HTML head `<head>..</head>`. This was already mentioned at the end of Section 3.3: The Central Stylesheet.

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

The condition `lte IE 7` means "lower than or equal to Internet Explorer Version 7.0". This special comment is only recognized and interpreted by IE - for all other browsers, it is a normal comment, and they ignore its contents.

In the following, all layout-relevant IE CSS bugs will be explained and their YAML framework fixes / workarounds described.

3.5.1 Structure- and Layout-Independent Bugfixes

All *structure- and layout-independent* bugfixes for IE's CSS bugs are collected in the file *iehacks.css* in the *yaml/core/* folder.

Important: the stylesheet *iehacks.css* from the *yaml/core/* folder is one of the core components of the YAML framework. It contains all the structure- and layout-independent bugfixes for IE (versions 5.x/Win - 7.0/Win). These corrections are essential for the strength and error-free presentation of YAML-based layouts in Internet Explorer. This stylesheet is required in every YAML-based layout and should remain unchanged!

Fundamental CSS Adjustments

YAML recommends forcing modern browsers (IE8, Firefox, Safari, ect.) to always display vertical scrollbars via CSS3 property `overflow-y` (see Section 3.6). In older Internet-Explorer versions, this workaround are not required, as the scrollbars are always displayed.

```
/**
 * (en) No need to force scrollbars in older IE's ...
 *
 * @workaround
 * @affected   IE6, IE7
 * @css-for    IE6, IE7
 * @valid      no
 */

body { o\verflow:visible; }
```

The next declaration is important for Internet Explorer 7, which has problems when zooming in on YAML-based layouts.

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

The relative positioning of the `body` solves nearly all IE 7's zoom problems. The container `#main` also gets this property. This avoids wrong positioning of columns after resizing the browser window while using IE-Expressions.

Adjusting Clearing Methods for IE

The CSS adjustments for the clearfix clearing are based on work done by [Roger Johansson](#) and are already compatible with IE 7.

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

Increasing the Reliability of the Layout

Numerous IE CSS bugs can be resolved by activating the proprietary property *hasLayout*. For some of the predefined containers in the source code structure, this bugfix can be used without there being a real need for it - purely as a precautionary measure.

```
body { height:1%; }

/* IE6 & IE7 */
.page_margins, .page, #header, #nav, #main, #footer { zoom:1; }

/* IE 5.x & IE6 | IE6 only */
* html .page_margins, * html .page { height:1%; hei\ght:auto; }


/* IE 5.x & IE6 | IE6 only */
* html #header, * html #nav,
* html #main, * html #footer { width:100%; wid\th:auto; }
```

The two containers which contain the layout (`.page_margins` and `.page`) are given *hasLayout* via the property `zoom:1` (IE6 & 7) or `height: 1%` (IE 5.x). The property `width` was intentionally left out here: as the file *iehacks.css* is the last to be imported into the browser, the designer's intentions could be unintentionally overwritten.

In the inner containers, the use of `height` is problematic, in case containers with fixed heights should be intended. To retain flexibility, the proprietary property `zoom` s used for IE 6. The use of `zoom:1` has no disturbing side effects. For IE 5.x, the box model bug is exploited, allowing the unproblematic use of `width: 100%`. IE 5.0 does not recognize the property `zoom`, thus requiring this additional declaration.

> **Important:** The rules for the former's ID `#page_margins` and `#page` still included for backward compatibility, but they are no longer recommended. Use the corresponding CSS classes in new projects.

Avoiding an Incomplete Display of Column Content

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

A further workaround helps to avoid display problems in older versions of IE. IE 5.x and IE 6.0 sometimes display content only partially or not at all. The relative positioning of the column containers solves this problem.

After these general preventative measures, the following details the handling of the most important known CSS bugs and their treatment.

Escaping Floats Bug

	IE 5.x/Win	IE 6.0	IE 7.0
Bug active	Yes	Yes	Yes

The Escaping Floats Bug causes Internet Explorer to position *floats* incorrectly within a DIV container. Two problems appear. First, the size of the surrounding DIV container is incorrectly calculated, and second, the *floats* float out of the right-hand side of the container.

Both problems can be solved with the activation of *hasLayout* - in our example, with `height:1%`. This bugfix has already been integrated within the basic layout in the section "Increasing the Stability of the Layout": further measures are not required.

Guillotine Bug

	IE 5.x/Win	IE 6.0	IE 7.0
Bug active	Yes	Yes	Yes*

The IE/Win Guillotine Bug is triggered by many actions, in particular hover effects on hyperlinks. This is absolutely the best-known IE bug -- and unfortunately, the most reliable way to avoid it is by: avoiding hover effects.

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

IE7 should have repaired this bug, yet reports of collapsing spacing still come in. The bugfix is therefore also set to be used by IE 7.

Double Float-Margin Bug

	IE 5.x/Win	IE 6.0	IE 7.0
Bug active	Yes	Yes	No

Internet Explorer doubles the values of the side margins when positioning floated containers: the ("Doubled Float-Margin Bug") creates layout problems for the variable order of content columns.

Bugfix: Happily, the bug is easy to fix. Both *float* columns #col1 and #col2 are given the property display:inline: ignored by all modern browsers, this guarantees that Internet Explorer 5.x and 6.0 display the margins correctly.

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

Expanding Boxes in Internet Explorer

	IE 5.x/Win	IE 6.0	IE 7.0
Bug active	Yes	Yes	No

Internet Explorer has great difficulty dealing with oversized content within fixed-width boxes. See Internet Explorer and the Expanding Box Problem.

Bugfix: force a special line break-mode to guarantee a clean display in IE:

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

The property word-wrap: break-word is proprietary to Internet Explorer and incomprehensible to other browsers. It allows the browser to break text not only between words, but after *every* letter. This does reduce readability somewhat when used in a very narrow column, but does provide a consistent layout. The older 5.x versions of IE unfortunately do not react to this hack.

Oversized content elements can only be dealt with on the layout level: suggestions below.

Internet Explorer and the Italics Problem

	IE 5.x/Win	IE 6.0	IE 7.0
Bug active	Yes	Yes	No

The IE Italics-Bug is one of the most difficult to recognize and probably one of the least known CSS bugs. IE expands the width of a container as soon as content in italics (marked with <i> or <em>) touch the right edge of the line. The CSS property font-style: italics can also trigger the bug.

The resulting greater width of the parent container creates problems in *float*-based layouts, as the container no longer fits in the layout. The problem mostly affects the static column #col3. Combined with the lack of *hasLayout*, static containers can even be completely hidden.

Bugfix: the fix for this problem is quite simple: the CSS property `overflow:visible;` is merely assigned to all elements of the web page. This property's position in *base.css*, the first stylesheet loaded, allows it to be overwritten by the later stylesheets should a layout require it.

```
* html body * { overflow:visible }
```

Although the value *visible* is the standard for the `overflow` property, and its explicit statement superfluous, it nevertheless solves the Italics Problem for IE 5.5+. There is no solution for IE 5.01 -- luckily, this browser is increasingly rare.

In addition there are some further corrections needed in IE 5.x and IE6, so that `frame`, `iframe`, `frameset`, `textarea` and `input` elements will be displayed correctly. This is done within *iehacks.css*:

```
* html iframe,
* html frame { overflow:auto; }
* html input,
* html frameset { overflow:hidden; }
* html textarea { overflow:scroll; overflow-x:hidden; }
```

Disappearing List Background Bug

	IE 5.x/Win	IE 6.0	IE 7.0
Bug active	Yes	Yes	No

The IE Disappearing List-Background Bug is triggered when lists are placed within floating DIV containers. In YAML, this happens primarily within the *float* columns `#col1` and `#col2` as well as in every list within floating content elements. The bug causes background colors or graphics to partially or completely disappear.

Bugfix: lists are assigned the property `position:relative`. This generally has no effect on the layout - except that it reliably eliminates the bug.

```
...
* 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

The *IE List Numbering Bug* is the last in this list of structure- and layout-independent CSS bugs. It is triggered when *hasLayout* is activeted for list items of ordered lists. In this case, all available IE versions fail to correctly assign numbers to the items of the list.

Bugfix: list items are assigned the property `display:list-item`. This generally has no effect on the layout - except that it reliably eliminates the bug.

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

In addition `body` in the selector raises specifity of the bugfix in the CSS cascade.

3.5.2 Structure- and Layout-Dependent Bugfixes

As already mentioned in the introductory Section on IE Adjustments, not all bugfixes can be implemented independent of the structure and layout of any particular YAML-based site. The programmer must apply these bugfixes to suit the particular design.

This collection of bugfixes also contains those that correct the display of certain content elements. As YAML cannot know your content as you do, you must adjust your classes accordingly. All these bugfixes should be assembled in the IE Adjustment Stylesheet *patch_[layout].css*.

3-Pixel-Jog Bug

	IE 5.x/Win	IE 6.0	IE 7.0
Bug active	Yes	Yes	No

The problem: as soon as a floating container is placed to the left of the static container `#col3`, the IE 3-Pixel-Jog Bug appears. If the content of the static column `#col3` is longer than that in the *float* column, that longer content in `#col3` moves 3 pixels to the left, as in the screenshot.
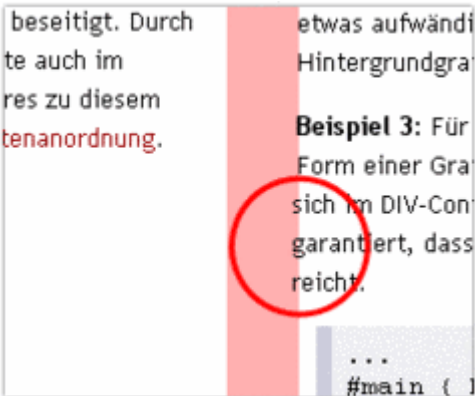
Solution: `#col3` must be assigned the CSS property `height: 1%`. This hack again works on the basis of assigning the IE proprietary property *hasLayout* to the problematic container.

However, this hack does not actually force IE to correct the mysterious jog, but rather moves all elements of container `#col3` to the right -- by exactly 3 pixels. This shudder can then be corrected with the help of two negative margins. This correction must be applied differently, depending on the order of the columns in the source code. Here is an example of a solution for the basic layout with the *float* columns each 200 pixels wide:

```
/* LAYOUT-DEPENDENT ADJUSTMENTS | 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; }
...
```

Note: the use of this bugfix for all six possible column orders of the basic layout is demonstrated in the *examples/03_layouts_3col/* folder.

Important: this bugfix collides slightly with the use of *graphic-free column separators*: they will not always reach the footer.

In these cases, you must use the "Faux Columns" technique to design column backgrounds with background images.

Handling Oversized Elements

The Internet Explorer 5.x and 6.0 expanding box problem has already been discussed and its solution for a more flexible text line break integrated in the file *iehacks.css*. But we still need tools to deal with oversized block elements (forms, tables, images, etc.).

Within flexible layouts, such elements can cause great problems inside columns with flexible widths, as IE forcibly widens the corresponding parent container, instead of letting the element itself flow out into the neighboring columns like all other browsers do.

YAML offers two different methods for solving this problem. Such elements can be put into a DIV container with the class `.floatbox`. If the content item is too wide for the parent container, the overhanging edges are cut off and layout problems avoided.

As an alternative, YAML offers the CSS class `.slidebox`. It can be assigned directly to the oversized element, which will then overlap neighboring areas of the layout without extending its parent container and destroying the layout.

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

Note: this class should only be applied to static elements: when used on floating elements, the negative margin creates an undesirable jog.

Disappearing Block Background Bug

The "Disappearing List Background Bug" is not the only bug that leads to incorrect display of background colors and images. IE 5.x and IE 6.0 have general problems displaying background images for elements with `display:block` -- as long as *hasLayout* is not activated.

The site creator must adjust these content elements specifically. Suitable CSS properties include `width`, `height,` or `zoom` used with concrete values other than `auto`.

3.6 Creating the Screen Layout

The real work for the site creator begins with the actual building of the screen layout. The basic CSS components *base.css* und *iehacks.css* provide the consistent basic layout in all browsers, yet does not supply a unique graphic design.

Your CSS declarations should be kept in a separate stylesheet so as not to interfere with the basic structure. YAML furnishes suitable structures for you, but their use is not mandatory.

Components of the Screen Layout

The construction of the screen layout can be divided into three relatively independent sections:

1. Design of the layout elements (header, footer, content area)
2. Design of the navigational elements
3. Design of the content

The YAML framework provides file templates and preformatted CSS components to create your own design in all three areas.

Avoid jumping in centered layouts

In pages that fit entirely within the browser's viewport, Firefox and Safari both hide the vertical scrollbar. Should the website suddenly become taller than the size of the viewport, vertical scrollbars appear. This disappearing act is irritating in centered layouts, as the center "jumps" from side to side.

This is not really about a bug as such, but something that simply annoys many users in centered layouts. This workaround is placed at the beginning of *basemod.css* and can be used or removed as desired.

```
/**
 * (en) Forcing vertical scrollbars in IE8, Firefox, Webkit & Opera
 *
 * @workaround
 * @affected IE8, FF, Webkit, Opera
 * @css-for all
 * @valid CSS3
 */

body { overflow-y:scroll; }
```

The CSS-3 standard property `overflow-y` forces the display of a vertical scrollbar, independent of the size of the corresponding element, and is fully supported by all the usual browsers.

Note: when validating your stylesheets as CSS 2, you will be warned that the property `overflow-y` is part of CSS 3: you can ignore this warning.

Design of the Layout Elements

The file *basemod_draft.css* in the *yaml/screen/* folder is an empty design template to be used for the basic layout resulting from the source code structure of the framework.

```css
@media screen, projection
{
 /*----------------------------------------------------------*/

 /**
  * Design of the Basic Layout
  *
  * @section layout-basics
  */

  /* Page margins and background */
  body { ... }

  /* Layout: Width, Background, Border */
  .page_margins { ... }
  .page{ ... }

  /* Design of the Main Layout Elements */
  #header { ... }
  #tovnav { ... }
  #main { ... }
  #footer { ... }

 /*----------------------------------------------------------*/

 /**
  * Formatting of the Content Area
  *
  * @section layout-main
  */

  #col1 {  }
  #col1_content {   }

  #col2 {  }
  #col2_content {   }

  #col3 {  }
  #col3_content {   }

 /*----------------------------------------------------------*/

 /**
  * Design of Additional Layout Elements
  *
  * @section layout-misc
  */


  ...
}
```

This template contains all the elements of a basic layout. You can copy this template and begin to desing the various containers as you wish. Additional elements should be integrated at the end of the file.

Here too, YAML provides examples and sample applications for your use: categorized and organized according to topic within the *examples/* folder of the download package. All the examples use the same basic screen layout, found in the corresponding *css/screen/* folder within each example topic in the CSS file *basemod.css*.

These numerous examples demonstrate how the basic YAML layout can be variously modified. This file is always the starting point for all customizations and adjustments.

Note: for now we will only discuss the basic method for creating a screen layout. Chapter 4 is dedicated to the thorough presentation of the wide-ranging modifications possible with the framework and intense analysis of many of the accompanying examples.

Desiging the Navigational Elements and the Content

These two points leave the site creator the most freedom. You can build them all completely from scratch or use YAML's many CSS components as a starting point for your designs. Due to their range and importance, each deserves its own section in the documentation: Section 3.7: Navigational Components and Section 3.8: Designing the Content.

3.6.1 Putting the Layout Together

So far, we have discussed the individual CSS components of the framework as well as the basic methods for creating a screen layout. The parts must now become a whole: the *central stylesheet* comes into play.

In Section 3.3: The Central Stylesheet, the layout's assembly is illustrated with the example of *3col_standard.html* from the *examples/01_layouts_basics/* folder of the download package.

Put all the CSS components of your layout together and link your central stylesheet to your webpage. Don't forget to set up your IE adjustments stylesheet, so that Internet Explorer has access to the *iehacks.css* stylesheet: it is absolutely necessary for the correct display of the layout.

As soon as the screen layout is finished, you can take care of any necessary CSS adjustments for Internet Explorer in your IE adjustments stylesheet.

3.7 Navigation Components

Of course a layout is never complete without a navigation. As navigational elements can become quite complex, these are managed in individual CSS files. The layout integrates them via the *central stylesheet*.

Within the YAML framework, several preformatted navigation components are available in the *yaml/navigation/* folder.

- *nav_slidingdoor.css* — Horizontal list navigation "Sliding Door II"
- *nav_shinybuttons.css* — Horizontal list navigation " Shiny Buttons"
- *nav_vlist.css* — Vertical list navigation

All listed components support tab navigation. The use of these components -- in particular the structure of the source code and the classes and IDs -- is explained briefly here. And of course, you are not at all required to use these particular components in your YAML layout.

3.7.1 Sliding Door Navigation

The first is a tab navigation based on the Sliding Door (and Sliding Door II) at A-List-Apart.

This is a flat horizontal navigation with graphic hover effects for the individual list elements. The hovers only work on standard-conform browsers (Firefox, Safari, Opera, and IE 7). The hover effect is not supported by IE 5.x and IE 6.0.
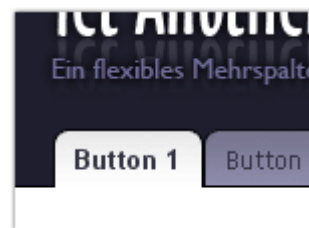
The XHTML markup of both tab navigations is simple and identical. The menu items are represented as unordered lists. The active menu item is highlighted by replacing the link itself with the element `strong`. Additionally, the surrounding list element has the CSS class `active` and can be manipulated as desired.

An excerpt of source code to demonstrate the markup structure:

```
<div class="hlist">
  <ul>
    <li class="active"><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/05_navigation/menu_slidingdoor.html

3.7.2 Shiny Buttons Navigation

The *Shiny Buttons* navigation requires few graphical elements. It uses the very same XHTML markup as the *Sliding Door Navigation*, explained
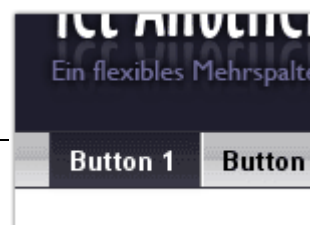
above. Switching between both design variants is easily accomplished by changing out the CSS component in the *central stylesheet* for the website.

The menu items are represented as unordered lists. The active menu item is highlighted by replacing the link itself with the element `strong`. Additionally, the surrounding list element has the CSS class `active` and can be manipulated as desired.

```
<div class="hlist">
  <ul>
    <li class="active"><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/05_navigation/mnu_shinybuttons.html

3.7.3 Vertical List Navigation

This navigation is a vertical list, usable at either a fixed or a flexible width. Up to four hierarchy levels are possible as well as highlighting of the menu title, using a `H6` heading with CSS class `vlist` right in front of the list.

The menu items are represented as unordered lists. The active menu item is highlighted by replacing the link itself with the element `strong`. Additionally, the surrounding list element has the CSS class `active` and can be manipulated as desired.

Subtitles can be easily integrated for the submenus by using the `span` element. In addition, each menu item incorporates a hover effect on mouseovers.

The file *nav_vlist.css* from the *yaml/navigation/* folder provides the functionality for this component. The corresponding (X)HTML markup is as follows:

```
<h6 class="vlist">Title</h6>
<ul class="vlist">
  <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 class="active"><strong>Button 3.2</strong></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/05_navigation/menu_vertical_listnav.html

Adjustments for Internet Explorer

When using this navigation component, you must import the file *patch_nav_vlist.css* from the *yaml/patches/* folder into the corresponding IE adjustment stylesheet:

```
/* Layout-Independent Adjustments ------------------------*/
@import url(/yaml/core/iehacks.css);
@import url(/yaml/patches/patch_nav_vlist.css); /* Box Model Corrections */

/* Layout-Dependent Adjustments --------------------------*/
@media screen
{
    ...
}
```

As the box model is particularly fault in Internet Explorer 5.x (see Section 2.4), this browser requires special adjustments for this navigation.

```
...
/* level 1 */
* html .vlist li a,
* html .vlist li strong,
* html .vlist li span { width: 100%; w\idth: 90%; }

/* level 2 */
* html .vlist li ul li a,
* html .vlist li ul li strong,
* html .vlist li ul li span { width: 100%; w\idth: 80%; }
...
```

This code sets the width of the list elements to 100 percent for IE 5.x / Windows, correcting for the faulty box model interpretation.

3.8 Content Design

YAML is a layout framework and as such provides a structure to display columns correctly in all browsers, no matter what content is added.

The structural, semantic, and visual composition of the content must be undertaken by the site designer, yet YAML does provide a starter kit with the file *content_default.css* in the *yaml/screen/* folder. This template sets up basic formatting for standard elements.

You can copy this template for your projects, change and expand it according to your needs, and integrate it into your YAML-based layout via the central stylesheet.

3.8.1 The content_default.css Template

A website's content also requires careful design. Each browser has its own set of standard predefined formats, resulting in more or less important differences in their displays.

Of course, not every single element can be explained in the documentation. That's why you will find all styled elements from *content_default.css* in the following layout example:

/examples/01_layouts_basics/styling_content.html

Setting the basic font size

The first step on the way to a uniform display is the setting of a uniform font size for all standard elements. The first step in resetting the various browser's individual settings is to define all font sizes as 16 pixels high via the `html *` selector. The odd number evens out the rounding errors in a few older browsers.

Note: the use of `html *` instead of `*` ensures that Internet Explorer will still recognize Javascript expressions for simulating the CSS properties `min-width` and `max-width`. See Section 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 all browsers
 * (de) Schriftgröße von monospaced Elemente in allen Browsern auf 16 Pixel
setzen
 *
 * @see: ...
 */
textarea, pre, code, kbd, samp, var, tt {
  font-family: Consolas, "Lucida Console", ...;
}

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

The second part is a correction for a special characteristic of Gecko-based browsers. These browsers set the standard font size of monospaced elements (`textarea`, `pre`, `tt`, `code` etc.) to 13px and not to 16px as all other browsers do. By changing the font family from *monospace* to *Courier New* or *Courier*, this problem can be avoided.

Below that we choose a new, sensible standard font size for the `body` element. As this property will be inherited, it will thus be set for all elements within `body`. For the basis: a sans serif font, 12 pixels high.

Headlines and Copytext

The next step sets the font sizes, margins, and line heights of the headlines and copytext.

```
h1,h2,h3,h4,h5,h6 {
  font-family: "Times New Roman", Times, serif;
  font-weight:normal;
  color:#222;
  margin: 0 0 0.25em 0;
}

h1 { font-size: 250% }                        /* 30px */
h2 { font-size: 200% }                        /* 24px */
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; }
```

Important: generally, the font sizes should be given in relative units of measurement to allow all browsers to zoom the text.

As soon as a value is given in **pixels** [px], Internet Explorer users (including users of IE 7) cannot use the text-zoom function of the browser to size the text to their liking.

HTML List Design

The next block deals with the design of HTML lists. The default values correspond to those in *base.css*. This redundancy is intentional: changes are easier to make when the original is available.

```
/* --- Lists | Listen  -------------------------- */

ul, ol, dl { line-height: 1.5em; margin: 0 0 1em 1em; }
ul { list-style-type: disc; }
ul ul { list-style-type: circle; margin-bottom: 0; }

ol { list-style-type: decimal; }
ol ol { list-style-type: lower-latin; margin-bottom: 0; }

li { margin-left: 0.8em; line-height: 1.5em; }

dt { font-weight:bold; }
dd { margin:0 0 1em 0.8em; }
```

Text Markup

Quotes, text emphasis, abbreviations / acronyms, and preformatted text (or code excerpts) all often require special text markup. These are included in our general formatting, with the basic properties of font face, margins, etc.

```css
/* --- general text formatting | Allgemeine Textauszeichnung --- */

...

blockquote, cite, q {
  font-family: Georgia, "Times New Roman", Times, serif;
  font-style:italic;
}

blockquote { margin: 0 0 1em 1.6em; color: #666; }

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

big { font-size:116.667%; }
small { font-size:91.667%; }

pre { line-height:1.5em; margin:0 0 1em 0; }
pre, code, kbd, tt, samp, var { font-size:100%; }
pre, code { color:#800; }
kbd, samp, var, tt { color:#666; font-weight:bold; }
var, dfn { font-style:italic; }

acronym, abbr {
  border-bottom: 1px #aaa dotted;
  font-variant: small-caps;
  letter-spacing: .07em;
  cursor: help;
}

sub { vertical-align: sub; font-size: smaller; }
sup { vertical-align: super; font-size: smaller; }
...
```

Generic Classes for Positioning and Highlighting Content Elements

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

.highlight { color: #f60; }
.dimmed { color: #888; }

.info { background: #f8f8f8; color: #666; ... }

.note { background: #efe; color: #040; ... }
.important { background: #ffe; color: #440; ... }
.warning { background: #fee; color: #400; ... }

.float_left { float: left; display:inline; ... }
.float_right { float: right; display:inline; ... }
.center { display:block; text-align:center; ... }
```

Three CSS classes have been created to highlight elements according to their contextual relevance: general information, an important note, and a warning.

The horizontal alignment of block elements is taken care of by three CSS classes: for left-aligned, right-aligned, and centered.

Automatic Formatting of Hyperlinks

CSS automatically formats external links. This process is restricted to the actual content area of the layout, the container `#main`. Adjust the URL label for your own domain.

```
/**
* 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;
  }
*/
```

If you use relative paths for internal links, you can even leave out the URL.

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

Note: the style declarations for the automatic formatting of external links are commented out in the template and must be activated for use in practice.

Important: the automatic link formatting requires the browser to support CSS 2.1 pseudoclasses. Internet Explorer unfortunately does not fulfil this criterium.

Simple Table Design

The next block deals with the display of simple tables. Normal tables are created with an automatic width, but by using the class .full the table can be forced to fill the entire width. Important to note: when using this class, additional margins or borders on the sides will automatically create an oversized element.

The second predefined CSS class, `.fixed`, allows the creation of tables at a fixed width: their cells will not expand to encompass oversized content. These tables are thus easier to incorporate into flexible layouts.

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

table { width: auto; border-collapse:collapse; ... }
table caption { font-variant:small-caps; }
table.full { width: 100%; }
table.fixed { table-layout:fixed; }

th,td { padding: 0.5em; }
thead th { color: #000; border-bottom: 2px #800 solid; }
tbody th { background: #e0e0e0; color: #333; }
tbody th[scope="row"],
tbody th.sub { background: #f0f0f0; }

tbody th { border-bottom: 1px solid #fff; text-align: left; }
tbody td { border-bottom: 1px solid #eee; }

tbody tr:hover th[scope="row"],
tbody tr:hover tbody th.sub { background: #f0e8e8; }
tbody tr:hover td { background: #fff8f8; }
```

The other definitions are self-explanatory. Column and row headlines can be clearly assigned by using the handy differences between `thead` and `tbody` as well as the elements `th` and `th.sub`.

3.9 Layout Adjustments for Printing

Preparing website content for paper is an important component of any website's design - and an attractive screen design is no hindrance to a legible and well-organized print version.

The switch between screen and printed page means changing from an interactive to a passive medium. Paper has a fixed size and proportions. Longer content areas must come to terms with page breaks - something unfamiliar in the online world. Links are no longer clickable on paper, so if the corresponding URL is not visible, important information is lost.

3.9.1 Printing Preparation

The headline does not quite capture the point. More accurately, you must merely decide if you want to print the content of all column containers, of some, or of only one.

The question is: w*hich parts of the layout contain important information and what is only decoration?*

The footer information, advertising in the margins, and search forms are all useless in print. The navigational elements are no longer usable on paper. It is unnecessary to print everything that appears on the screen, so for a start, the print stylesheets hide the footer and the main navigation.

Choosing the Printable Column Containers

Within the YAML framework, the order and thus the use of the column containers of content, navigation, or anything else, is variable. The print stylesheets are designed to let you freely choose any combination of column containers to be printed.

You choose by linking one of the seven print stylesheets from the *yaml/print/* folder in the *central stylesheet* of your layout.

Print Stylesheet	#col1	#col2	#col3
print_100_draft.css	Yes	-	-
print_020_draft.css	-	Yes	-
print_003_draft.css	-	-	Yes
print_120_draft.css	Yes	Yes	-
print_023_draft.css	-	Yes	Yes
print_103_draft.css	Yes	-	Yes
print_123_draft.css	Yes	Yes	Yes
print_draft.css	no predefined settings		

3.9.2 Structure of the Print Stylesheets

The structure of these print stylesheets is nearly identical. Most of the decisions made for the printed version of a website are independent of the columns chosen for printing. All the print stylesheet must also adjust the screen layout for paper, by doing such things as hiding unnecessary layout elements, displaying URLs, abbreviations, or acronyms, so that very little information is lost.

Switching the Units of Measurement for Font Sizes

Practical font sizes are different for the screen and for paper. Onscreen, the scalability of the type is very important, so we usually use relative units like *em* or *percent*. For print we use absolute units like *points* or *pica*.

Normal text should not be set at anything smaller than **10pt** (10 points) to be legible on paper. This basic font size is set for the `body` element and is inherited down to all child elements.

```
/* (en) change font size unit to [pt] ...   */
/* (de) Wechsel der der Schriftgrößen-Maßeinheit zu [pt] ... */
body { font-size: 10pt; }
```

Important: switching to **pt** (points) is necessary for consistent print results in Firefox. If you need to change the standard font size, do use this unit of measurement.

General Layout Adjustments

```
/* (en) Hide unneeded container of the screenlayout in print layout */
/* (de) Für den Druck nicht benötigte Container des Layouts abschalten */
#topnav, #nav, #search, nav { display: none; }
```

Navigation elements are generally turned off as they are useless in print. Please note here the `#search` selector. In the basic layout, there is no predefined container for including a search form -- tastes and styles are too different to be able to provide one solution for everyone. Of course, this element does exist in the majority of CMS-driven websites: it has been turned off here as search functionality is also not useful on paper.

Controlling page breaks

Next, we attempt to avoid page breaks immediately after a headline by using the property `page-break-after:avoid`. This too will help readability on paper.

```
/* (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; }
```

Linearization of the Container Columns

The display of the column containers must be changed for paper. It is not practical to print them on paper next to each other as they appear on the screen. Depending on the amount of content in the various columns, unnecessary white space would be printed.

To avoid this, the container columns are *linearized*, or printed in the row in which they appear in the source code -- and across the entire page. The following is an excerpt from the print stylesheet *print_103_draft.css*. In this one, the column containers `#col1` and `#col3` are adjusted for the print version, and `#col2` is turned off.

```
#col1, #col1_content { float: none; width: 100%; margin: 0; padding: 0;
border: 0}
#col2 { display:none; }
#col3, #col3_content { width: 100%; margin: 0; padding: 0; border:0; }
```

Optional Column Labeling

Linearization is practical for printing web pages of several columns. As the left- or right-alignment disappears, the column containers must appear in the same order in which they appear in the source code.

That means that in the basic layout (column order 1-3-2), column `#col3` -- which usually contains the main content -- would be printed last. As long as only this column is printed, this is irrelevant.

When several columns are printed, the hierarchy of the columns and their contextual relation to each other can be lost as a result of the linearization. To improve the user's orientation, optional headings can be added to each column container for the print layout, naming perhaps the column's position on screen or labeling its content. This is simple and elegant with the CSS 2 pseudoclass `:before`.

```
/* (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;
}
```

Should a title be desired, the corresponding container need only be provided with the value for the `content` property.

```
/* Optional Column Titles | Optionale Spaltenauszeichnung */
/*
   #col1_content:before {content:" [ left | middle | right column ]"}
   #col3_content:before {content:" [ left | middle | right column ]"}
*/
```

Note: the optional naming of the columns is predefined in all print stylesheets which print more than one column, but for safety's sake is commented out.

Automatic Display of URLs, Acronyms and Abbreviations

As mentioned at the beginning, paper is static. Hyperlinks cannot be clicked, yet the URL should not be completely lost -- neither should explanatory text for acronyms or abbreviations.

We must ensure that these items appear on the printed page. A CSS2 pseudoclass helps us avoid this stumbling block.The additional text is printed in parentheses, URLs in brackets, each directly after the corresponding element.

```
/* (en) Disable link background graphics  */
/* (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:" <URL: "attr(href)">";
   color:#444;
   background:inherit;
   font-style:italic;
}
```

Important: the following passages from the print style sheet require CSS 2.1 pseudoclasses in the browser. Internet Explorer including Version 7 unfortunately does not meet these requirements.

These declarations allow URLs and explanatory texts to print directly after the linked text or marked abbreviation. Little information from the website is lost in the transition to paper.

Note: this option is predefined in all print stylesheets but for safety's sake is commented out.

3.10 The Form Construction Kit

Forms, necessary though they may be, are no fun to program. Their elements are limited in their flexibility, as most browsers automatically display them to conform to the user's operating system. The difficulty of a uniform presentation is only increased by certain CSS bugs in Internet Explorer.

YAML 3.1 introduces a Form Building Kit as a new building block in the layout framework. As with the subtemplates, it is a system of HTML building blocks and the corresponding CSS. The markup is programmed according to best practice rules and is written to support accessibility standards.

An example for using the kit as well as the two basic design options (linear or column-based) is here:

/examples/01_layouts_basics/styling_forms.html

Apart form this simple structure, with the help of subtemplates also multi-column forms can be easily implemented.

/examples/01_layouts_basics/multicolumnar_forms.html

3.10.1 The Markup

Of course you are free to program your forms any way you wish - using YAML does not force you to use the Form Construction Kit.

To begin using the kit, assign the surrounding `<form>` element the CSS class `yform`.

```
<form method="post" action="" class="yform">
  <fieldset>
    <legend>fieldset heading</legend>
    ...
  </fieldset>
</form>
```

You can structure your form by using the `<fieldset>` tag, though this element is not required. Use fieldsets to group elements in longer forms. Short forms (like a simple contact form) will not require such subgroupings.

HTML Blocks for Form Elements

The following overview contains the standard markup for all the form elements of the Construction Kit. As you can see, each HTML block consists of a form element (INPUT, TEXTAREA, SELECT etc.), the corresponding label, and an enclosing DIV container. The CSS class for the container determines its design and position.

Textfield

```
<div class="type-text">
  <label for="your-id">your label</label>
  <input type="text" name="your-id" id="your-id" size="20" />
</div>
```

Textarea

```
<div class="type-text">
  <label for="your-id">your label</label>
  <textarea name="your-id" id="your-id" cols="30" rows="7"></textarea>
</div>
```

Select

```
<div class="type-select">
  <label for="your-id">More Options</label>
  <select name="your-id" id="your-id" size="1">
    <option value="0" selected="selected" disabled="disabled">Please
choose</option>
    <optgroup label="First options to choose from">
      <option value="value #1">Option 1</option>
      <option value="value #2">Option 2</option>
    </optgroup>
    <optgroup label="Yet more options to choose from">
      <option value="value #3">Option 3</option>
      <option value="value #4">Option 4</option>
      <option value="value #5">Option 5</option>
    </optgroup>
</select>
</div>
```

Checkbox

```
<div class="type-check">
  <input type="checkbox" name="your-id" id="your-id" />
  <label for="your-id">Your checkbox label</label>
</div>
```

Radio-Buttons

```
<div class="type-check">
  <input type="radio" name="your-id" value="value #1" id="your-id" />
  <label for="your-id">Your radio-button label</label>
</div>
```

Button-Set

```
<div class="type-button">
  <input type="button" value="button" id="button1" name="button1" />
  <input type="reset" value="reset" id="reset" name="reset" />
  <input type="submit" value="submit" id="submit" name="submit" />
</div>
```

You can transfer these blocks to your source code just by copying and pasting. Make sure that you use a unique name for the attribute `id="your-id"`, as it must be used in the `for="your-id"` attribute of the label in order to connect the label to the correct form element.

Note: the "name" attribute within the form element is optional and may be left out. A few JS-based form validators use this attribute: should you choose to use one of these scripts, make sure you use the same word for the name as well as for the id.

The visual appearance of the form elements is controlled indirectly, by a surrounding div container with a particular CSS class (for instance: *type-text*). This method allows us to influence input fields, checkboxes, and radio buttons even in older versions of Internet Explorer (IE5.x and IE6). If we were using attribute selectors, like `input[type="text"] { ... }`, we could never design for these browsers as they do not understand the constructs.

The following predefined classes are ready to use:

CSS Class of the Parent Element (DIV)	Affected Form Elements
type-text	input fields, text areas
type-select	select boxes
type-check	checkboxes, radio buttons
type-button	buttons (i.e.: reset, submit)

Note: hidden input fields (type="hidden") can be inserted anywhere in these predefined containers. They are always invisible, no matter how the other input elements are defined.

3.10.2 The CSS of the Form Components

The second component of the Form Construction Kit comprises the stylesheet *forms.css*, which is saved in the folder *yaml/screen/*. This is not a core component of the framework which may not be edited, but a stylesheet that you are free to change to meet your needs.

Visual Design of the Form Elements

The stylesheet *forms.css* is divided into two sections. The first part contains all the CSS rules for the visual presentation of the individual elements.

This part can be edited at will to suit the appearance of your forms, the fieldsets, as well as all the various form elements to your own site design.

```
/**
 *  YAML Forms – visual styling
 *
 *  (en) visual form styling area
 *  (de) Festlegung des optischen Erscheinungsbildes
 */

.yform {
  background: #f4f4f4;
  border: 1px #ddd solid;
  padding: 10px;
}
...
```

The visual design includes colors, borders, margins, and perhaps images for specifying the appearance of the various form elements -- including the `:focus`, `:hover` and `:active` conditions.

Technical Basis of the Form Construction Kit

The second section contains the definitions for positioning the elements in a way which increases the accessibility of the forms. The standard form is a vertical, linear sequence of labels and form elements.

```
/**
 * Vertical-Forms - technical base (standard)
 *
 * |-----------------------------|
 * | fieldset                    |
 * |-----------------------------|
 * |    label                    |
 * |    input / select / textarea    |
 * |-----------------------------|
 * | /fieldset                   |
 * |-----------------------------|
 *
 * (en) Styling of forms where both label and ...;
 * (de) Formulargestaltung, bei der sowohl Label als auch ...
 *
 * ...
 */

/* General form styling  | Allgemeine Formatierung des Formulars */

.yform { overflow: hidden; }
.yform fieldset { overflow: hidden; }
.yform legend { background: transparent; border: 0; }
.yform label { display:block; cursor: pointer; }

...
```

The specifics of the CSS rules cannot be explained here. The construction of the form components bases on proven best practice rules and ensures correct display in all relevant browsers, in fixed and in flexible layouts.

Note: the CSS contains further preventive measures to preclude display errors in older versions of Internet Explorers, in particular relating to the relative positioning of the form elements. Again, the complexity of the bugfixes is too great to explain here in the space available.

Alternative Display Variation

As an alternative to the vertical ordering of labels and fields, the Form Construction Kit offers the css class `.columnar`. When added to a form element, a fieldset, or a DIV container, this class forces the form elements into two columns.

Each label then appears in a row with its corresponding form element (based on floats), with labels in the first column and the elements in the next.

```
/**
 * Columnar forms display - technical base (optional)
 *
 * |----------------------------------------|
 * | fieldset                               |
 * |----------------------------------------|
 * |                                        |
 * |    label    |    input / select / textarea    |
 * |                                        |
 * |----------------------------------------|
 * | /fieldset                              |
 * |----------------------------------------|
 *
 * (en) Styling of forms where label floats left of form-elements
 * (de) Formulargestaltung, bei der die label-Elemente nach links fließen
 *
 * ...
 */

/* Columnar display | Spalten-Darstellung */
.columnar .type-text label,
.columnar .type-select label {
  float: left;
  width: 30%; /* Can be fixed width too | Kann auch eine fixe Angabe sein */
}

/* Indent Checkbox fields to match label-width ... */
.columnar div.type-check { padding-left: 30%; }
.columnar div.error .message { margin-left: 30%; }

.columnar div.type-text input,
.columnar div.type-text textarea { width: 67.8%; }
.columnar div.type-select select { width: 69.4%; }

/* width adjustments for IE 5.x & IE6 ... */
* html .columnar div.type-text input,
* html .columnar div.type-text textarea { width: 67.2%; }
* html .columnar div.type-select select { width: 68.8%; }
```

The columns are set to 30% width for the label and 70% for the form element. The widths of the elements have been chosen carefully and thoroughly tested: avoid changing them if you can.

The odd-looking numbers are a result of the necessary jiggling for flexible width layouts: the exact width of the form elements cannot be determined, as paddings on the side are generally given in PX or EM. Even more annoying, select elements generally have different widths in various browsers. The given widths ensure an extremely similar width for all element types and simultaneously avoid annoying line breaks in the floated surroundings.

3.10.3 Adjustments for Internet Explorer

As in the elements for the page layout, Internet Explorer versions 5.x - 7.0 still need help in displaying forms that will look like those in the other browsers.

General Adjustments

Many display errors are connected to the use of `<fieldset>` elements -- fieldset backgrounds are not completely rendered, for example. These problems are so general that they are corrected in the *iehacks.css* file for the entire YAML framework.

```
/**
 * Form related bugfixes
 *
 * @bugfix
 * @affected   IE 5.x/Win, IE6, IE7
 * @css-for    IE 5.x/Win, IE6, IE7
 * @valid      no
 */

fieldset, legend { position:relative; }
```

The bugfix for these display problems involves assigning the property `position:relative`. This property generally does not influence a form's design, so it can be assigned directly to the elements and thus correct all forms within any YAML layout.

```
/**
 * Global fixes for YAML's form construction set
 *
 * @workaround
 * @affected IE 5.x/Win, IE6, IE7, IE8
 * @css-for IE 5.x/Win, IE6, IE7, IE8
 * @valid no
 */
.yform,
.yform div,
.yform div * { zoom:1; }
```

The second part specifically concerns the Form Construction Kit. Therefore the definitions are located in *forms.css*. The kit elements are assigned the property `zoom:1` to activate *hasLayout* -- just in case. As this method could change existing forms, it is limited to the YAML form kit (`.yform`) as a preventive measure.

Note: `zoom` is a proprietary CSS property of the Internet Explorer that does not validate. This error can be ignored as this property doesn't have any influence on other browsers.

Correct Display of Legends Within Fieldsets

A further problem in Internet Explorer is the problematic display of fieldsets with legends, anytime these are centered above the upper edge of the fieldset. Unfortunately, this is the exact default position for legends in any other web browser, so a workaround had to be built into *forms.css*.

```
/**
 * Forms Fieldset/Legend-Bug in IE
 * @see ...
 *
 * @workaround
 * @affected IE 5.x/Win, IE6, IE7, IE8
 * @css-for IE 5.x/Win, IE6, IE7, IE8
 * @valid no
 */

/* all IE */
.yform { padding-top: 0\9; }
.yform fieldset { padding: 0 5px\9; padding-top:1em\9; }
.yform legend { position:absolute\9; top:-.5em\9; *left: 0\9; }
.yform fieldset { position:relative\9; overflow:visible\9; margin-
top:1.5em\9; zoom:1; }

/* IE5.x, IE6 & IE7 */
.yform legend { *padding: 0 5px; }
.yform fieldset { *padding-top:1.5em; }

/* IE5.x & IE6 */
* html .yform { padding-top: 10px; }
```

The workaround uses absolute positioning to place the legend above the fieldset and thus allow a display equivalent to the standard default.

As it is absolutely positioned, the legend drops out of the normal text flow: this means that the upper padding and margin of the fieldset must be adjusted so that neighboring and child elements can continue to be placed precisely.

Note: the necessary CSS hacks for this IE workaround will **not validate.** However the adjustments of the padding and margins of the fieldset depend upon the individual layout (visual presentation of the forms, see above). This means that developers must be able to edit these properties: thus they are included at the very end of *forms.css*.

4 Practice

4.1 Five Rules...

The following rules summarize the basic principles defining YAML's development:

Rule 1: YAML is not a Prefab Layout

> YAML bases on web standards and is a versatile tool for creating flexible, accessible CSS layouts. The best basis for effective work with the framework is a thorough understanding of YAML's structure and workings. Please take time to read the documentation before you begin your work.

Rule 2: YAML is Based on the Top-Down Principle

> YAML provides a flexible, multi-column layout with all the important standard web page elements and functional stylesheets for correct display in all browsers, as well as an optimized layout for the printed version. The user optimizes the finished layout by deleting unneeded elements from the source code.

Rule 3: CSS Basic Components

> Every YAML-based layout needs two basic CSS components base.css and iehacks.css from the yaml/core/ folder. They are responsible for the correct display in all browsers and ensure perfect printing.

Rule 4: Separation of YAML and User CSS

> The files in the YAML folder should remain unchanged. Custom stylesheets or changed versions of YAML's CSS components belong in the user's own separate CSS folder. Only then can the layout's development basis remain stable over time and bugfixing as well as maintenance and updates are simplified.

Rule 5: Have Fun with YAML!

4.1.1 Samples Included

In addition to the documentation, the *examples* folder in the YAML download package contains a great number of prefabricated sample layouts, which can help you understand how the framework functions and serve as a starting point for your own projects.

Note: if you are new to YAML, please take the time to read the documentation through to the end. Chapter 4 contains the complete instructions for practical use, which you should read before you start.

These examples introduce the basic layout's various modification possibilities, as well as the use of the various CSS components provided. Section 1.5: The Structure of the Download Package provides an overview.

4.1.2 Tips for CSS Beginners

If you are not yet familiar with CSS, take one of the examples which fits your design requirements best and play around with the various style definitions of the screen layout. Try out what changes do what to the layout.

Change margins, font sizes, colors, and container widths. Messing around with them will help you overcome any awe of CSS and you'll quickly learn exactly what parts of YAML do what -- and how.

4.2 Recommended Project Structure

There are generally no requirements for working with YAML. The project structure recommended here has proved to be practical, as it makes bugfixing easier when creating a layout and maintenance easier when a new version of YAML is released.

4.2.1 Step 1: Creating Files and Folders

First, copy the complete *yaml/* folder onto your server and create another folder on the same hierarchical level called *css* for your own unique CSS files.

XHTML Source Code: copy the XHTML template *markup_draft.html* from the *yaml/* into your project folder and rename the file.

Central Stylesheet: copy the stylesheet template *central_draft.css* into your *css* folder and rename the file accordingly.

IE Patches: copy the file template *patch_layout_draft.css* from the *yaml/patches/* folder into your *css/my_patches/* folder and rename it to match the name of your central stylesheet (so that the relationship is easier to remember).

The screenshot shows the protostructure of your new project (as seen in a Dreamweaver project window).
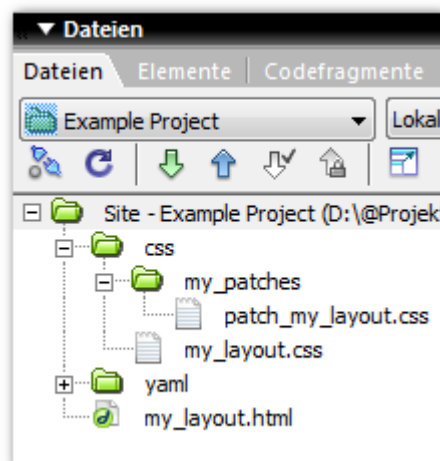
4.2.2 Step 2: Adjusting the Paths

After creating the project structure, you must check all paths for the CSS components. The XHTML source code must contain the paths to the central stylesheet and to the patch file. The central stylesheet and patch file themselves must have the correct paths to *base.css* and *iehacks.css*. After these checks, the basic layout is ready to go and the real graphic design can be implemented.

4.2.3 Step 3: Layout Design

From this point on, you have the choice: you can create your own stylesheets for the screen and print layouts as well as for the navigation, or you can start off with YAML's file templates and preformatted CSS components.

The folder *yaml/screen/* contains the file templates *basemod_draft.css* for the page layout and *content_default.css* to format content.

Copy these templates into your *css* folder and change them to suit your wishes. You can work with the navigation components and the print stylesheets in the same fashion.
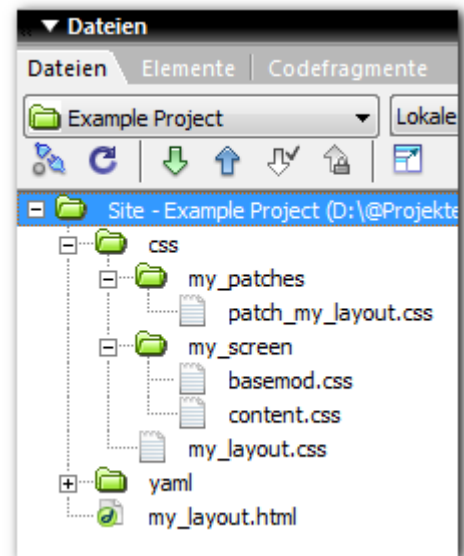
Note: do not forget to include these additional components in your central stylesheet.

4.3 Basic Variations

YAML offers you many ways to customize the basic layout to your wishes. I will explain these possibilities in this and the following sections. First let us examine the (X)HTML source code structure and the column order within.

The order of the column containers in the (X)HTML source code is fixed and should not be changed: all CSS components, in particular the adjustments for Internet Explorer, depend on this structure.

The basic layout can be varied and yet retain its full functionality in all browsers -- in particular the IE clearing, which ensures that `#col3` even in IE remains the longest column and permits graphic-free column separators.

Accessible layouts often demand that the actual content of a page be at the very beginning of the source code. The idea is to allow text browsers or screenreaders easy access to the main subject matter. Other page elements (sidebars, advertising, etc.) should then follow further down.
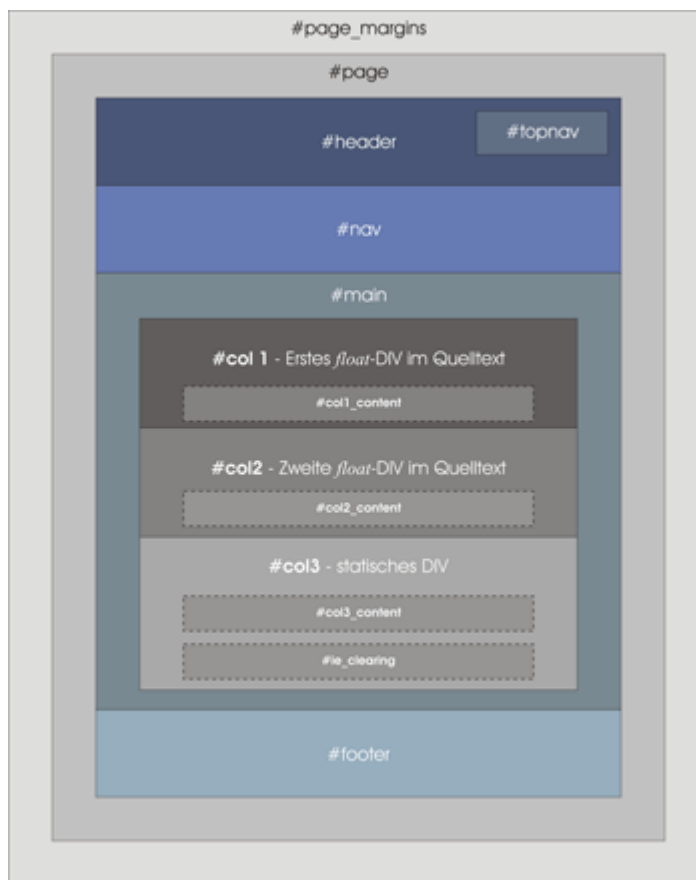
Note: for the three-column layouts, Section 4.4 thoroughly describes the means for YAML to fulfil this concept absolutely. This involves the completely free ordering of the individual columns on the screen, independent of their position in the source code.

The disadvantage of the independent column order is that four of the six possible variations are incompatible with the IE clearing, and thus can no longer utilize graphic-free column separators.

4.3.1 3-Column Layouts
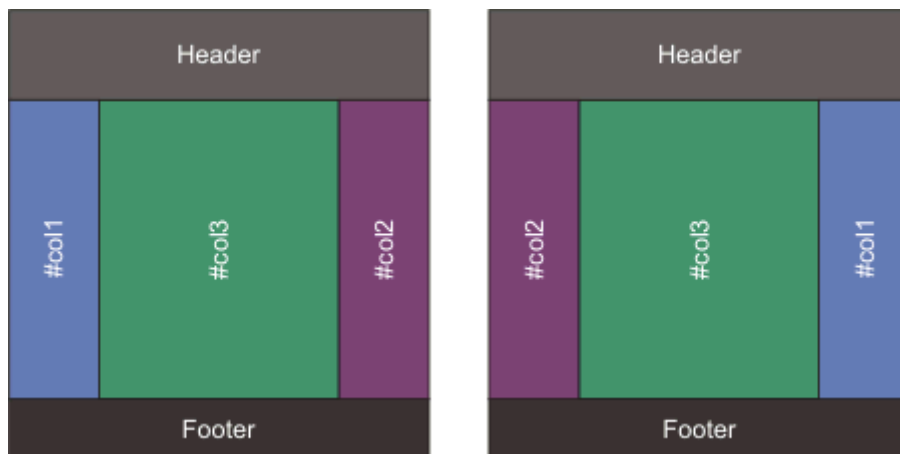
The basic layout uses the column order 1-3-2. The static column `#col3` is surrounded by the two *float* containers `#col1` and `#col2`. To switch to column order 2-3-1, you must merely change the *float* direction.

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

Switching the property allows you to change the layout order of the content in your side columns. You can use this method to layout a subnavigation on either the left or the right and yet still have it directly follow on the main navigation in the source code. The subnavigation need merely be placed in the column `#col1` and one of the two column orders to locate it either on the right or the left.

In both cases, `#col3` is meant for the main content and is in last place in the source code. This is certainly not ideal for accessibility purposes, but is easy enough to compensate for via the skip-links built in to the standard layout.
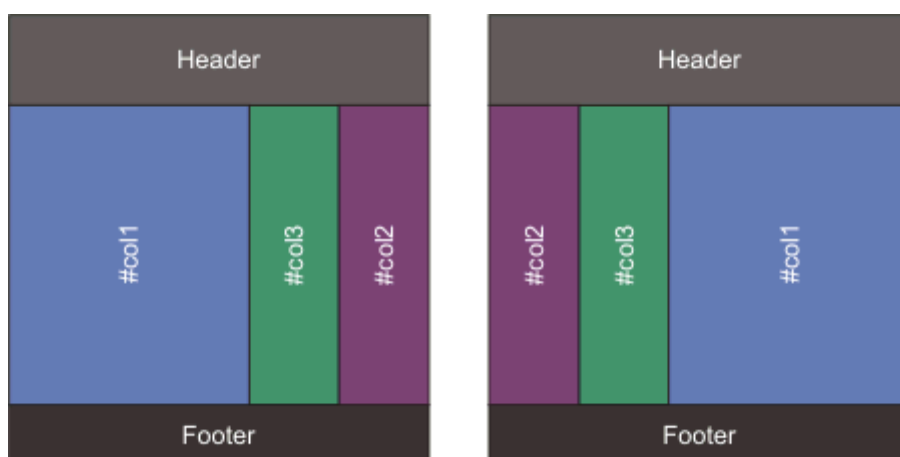
That is certainly the most often-used column arrangement -- but it is by no means the only one. An alternative layout can use one of the side columns for the main content. In this case, navigation, sidebars, and extras can appear in two narrow columns next to each other.

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

This variation also allows the switching of the *float* direction of the two columns #col1 and #col2, depending on the location of the main content, left or right. The advantage here is that the static column #col3 is still between the two side columns and the use of graphic-free column separators presents no problems.

4.3.2 2-Column Layouts
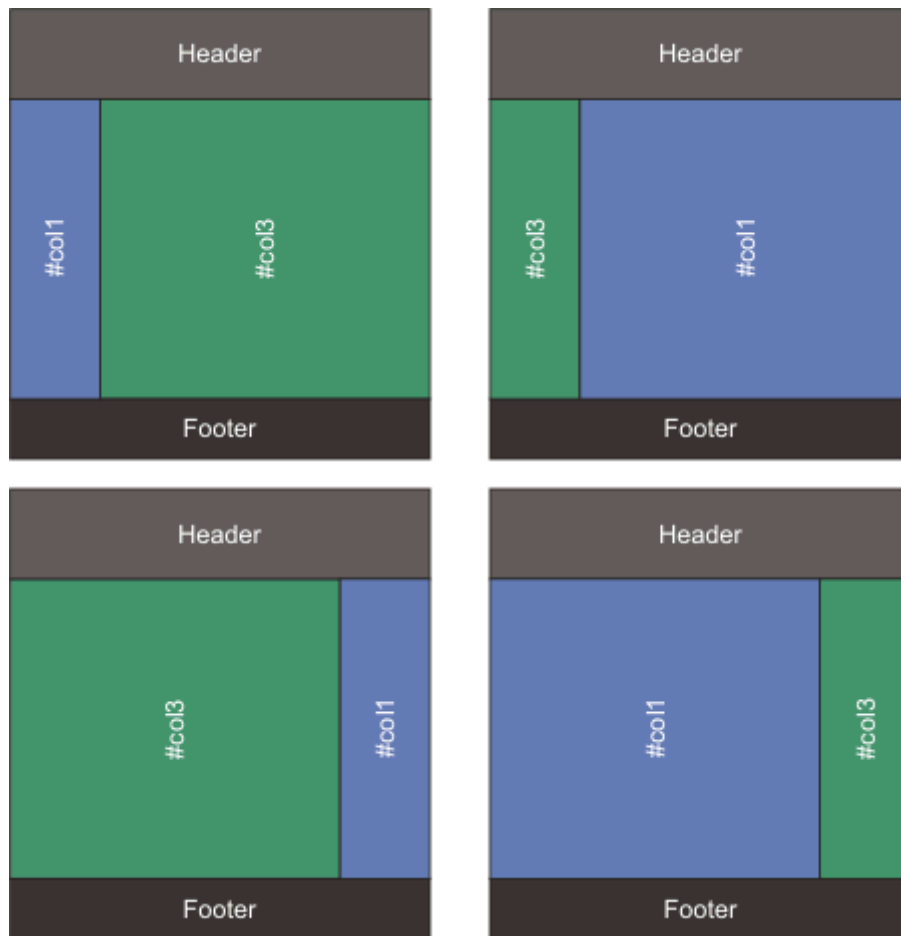
Two columns also allow an optimal placement of content in the source code while yet retaining full control of its position in the layout. Usually a narrow column will contain the navigation, and a wide column holds the content.

In our example, the navigation should appear on the left. There are two ways to accomplish this.

These images demonstrate the possibilities for column arrangement. Generally one uses one floating container (#col1) and one static container (#col3).

All these combinations provide full framework functionality: by this we mean the graphic-free column separators or backgrounds. Simultaneously, the content can be placed in the source code in a location optimized for search engines.

The required changes in the basic layout are minimal. The CSS must change the left / right orientation of the container `#col1` and the corresponding margins for `#col3`. The width of the column alone determines which will perform which function within the layout.

Note: in the samples of the download package, there are four varieties of two-column layouts, each realized with containers `#col1` and `#col3`. All the possible combinations for the container order have been included.

4.3.3 Further Alternatives for Sorting the Containers

But wait, there's more! The previous two-column layouts merely hid one of the two *float* columns. Yet a two-column layout can be built from `#col2` and `#col3`, leaving `#col1` available for other purposes.

The standard layout treats the three containers as columns of a multi-column layout. Of course - only you decide which container is used for what purpose and in which order.

The example to the right needs an additional container in full width between the page header and the two-column main area. In this case, it is simple to place `#col1` directly above the two other columns `#col2` and `#col3`.
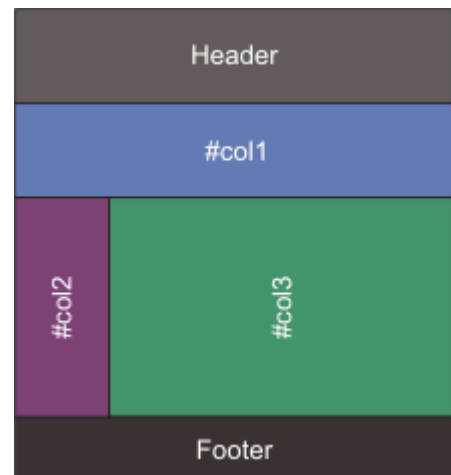
```
#col1 {float: none; width: auto; }
#col2 {float: left; width: 25%; }
#col3 {margin-left: 25%; margin-right: 0 }
```

There are few restrictions in the placement of the column containers on the screen. As the source code itself remains unchanged, it is quite easy to recognize and work around possible stumbling blocks in the known IE CSS bugs.

4.3.4   Generic CSS Classes for Layout Design

Besides the possibility of alternative layout options through the integration of alternative stylesheets, taking advantage of the CSS cascade, you can now use the following standard classes to hide and display the columns.

```
/**
 * @section generic classes for layout switching
 * @see     www.yaml.de/en/documentation/practice/basic-variations.html
 *
 * .hidecol1 -> 2-column-layout (using #col2 and #col3)
 * .hidecol2 -> 2-column-layout (using #col1 and #col3)
 * .hideboth -> single-column-layout (using #col3)
 */


.hideboth #col3 { margin-left: 0; margin-right: 0; }
.hideboth #col3_content{ padding-left: 20px; padding-right: 20px; }

.hidecol1 #col3 { margin-left: 0; margin-right: 25%; }
```

```
.hidecol1 #col3_content{ padding-left: 20px; }

.hidecol2 #col3 { margin-left: 25%; margin-right: 0; }
.hidecol2 #col3 content{ padding-right: 20px; }

.hideboth #col1, .hideboth #col2,
.hidecol1 #col1, .hidecol2 #col2 { display:none; }
```

The relevant class should be assigned either to the `body` element or any other parent container of your columns (e.g. `#main`).

These classes must of course be adjusted to the desired column widths of the screen layout. Here they are meant as orientation tools and have only the standard values.

Note: the use of these classes for modifying the layout is particulary useful in the context of Content Management Systems. Many CMS do not offer access to the HTML header, so that exchanging stylesheets for layout modification is difficult to impossible. Alternate versions of the basic layout often require separate templates.

Manipulating the HTML elements within the `body`, on the other hand, is generally simple. By using these generic classes, a template can yet be easily modified.

The following example demonstrates the usage of these generic CSS classes:

examples/04_layouts_styling/dynamic_layout_switching.html

4.4 Variable Order and Use of Content Columns

Section 4.3 demonstrated several fundamental variations on the basic layout. Those all fulfilled the requirement that YAML's full functionality (including the use of the borders on `#col3` to create column separators or backgrounds, see Section 4.6) remain intact.

This requirement is merely a design criterium and not absolutely necessary when developing a layout. With regards to a website's accessibility (for example, its display in text browsers), other criteria can be more important, which might even demand a different order of the column containers than that of the basic layout.

Many web designers prefer to place the content close to the beginning of the source code, and leave the less important elements such as the navigation or the sidebars for later. Though the necessity of this sorting is debatable, the discussion must be carried out elsewhere. Here, we will see how YAML can also fulfil this demand.

Note: the following CSS excerpts were taken from the sample layouts in the folder *examples/03_layouts_3col/*. You will find the corresponding *basemod_xy.css* files in the *css/screen/* folder, which modifies the column order of the basic screen layout.

Important: the bugfix for the IE 3 Pixel Bug is built into the patch files of this layout sample, as its basic use can be demonstrated here with several different column orders.

4.4.1 Ordering Columns

The greatest design freedom can be had when the order of the column containers in the source code has no influence on their position on the screen. In this case, the web designer can place the content in the source code according to other demands (accessibility, search engine optimization, etc.) and has complete control over their screen and paper layout via the stylesheets.

As described in Section 4.3, the order of the columns in the source code cannot be changed at will. But it is also completely unnecessary.

The position and order of the columns on the screen is completely controlled via CSS. You must only insert your content at that point of the source code where we'd like it. Afterwards, the containers are arranged with CSS, and
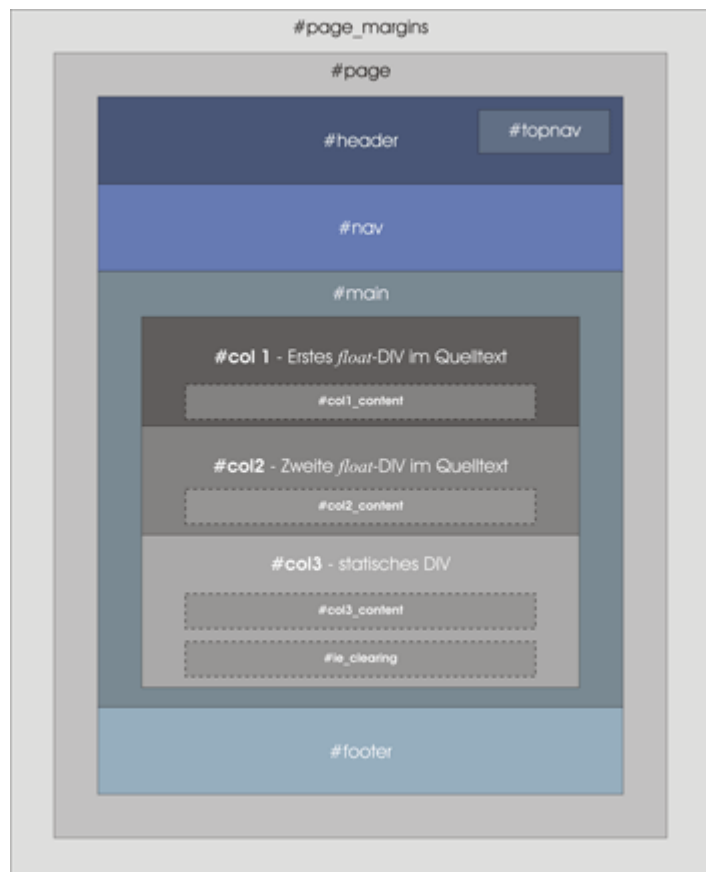
variously dependent upon the final medium.

For three columns with three various contents, there are exactly six possible combinations for their placement next to each other on the screen. These combinations are described and their limitations outlined in the following.

All these combinations use a three-column layout with proportions 25 | 50 | 25 percent. The positioning examples are in the *examples/03_layouts_3col/* folder of the download package.

The most important characteristics have been summarized in this table for each possible column set.. The following legend explains the table's abbreviations:

Abbreviation	Explanation
U-Mix	Various units of measurement can be mixed within the layout to set column width: fixed (pixels), flexible (%), and elastic (EM).
Percent	A flexible layout is possible with all column widths set as percents.
Pixels	A fixed layout is possible with all column widths set in pixels.
EM	An elastic layout is possible with all column widths given in EM / EX values.
3P-Fix	The 3 Pixel Bug can be overcome.
SPT	The border property of `#col3` can be used to represent graphic-free column separators or backgrounds.
Faux	The *"Faux Columns"* technique for displaying column separators or backgrounds is applicable.

4.4.2 Column Order 1-3-2 and 2-3-1

Layout	U-Mix	Percent	Pixels	EM	3P-Fix	SPT	Faux
1-3-2	Yes	Yes	Yes	Yes	Yes *)	Yes	Yes
2-3-1	Yes	Yes	Yes	Yes	Yes *)	Yes	Yes

*) The use of graphic-free column separators and the fix for the 3 Pixel Bug via `#col3` are mutually incompatible.
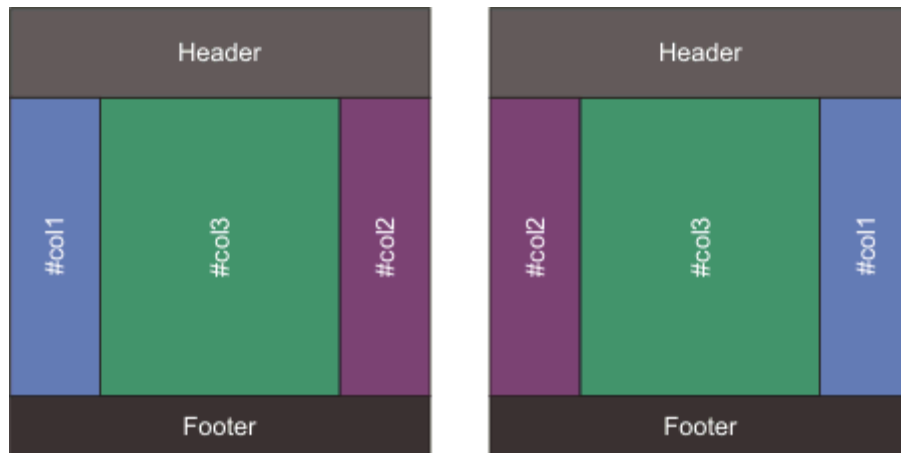
The column order 1-3-2 corresponds exactly to the standard definition, as anchored in the file *base.css* (see Section 3.4). I discussed voth variations while explaining the three-column layouts in Section 4.3.

```
/* #col1 becomes the left column | wird zur linken Spalte */
#col1 { width: 25%; }

/* #col2 becomes the right column | wird zur rechten Spalte */
#col2 { width: 25%; }

/* #col3 becomes the middle column | wird zur mittleren Spalte */
#col3 { margin-left: 25%; margin-right: 25%; }
```

03_layouts_3col/3col_1-3-2.html

To display the reverse order, 2-3-1, we needn't change the order of the columns in the source code -- merely change the *float* direction of the two columns in a *basemod_xy.css* file.

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

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

/* #col3 becomes the middle column | wird zur mittleren Spalte */
#col3 { margin-left: 25%; margin-right: 25%; }
```

That's it. The screen would now show the order 2-3-1.

03_layouts_3col/3col_2-3-1.html

4.4.3 Column Order 1-2-3 and 3-2-1

Layout	U-Mix	Percent	Pixels	EM	3P-Fix	SPT	Faux
1-2-3	-	Yes	Yes	Yes	Yes	-	Yes
3-2-1	-	Yes	Yes	Yes	Yes	-	Yes
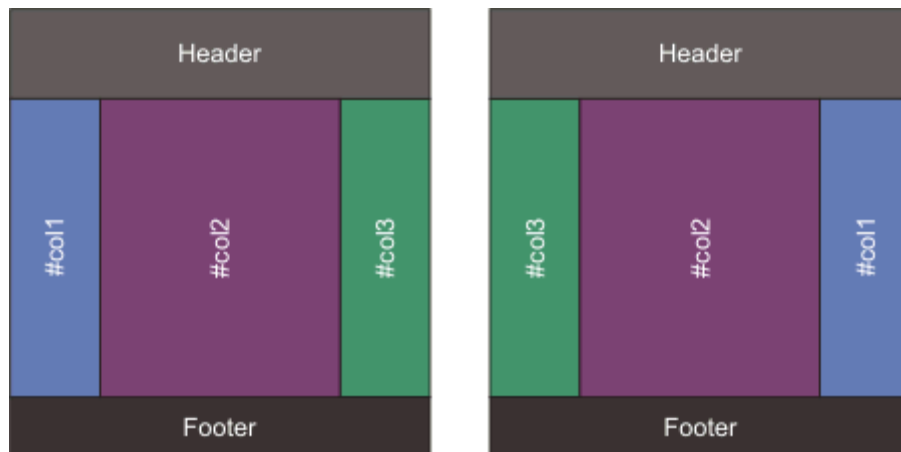
The columns should display in either 1-2-3 from left to right, or in the opposite order, 3-2-1, in which they appear in the source code.

This presentation order is also simply manipulated. First, the two *float* columns must be placed next to each other. For that, both containers need only float in the same direction. So for the order 1-2-3, #col2 must float:left, and for the order 3-2-1, the container #col1 must float:right.

In the second step, #col3 is shoved to the left or right edge. This is easy enough with a margin on one side which is exactly as wide as the two columns #col1 and #col2 together.

For the column order 1-2-3, the containers are sorted from left to right.

```
/* #col1 becomes the left column | wird zur linken Spalte */
#col1 { width: 25%; margin: 0;}

/* #col2 becomes the middle column | wird zur mittleren Spalte */
#col2 { width: 50%; float:left; margin: 0;}

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

03_layouts_3col/3col_1-2-3.html

For the column order 3-2-1, the containers are sorted from right to left.

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

/* #col2 becomes the middle column | wird zur mittleren Spalte */
#col2 { width: 50%; margin: 0;}
```

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

03_layouts_3col/3col_3-2-1.html

4.4.4 Column Order 2-1-3 and 3-1-2

Layout	U-Mix	Percent	Pixels	EM	3P-Fix	SPT	Faux
2-1-3	-	Yes	Yes	Yes *)	not required	-	Yes
3-1-2	-	Yes	Yes	Yes *)	not required	-	Yes

*) EM based column widths are possible only if the layout width is also defined in EM.
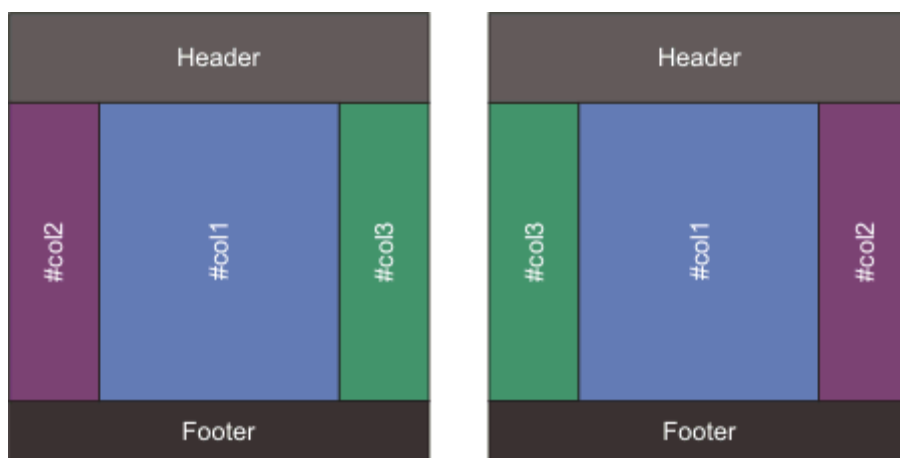
The last two combinations let the first column in the source code order be placed in the middle on the screen. The previously described column order shows that when `#col1` and `#col2` have the same float direction, they appear in the same order onscreen as they do in the source code. We have to change that now.

At this point, we need to think outside of the box: other creative people have already wracked their brains over this issue and have found a wonderfully simple solution: negative margins. Alex Robinson uses this technique in the *"any order columns"* section of his article "In search of the One True Layout". By using negative margins, the two columns can be moved to precisely the position necessary. The same principle can be used on both YAML's *float* columns.

The first step ensures that both `#col1` and `#col2` float in the same direction: both are assigned `float:left`. Then needs a margin-left that's exactly as wide as `#col2`. `#col1` is then already in its final position, and `#col2` floats right up next to it -- but on its right, not yet on the left.

Now come the negative margins. Our reference point on `#col2` is the top left corner. In order to move it to the left of `#col1`, it has to be moved to the left by its own width as well as the width of `#col1`. The resulting margin totals up to -75 percent. The final step moves `#col3` to the right side by adding `float:right` property.

As all columns are floats now, we have to force `#main` to contain its floating children. This is done by adding `float:left` property.

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

/* #col1 becomes middle column */
#col1 { width: 50%; float:left; margin-left: 25%; }

/* #col2 becomes left column */
#col2 { width: 25%; float:left; margin-left: -75%; }

/* #col3 becomes right column */
#col3 {
  margin-left: -5px;
  margin-right: 0;
  float:right; width: 25%;
}
```

In this column order, the IE Doubled Float Margin Bug (see Section 2.13.5) would usually strike - literally doubling all margins and absolutely destroying this layout. But have no fear: the corresponding bugfix is already integrated in the file *iehacks.css* and incorporated into every YAML-based layout.

03_layouts_3col/3col_2-1-3.html

The procecure for the column order 3-1-2 is quite similar: just the *float* directions for #col1 and #col2 are switched out, the margins added together for the right side, and a different column gets the negative margin, as IE fails to comprehend a negative margin-right for #col2.

So #col1 must float:right, the same direction as #col2. Then #col1 is moved to the middle with a negative margin of the sum of its width and the width of #col2 (margin-left: -75%). To ensure that older versions of IE can still play along nicely, the margins for both sides are explicitly assigned for each column. Now that #col1 is in the middle of the page, #col2 floats up to the right. Last, #col3 lands on the left side and again, #main is forced to contain its floating children.

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


/* #col1 becomes middel column */
#col1 { width: 50%; float:right; margin-left: -75%; margin-right:25%;}

/* #col1 becomes right column */
#col2 { width: 25%; float:right; margin-right: 0%;}

/* #col3 becomes left column */
#col3 {
  margin-left: 0;
  margin-right: -5px;
  float:left; width: 25%;
}
```

03_layouts_3col/3col_3-1-2.html

Column arrangement with negative margins works in all modern browsers. Alex Robinson points out that Netscape 6 & 7 and the older Opera 6 still have problems, but the current browser version Netscape 8.x did fine in our testing, and Opera 6 is, shall we say, antiquated.

4.4.5 The Upshot

YAML allows you to arrange your columns on the screen in any order, completely independent of their position in the source code.You alone decide which column will contain which content, navigation, or sidebar. The advantages and disadvantages of the various placement methods are easily compared with their relative usefulness.

Note: YAML with its print stylesheets offers an optional heading for the column containers for the print version. This can be useful when the linearized presentation is set to print the containers in a different order than they appear on the screen.

4.5 Subtemplates

The website is of course not finished once the basic layout is done: the content itself has yet to be arranged. Many pages require several short content sections next to each other - though we are not speaking of tabular data. And of course a traditional column layout does not always meet the demands of today's design: the YAML homepage itself (www.yaml.de) exemplifies a much freer use of content blocks.

For these purposes, YAML offers *subtemplates*. These are XHTML code snippets which allow a horizontal division of content within various containers. These components are based on nested floating DIV boxes.

Note: all the required CSS definitions for the subtemplates are found in the file *base.css*. The adjustments for the correct automatic clearing in Internet Explorer are in the file *iehacks.css*. Subtemplates are integrated in the basic components of the framework and are available to all YAML layout variations.

Subtemplates can also be nested within each other. This allows you to vary the column divisions in countless various ways.

4.5.1 Structural Composition

The structure of such a code snippet is easy to understand with an example. Below is the required XHTML code for a 50/50 split - a division into left and right blocks of equal size.

```
...
<!-- Subtemplate: 2 columns with 50/50 division -->
<div class="subcolumns">
  <div class="c50l">
    <div class="subcl">
      <!-- left content block -->
      ...
    </div>
  </div>

  <div class="c50r">
    <div class="subcr">
      <!-- right content block -->
      ...
    </div>
  </div>
</div>
...
```

You get the general idea: let's now take a look at the details.

The Container
A *subtemplate* always begins with a DIV container of the `.subcolumns` class, which encompasses the smaller individual containers that actually divide the space.

```
<!-- Subtemplate: 2 columns with 50/50 division -->
<div class="subcolumns">
...
</div>
```

The file *base.css* assigns the class `.subcolumns` the following CSS properties, which should not be changed.

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

.subcolumns { display:table; width:100%; table-layout:fixed; }
.subcolumns_oldgecko { width: 100%; float:left; }
```

The container width is set as 100 percent by default, so that it completely fills the available horizontal space. Simultaneously, this definition activates the property *hasLayout* in Internet Explorer, forcing it to encompass the content within. All other browsers need the CSS property `overflow:hidden` (see [Section 2.3: Markup-Free Clearing](#)).

> **Note:** Netscape browsers up to and including Version 7.1 as well as old Gecko browsers (up to about July 2004) have problems with the display of the subtemplates due to a bug in connection with `overflow:hidden`, which was used up to YAML 3.2.x for containing floats.
>
> If support of these older Gecko-based browsers is required, you can use the class `.subtemplates_oldgecko` instead. Please note the information in [Section 5.3](#) on the Netscape *overflow-Bug*.

Dividing Space with DIV Blocks

DIV blocks with the CSS classes `c50l` and `c50r` divide the horizontal space. The "*c*" stands for *column*, the number *"50"* for *50 percent of the available width* and the letters "*l*" and "*r*" for *left-* and *right*-floating blocks.

```
<!-- Subtemplate: 2 columns with 50/50 division -->
<div class="subcolumns">
  <div class="c50l">
  ...
  </div>
  <div class="c50r">
  ...
  </div>
</div>
```

In general, two blocks (a left and a right) form a pair. The sum of the widths of all blocks within a subtemplate *should* always equal 100%. The following division ratios are provided for as part of YAML's predefined CSS classes, e.g.:

- 50% / 50% Division (classes `c50l` and `c50r`)
- 33% / 66% Division (classes `c33l` and `c66r` as well as `c66l` and `c33r`)
- 25% / 75% Division (classes `c25l` and `c75r` as well as `c75l` and `c25r`)
- Golden Ratio (classes `c38l` and `c62r` as well as `c62l` and `c38r`)

The class definitions are in the file *base.css*.

```
.c20l, .c25l, .c33l, .c40l, .c38l, .c50l,
.c60l, .c62l, .c66l, .c75l, .c80l {float: left; }

.c20r, .c25r, .c33r, .c40r, .c38r, .c50r,
.c60r, .c66r, .c62r, .c75r, .c80r {float: right; margin-left: -5px; }

.c20l, .c20r { width: 20%; }
.c40l, .c40r { width: 40%; }
.c60l, .c60r { width: 60%; }
.c80l, .c80r { width: 80%; }
.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%; }
.c62l, .c62r { width: 61.8%; }
```

The real width of a floating container is calculated by the browser just before it is rendered. The percentage values' conversion into pixels requires rounding. and Internet Explorer is sometimes less accurate than other browsers relating to the total width of all DIV blocks within a subtemplate.

The result is that the sum of the individual containers is greater than the width of the parent container `.subcolumns`, and the floating DIV boxes are displaced. To avoid this effect, all right-floating DIV blocks are assigned a `margin-left: -5px`. This allows any right-floating container to overlap an element to its left by a maximum of five pixels: an elegant compensation for the rounding errors.

Important: the compensation for these rounding errors demands **exactly one** right-floating container within each subtemplate.

These predefined CSS classes allow the following arrangements, even without nesting the subtemplates:

- 80% - 20%
- 75% - 25%
- 66% - 33%
- 62% - 38%
- 60% - 40%
- 50% - 50%
- 40% - 60%
- 38% - 62%
- 33% - 66%
- 25% - 75%
- 20% - 80%
- 33% - 33% - 33%

These blocks can be nested deep within each other simply by inserting further subtemplates. This allows nearly absolute freedom in the division of your columns.

The Content Containers

As in the layout columns of the YAML framework, the outer containers (here the DIV blocks `c50l` and `c50r`) set up the division of space, while the inner containers `subc`, `subcl` and `subcr` maintain the `padding`, `margin`, and `border` around the content.

```html
<div class="subcolumns">
  <div class="c50l">
    <div class="subcl">
      <!-- left content block -->
      ...
    </div>
  </div>

  <div class="c50r">
    <div class="subcr">
      <!-- right content block -->
      ...
    </div>
  </div>
</div>
```

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

The final letters "*l*" and "*r*" stand for content blocks on the left or right side of the subtemplate. This influences the `padding`). For content blocks which are not on the side, such as the middle block of a 33/33/33 division, we have the container `subc`, which has padding on both sides.

The sum of the assigned paddings must always be identical for each block to guarantee that each column has exactly the same width.

The containers `subcl` and `subcr` on the sides are each assigned a padding of 1 em on the inner side. The container `subc` needs padding on both sides, which must total 1 em: it is assigned left and right paddings of 0.5 em each.

4.5.2 Adjusting the Subtemplates for Internet Explorer

The subtemplates use the CSS property float rather extensively. This means that we must deal with the corresponding IE bugs such as the Escaping Floats Bug and the Doubled-Float-Margin Bug -- and of course the Expanding Box Problem turns up when we work with flexible container widths.

Analogous to the bugfixes for the YAML basic layout, the bugfixes are also applied to the subcolumns.

```css
.c20l, .c25l, .c33l, .c38l, .c40l, .c50l, .c60l,
.c62l, .c66l, .c75l, .c80l, .c20r, .c25r, .c33r,
.c38r, .c40r, .c50r, .c60r, .c66r, .c62r, .c75r, .c80r {
   display:inline;
}
```

```
/* avoid growing widths */
* html .subc,
* html .subcl,
* html .subcr { word-wrap:break-word; o\verflow:hidden; }
```

The Escaping Floats Bug is taken care of when the container `.subcolumns` is provided with *hasLayout* via `width:100%`. The property `display:inline` defuses the Doubled-Float-Margin Bug, and `word-wrap:break-word` and `overflow:hidden` ensure that even the older IE generations (IE 5.x and IE 6) cut off oversized content elements and do not let them destroy the layout.

Note: the file *iehacks.css* sets the property `word-wrap` back to the standard value of `word-wrap:normal` for printing.

4.5.3 Examples for Subtemplates Use

The following examples show subtemplates used directly as well as nested. Read the source code of the examples in the online documentation carefully to understand exactly what's happening.

50 / 50 Split

Block 1: Lorem ipsum dolor sit amet, consectetuer 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, consectetuer 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 Split

Block 1: Lorem ipsum dolor sit amet, consectetuer 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, consectetuer 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, consectetuer 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. ...

Divisions According to the Golden Ratio

Block 1: Lorem ipsum dolor sit amet, consectetuer 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, consectetuer 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 ...

Endless Variety with Nesting

Subtemplates can be endlessly nested within each other, allowing you countless various column divisions. The only requirement is that within each nesting level, the sum of the blocks' width must always be 100%. The following example shows such a nesting. Within the left *50 percent* block are two further *50 percent* blocks. The right *50 percent* block is further divided according to the Golden Ratio.

Block 1: Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In ac lectus. Aenean tincidunt metus nec orci. Nulla dapibus mattis tellus.

Block 2: Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In ac lectus. Aenean tincidunt metus nec orci.

Block 3: Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In ac lectus ...

Block 4: Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In ac lectus. Aenean tincidunt metus nec orci. Nulla dapibus mattis tellus. ...

4.5.4 Special Case: Equal Height Boxes

Starting with version 3.1, YAML natively supports the creation of content containers of equal height - independent of their content - with pure CSS. The concept behind this bases on the markup of the subtemplates, but combined with several CSS layout concepts in order to ensure correct crossbrowser display.

Technical Background

In modern browsers with complete CSS2 support, the subtemplates are transformed into CSS tables. In Internet Explorer 5.x - 7.0 we must still use floating containers, extended with a CSS hack to simulate equally tall columns with the help of large paddings. This technique is explained at length by Alex Robinson in his "One True Layout" article.

The corresponding CSS rules for the combination of the various techniques are completely integrated in *base.css* and *iehacks.css* for easy access. Rules are tied to the CSS class equalize, which overwrites some of the properties of the normal subtemplates via the CSS cascade. Again, the technique is too complex to relate here.

In Practice

The creation of equal height boxes within YAML is very easy. You activate this property of the subtemplates merely by assigning the surrounding container the additional CSS class `equalize`. The characteristics are then handed down, so all nested subtemplates inherit the equal height.

```
<!-- Subtemplate: 2 columns with 50/50 division -->
<div class="subcolumns equalize">
...
</div>
```

By assigning the CSS class `.equalize`, the column containers `.cxxl` and `.cxxr` are forced to the same height. The content containers `.subcl`, `.subcr`, and `.subc` do not change.

The following sample demonstrates the possibilities of this technique with three flexible boxes, lined up next to each other, with a decorated frame.

/examples/06_layouts_advanced/equal_height_boxes.html

Important: as a result of the techniques used for Internet Explorer 5.x - 7.0, neither elements nor background images can be absolutely positioned at the bottom of the equal height boxes.

Should this positioning be necessary for layout reasons, use the CSS class `.no-ie-padding` to deactivate the expansion of the containers in Internet Explorer. In this case, you as layout designer must ensure the equal height of your boxes -- either with content or by explicitly assigning a specific height.

The layout sample "*equal height boxes*" demonstrates how this complicated-sounding workaround functions in practice, by positioning the "more" links. For more information, read the source code of the sample layout.

4.6 Column Design

In Section 2.7 we discussed the particulars of the special clearing at the end of the static column `#col3`.

Of course this does not deliver our ideal genuine equal-length columns, but YAML's method does bring us very close. How close exactly? Let's examine the following two examples.

> **Important:** The column `#col3` nearly always needs the value `width:auto;`! Otherwise Internet Explorer will apply the attribute *hasLayout = true* (see the article: on having Layout), destroying our careful IE clearing at the end of `#col3` by encompassing it.
>
> **Background:** the static column `#col3` is the *bearer* of the column separators. Without the clearing, the column and thus the lines would not always reach all the way down to the footer.

4.6.1 Example 1 - Column Separators

A common design element is vertical separation lines between the individual content columns. The *Faux Columns* technique (with background images) is often used to ensure that these lines are always the same length, independent of how full each column container is

When using the column order 1-3-2 and 2-3-1 (see Section 4.4), YAML can create lines without using any background images at all. Instead, we use the CSS `border` property of the static column `#col3`. This is possible in these column orders because #col3 is always the longest.

Below is an example of a two-pixel wide dotted line as a column separator for a three-column layout:
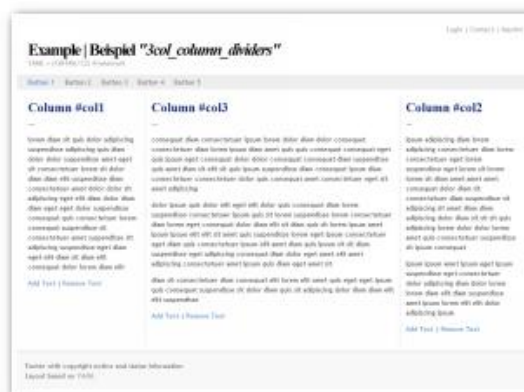
```
#col3 {
  border-left: 2px #ddd dotted;
  border-right: 2px #ddd dotted;
}
```

04_layouts_styling/3col_column_dividers.html

4.6.2 Example 2 - Column Backgrounds

As already mentioned, CSS cannot really construct a "genuine" column layout with floating elements, as the individual columns are always only as high as the contents require. So we need to use a little trick to put background images or colors on those columns. We have chosen the "*Faux Columns*" technique as it is the most reliable method for producing the desired effect.

We use the fact that all the column containers, whose height varies, are contained within a parent container like `#main`, which thus automatically expands to the height of the tallest column. So we leave the background of the column containers transparent and assign the desired background image to a container like `#main`. As CSS 2 can only assign one background image per container, the second column graphic needs a second container. The accompanying examples use `.page`. And to ensure that the background image from `.page` (which surrounds `#header` and `#footer`) is really only visible in the column area, `#header` and `#footer` are assigned solid white backgrounds.

The following example details the simple implementation of this technique. The necessary CSS definitions are in the file *basemod_faux_columns.css*.

04_layouts_styling/3col_faux_columns.html

Our trick is a bit more involved when using flexible layouts. Assuming you define your column widths in percent, we can still use the Faux Columns technique. It is slightly more complicated because you have to prepare a particular background image so that the CSS positioning works correctly. The article "*Faux Colums for liquid layouts*" details the technique extensively. The following example shows its use within a three-column layout with embellished side columns.

04_layouts_styling/3col_liquid_faux_columns.html

Note: in earlier YAML versions it was also possible to use the border property of the `#col3` container to create a solid column background for the side column. This method is no longer supported in YAML 3.2 and above, as it creates a particular accessibility problem when certain contrast modes are used with Windows.

4.7 Minimum & Maximum Widths for Flexible Layouts

Flexible layouts adjust themselves dynamically to the current width of the browser window. This behavior is normally quite useful, but is sometimes inconvenient. For example, an extremely narrow browser window can make the layout illegible and thus unusable. You should define a lower limit for your elements' width, perhaps oriented to a desktop resolution of 800x600 pixels, to guarantee a legible layout even at that size.

Just as important: an upper limit for the layout's width. If the layout is too wide, copytext appears in very long lines. In extreme cases, paragraphs of several lines become individual lines of text. This is very tiring for readers' eyes, as they must travel a long way before reaching the break at the edge of the page. Even these details can frustrate your site's readers.

Both scenarios can be easily avoided with the CSS properties `min-width` and `max-width`.

The YAML framework's basic layout should contain all the width definitions in the container `.page_margins`, as this encompasses and thus defines all the other elements.

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

This example defines a minimum layout width of *760 pixels*. This will work even on a desktop resolution of 800x600 pixels and allows the layout to display in the browser's full-picture mode without creating horizontal scrollbars.

A maximum width is better defined according to the font size, in EM. A value in pixels would create problems when zooming on text, as the layout would not adjust itself for the larger letters. Unintentional line breaks and oddly-placed pictures would result. Basing the layout width on the font size itself easily eliminates this problem: the example below shows a value of 100em.

4.7.1 CSS Support Lacking in Internet Explorer 5.x and 6.0

Again, Internet Explorer makes life harder for us web designers: IE up to and including version 6.0 supports neither `min-width` nor `max-width`. Only with Internet Explorer 7.0 did Microsoft finally add these properties, as well as the additional `min-height` and `max-height`. This novum as well as the fixed CSS bugs and the greater surfing security is a blessing for web programmers. One can only hope that IE7 spreads quickly.

And yet, the older IE versions cannot be ignored when creating a layout, as IE 6 still rules the browser statistics and will certainly not disappear so quickly, even though its market share is steadily eroded by IE7.

For Internet Explorer 5.x and 6.0, I have prepared two Javascript methods to mimic the `min-width` and `max-width` properties for these browsers.

4.7.2 Solution 1: IE Expressions

Internet Explorer allows the web page creator dynamic access to CSS properties with the proprietary property *expression()*. With help from Javascript, we can quite simply fake the missing CSS properties. Svend Tofte's article max-width in IE offers a general overview of the technique. However, the examples in that article demand *Quirks Mode* (see DocTypes & Display Modes in Section 2.4). Jeena Paradies invented a Code Variant, which also works in *Standard Mode* in IE and provides the basis for the solution discussed here.

Important: earlier YAML versions required Internet Explorer be set to *Quirks Mode* for this method: **no more**!

The JS expressions should be built into the patch files, so that only IE is forced to load the required code. Below is an excerpt, as used in the layout examples in the download package:

```
/**
 * min-width/max-width workaround for IE5.x & IE6
 *
 * @workaround
 * @affected   IE 5.x/Win, IE6
 * @css-for    IE 5.x/Win, IE6
 * @valid      no
 */

* html .page_margins {
  /* Fallback if JavaScript is disabled */
  width: 80em;

  /* JS-Expression for min-/max-width simulation */
  width: expression( ... );
}
```

Note 1: the doubled request for the current viewport size is indeed necessary, as IE 6.0 in S*tandard Mode* reaches `.clientWidth` in a different way than in IE 5.x, which is generally in *Quirks Mode*.

Note 2: You don't have to create the JS-Expression by hand for your individual layout. The YAML Builder will help you by creating all necessary code dynamically depending on your layout settings.

This example implements a minimum layout width of 740 pixels. The maximum width is based on the font size. The current font size of the body element must be determined and then compared with the value 80em.

The fallback solution for those surfing without Javascript is to define `width:80em` before the *expression()* appears in the code.

4.7.3 Solution 2: External Javascript "minmax.js"

The Javascript file *minmax.js* from doxdesk.com is included in the YAML download package, in the *js/javascript* folder. This file can be integrated into the web page's head and mimics the full functionality of the `min-width`, `max-width`, `min-height`, and `max-height` -- by evaluating the CSS definitions and adjusting IE's rendering accordingly.

This script, when linked via *Conditional Comments* (`<link href="css/patches/patch_3col.css" rel="stylesheet" type="text/css" />`), is only loaded by those browsers which need it: IE versions 5.x and 6.0. Internet Explorer 7 no longer requires this script, as it interprets the standard CSS properties.

```
<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="js/minmax.js"></script>
<![endif]-->
</head>
```

That was it -- no more work is necessary. Watch the effect of the Javascript on the test page *minmax_js.html*.

/examples/08_special_interest/minmax_js.html

Note: This Javascript does have a notable disadvantage: it is only loaded after the page has been fully rendered. That means that a too-small or too-large browser window will first show the page without the `min-width` or `max-width` adjustments, and will only adjust the layout after a few tenths of a second. The page will visibly "jump", and this can be rather annoying while surfing a website. Please test this effect before adding the script.

4.8 Selected Application Examples

The following three sections explain example layouts for specific demands, all created with YAML. The structure of the examples will help you understand the various ways to design the basic layout and how to manipulate the framework. All the samples contained in the download package *examples/* folder are based on a simple screen layout, explained below.

The Screen Layout of the Examples

The basis is a flexible three-column layout with the column order 1-3-2 (the standard order) and the columns divided into 25% | 50% | 25% of the screen. This layout is in the *examples/01_layouts_basics/* folder.

01_layouts_basics/layout_3col_standard.html

The minimum width is fixed at 740 pixels, orienting itself to a desktop resolution of 800x600 pixels, and allowing a display at that resolution without horizontal scrollbars. The maximum width of the layout is set at 80em, which in combination with the standard font size of 75% (16px*0,75=12px, set in *content.css*) results in a width of 960 pixels.

The screen layout is included via the CSS file *basemod.css*, which is found in every theme folder within the respective *css/screen/* folder. Below is a code excerpt:

```
/* (en) Marginal areas & page background */
body { background: #9999a0; padding: 10px 0; }


/* (en) Layout: width, background, borders */
.page_margins {
  min-width: 740px; max-width: 80em;
  margin: 0 auto; border: 1px #889 solid;
}

.page{ background: #fff; border: 1px #667 solid; }

/* (en) Centering layout in old IE-versions */
body { text-align: center }
.page_margins { text-align:left }


/* (en) Designing main layout elements */
#header {
  color: #fff;
  background: #000 url("…") repeat-x bottom left;
  padding: 45px 2em 1em 20px;
  position: relative;
}

#topnav {
  background:transparent;
  color:#aaa;

  position:absolute;
  top:10px;
  right:10px;
  text-align:right;
```

```
}

#main { background: #fff }

#footer {
  color:#fff;
  background: #336 url("…") repeat-x bottom left;
  padding: 15px;
}

/* (en) adjustment of main navigation */
#nav ul { margin-left: 20px; }
#nav_main {background-color: #336}

/**
 * (en) Formatting content 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; }
```

Note: The structure of the file is based on the template *basemod_draft.css* from the *yaml/screen/* folder, which was explained in .

The file *nav_shinybuttons.css* from the *yaml/navigation/* folder has been linked unchanged. The only adjustment was in the distance of the first menu item from the left edge of the layout (`#nav ul { margin-left: 20px }`).

Adjustments of the Screen Layout for Internet Explorer

The basic layout still needs two special adjustments for an error-free display in Internet Explorer 5.x and 6.0. The 3 Pixel Bug must be fixed and minimum and maximum layout widths set. The *JS-expressions* used are explained in .

The adjustments for Internet Explorer are kept in the patch file *patch_3col_standard.css*, corresponding to the basic layout, in the *css/patches/* folder.

```
/* IE 3 Pixel Bug | 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( ... );
}
```

That finishes the most basic version of the screen layout.

4.8.1 Draft Layout "2col_advanced"

This first layout draft with the name **2col_advanced** meets the following requirements:

- Two-column layout (navigation left in `#col3` and main content right in `#col1`)
- Flexible layout with flexible column widths (25% | 75%)
- Further division of the main content area in two columns after the first paragraph
- Vertical 1 pixel wide separator between the columns with a vertical spacing of 1em to both header and footer.
- Horizontal main navigation "Shiny Buttons"
- Print layout: only the main content from `#col1`.
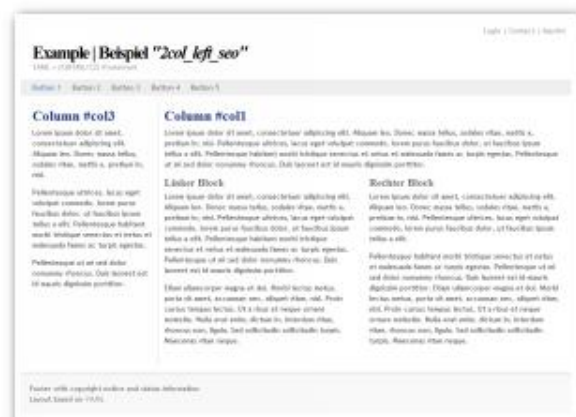
/examples/06_layouts_advanced/2col_advanced.html

Layout Draft in Detail

The central stylesheet *layout_2col_advanced.css* contains the following CSS components:

```
/* import core styles | Basis-Stylesheets einbinden*/
@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/basemod_2col_left_seo.css);
@import url(screen/content.css);

/* import print layout | Druck-Layout einbinden */
@import url(../../../print/print_001.css);
```

First, the base stylesheet *base.css* from the *yaml/core/* folder is loaded, as well as the unmodified navigation *nav_shinybuttons.css*.

Then, the basic version of the screen layout *basemod.css* is imported, which forms the basis of the layout. The modifications for the requirements of the desired two-column layout are found in the *basemod_2col_advanced.css* file.

```
/* #col1 becomes the main content column */
#col1 { width: 75%; float:right}
#col1_content { padding: 10px 20px 10px 10px; }

/* hide #col2 | #col2 abschalten */
#col2 { display:none; }

/* #col3 becomes the left column */
#col3 { margin-left: 0; margin-right: 75%; }
#col3_content { padding: 10px 10px 10px 20px; }


/* graphic-free column separators between #col1 and #col3 */
#col3 {border-right: 1px #ddd solid;}
#main {padding: 1em 0}
```

2 Columns: with the first declaration, `#col1` receives 75 percent of the available width and is *floated* to the right, becoming the main content column. The container `#col2` is not needed and is hidden. Finally, #col3 is moved to the left side by adjusting its margins.

Column separators: in addition, this example uses a 1 pixel wide dotted line as a vertical column separator. This is created by using the CSS `border` property for the static `#col3`. The top and bottom margins of the `#main` container determine the spacing of the line from the header and footer.

Adjustments for Internet Explorer
The adjustments for Internet Explorer are collected in the file *patch_2col_advanced.css* in the *css/patches* folder. As the graphic-free column separators are used, the 3 Pixel Bug cannot be fixed in this layout.

```
/* LAYOUT-INDEPENDENT ADJUSTMENTS --------------------------- */
@import url(../../../../yaml/core/iehacks.css);

/* LAYOUT-DEPENDENT ADJUSTMENTS ----------------------------- */
@media screen, projection
{
/* min-width / max-width for IE

* html .page_margins {
   width: 80em;
   width: expression( ... );

}
```

First, the stylesheet integrates the global adjustment file *iehacks.css* from the *yaml/core/* folder. (Do not be distracted by the relative paths in this example - they are only due to the folder structure of the sample folder.)

Next, we incorporate the IE expressions to simulate `min-width` and `max-width` in IE 5.x and 6.

Note: If you look at this example in IE5.01, you will notice that some paddings collapse. The corrections are not demonstrated in this example, as they are not necessary for understanding YAML.

4.8.2 Layout Draft "3col_advanced"

In this layout draft named **3col_advanced**, we meet the following challenges:

- Three-column layout (column order 2-1-3)
- Total width 960 pixels (240 | 480 | 240 columns)
- Further subdivision of the main content area in two columns after the first paragraph
- Column background left: background image with the "Faux Columns" technique.
- Horizontal main navigation "Shiny Buttons"
- Print layout: only the main content from `#col1`
- Basis for the screen layout is the three-column basic layout of the YAML framework

/examples/06_layouts_advanced/3col_advanced.html

Layout Draft in Detail

The central stylesheet *layout_3col_advanced.css* contains the following CSS components:

```
/* import core styles | Basis-Stylesheets einbinden*/
@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/basemod_3col_fixed_seo.css);
@import url(screen/content.css);

/* import print layout | Druck-Layout einbinden */
@import url(../../../print/print_001_draft.css);
```

First, we link the basic stylesheet *base.css* from the *yaml/core/* folder as well as the unmodified navigation *nav_shinybuttons.css*.

Then we import the basic version of the screen layout *basemod.css*, which forms the basis of the layout. The modifications for our current requirements are in the file *basemod_3col_advanced.css*.

Fixed Width and Centered Layout: the fixed layout width of 960 pixels is set at the outermost container `.page_margins`. Setting the side margins to `auto` can then center this container. The minimum and maximum widths are turned off, as they are useless in a fixed layout.

```
/* Setting the layout width and centering | Festlegung der Layoutbreite und
Zentrierung*/
.page_margins {
    width: 960px;
    min-width:inherit;
    max-width:none
}
```

Column order 2-1-3: I described the technique for resorting the columns in <u>Section 4.4: Variable</u> <u>Column Order</u>. Now it will be used to arrange the content within the source code according to its relevance.

```
/* #col1 becomes the middle column | #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 becomes the left colum |#col2 wird zur linken Spalte */
#col2 { width: 240px; float:left;  margin-left: -720px; }
#col2_content {padding-left: 20px; padding-right: 10px}

/* #col3 becomes the right column | #col3 wird zur rechten Spalte */
#col3 { margin-left: -5px; margin-right: 0; width: 240px; float:right; }
#col3_content {padding-left: 10px; padding-right: 20px}
```

Note the declaration of #col3: it is now floated. With this trick, we can completely avoid the IE 5.x and IE 6 3 Pixel Bug. The web designer's freedom is not at all limited by this step, as the column order 2-1-3 is inherently only compatible with pure pixel- or percent-based layouts -- see <u>Section 4.4</u>.

Faux Columns Background: the floated #col2 on the left side needs a continuous column background. The Faux Columns Technique is perfect. The container #main is assigned the graphic as a left-aligned and vertically-repeating background image.

```
/* Background image for the left column - width 240 pixels |
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;
}
```

Now the layout is complete. Only Internet Explorer left to manage.

Adjustments for Internet Explorer

The adjustments for Internet Explorer are collected in the file *iehacks_3col_advanced.css* in the *css/patches* folder. In the first step, the global adjustment file *iehacks.css* is linked.

```
/* Layout-independent adjustments ------------------- */
@import url(../../../../yaml/core/iehacks.css);



/* Layout-dependent adjustments -------------------- */
```

```
@media screen, projection
{
/* No layout-dependent adjustments necessary */
}
```

Thanks to the care taken with *iehacks.css*, even in this relatively complex layout, no further adjustments are necessary.

Alternative Solution for Centering Fixed Layouts (IE5.x Compatible)

The centering method used in this layout draft works in all modern browsers, no matter if with a fixed or a flexible layout. Internet Explorer 5.x, an exception, will display the layout on the left.

For fixed layouts, there is alternative method for centering that will also work in the outdated Internet Explorer 5.x.

```
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; }
```

Note: the web page is centered here with a negative margin. This variant is accordingly incompatible with flexible layouts.

4.8.3 Layout Draft "Flexible Grids"

A "normal" column layout cannot always meet all the demands of current website design. More flexible systems are necessary to divide a web page into smaller units. The term "grids" has become common, as the units are often oriented to a specific matrix of rulers and spacing.

YAML can simply and easily adapt to this concept using subtemplates. They allow space to be divided according to percentages and can simultaneously be nested within each other. The layout example "**flexible_grids**" demonstrates some of the possibilities of such grid-based layouts.
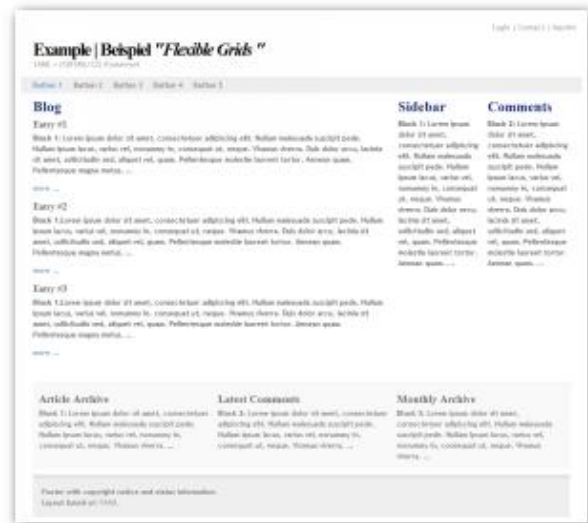
/examples/06_layouts_advanced/flexible_grids.html

Layout Draft in Detail

The central stylesheet *layout_grids.css* contains the following CSS components:

```
/* import core styles | Basis-Stylesheets einbinden*/
@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(../../../core/print_base.css);
```

First, we link the basic stylesheet *base.css* from the *yaml/core/* folder as well as the unmodified navigation *nav_shinybuttons.css*.

Then we import the basic version of the screen layout *basemod.css*, which forms the basis of the layout. For the print layout, we need to link the unchanged *print_draft.css* from the *yaml/print/* folder.

Important: subtemplates are not generally linearized for print, as their use is too variegated to predict. The page creator must regulate any desired linearization individually.

Implementation of the Grid Concept

This draft is implemented by adding the necessary subtemplate containers to the HTML source code. As you can see in the screenshot, the content columns of the basic layout are obviously no longer necessary.

The label "subtemplate" actually implies that these components are meant to be inserted into the content columns. That was indeed the original goal. Yet the nesting possibilities make them particularly interesting for layout development.

Correspondingly, I break here with the standard source code structure and completely replace the content columns `#col1` to `#col3`.

Structure of the Upper 66/33 Block

First, we must ensure that the upper and lower blocks are properly aligned with each other. The upper block uses a division of 66% | 33%, while the lower block divides into 33% | 33% | 33%. The right containers of both blocks duly reach the same width. In the second step, the 33% container of the upper block is divided again with a further subtemplate into two equal areas. The content containers are inserted so that they take up the same vertical space in the upper as in the lower block.
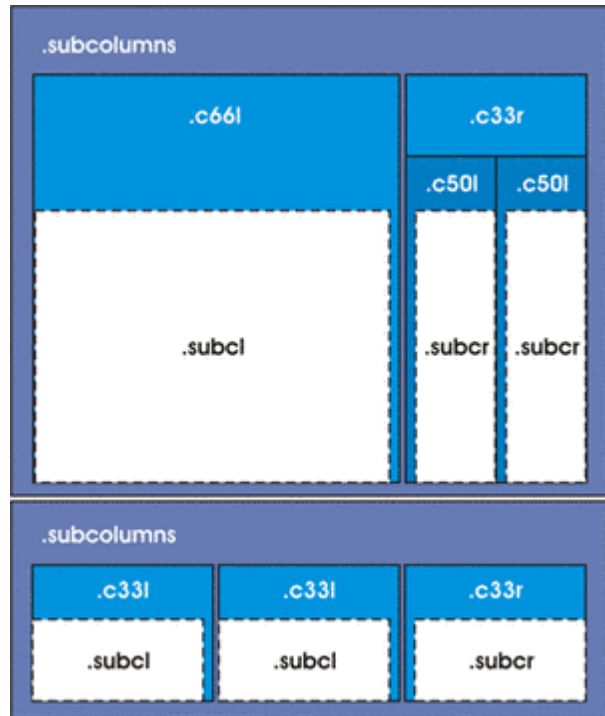
Structure of the Lower 33/33/33 Block

This is a simple subtemplate divided into 33% | 33% | 33%. The only peculiarity compared to the standard structure is that the center content block (`.subcl`) must be floated left. The reason is simple: the text within should end flush with that of the 66% container above it. If the block were centered, the margins would be different in the upper and lower blocks.

```
<!-- #main: Beginning of Content Area | Beginn Inhaltsbereich -->
<div id="main">
  <a id="content" name="content"></a><!--Skiplink:Content -->

  <!-- Subtemplate: 2 Columns with 66/33 Division | 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 class="c50r">
          <div class="subcr">
            <h2>Advertisement</h2>
            ...
          </div>
        </div>
      </div>
    </div>
  </div>

  <!-- Subtemplate: 3 Columns with 33/33/33 Division | 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 -->
```

The number of required DIV containers for this layout is naturally relatively large. Nevertheless, it is based completely on flexible widths and adjusts itself optimally to all screen proportions. The complete width is again assigned to the container `.page_margins`. The spatial divisions within `#main` are automatically adjusted by the subtemplates themselves. Certainly we could simplify the DIV constructions in this layout by using fixed widths -- but only then.

Adjustments for Internet Explorer
The adjustments for Internet Explorer are collected in the file *patch_grids.css* in the *css/patches* folder.

```
/* LAYOUT-INDEPENDENT ADJUSTMENTS ---------------------------- */
@import url(../../../../yaml/core/iehacks.css);

/* LAYOUT-DEPENDENT ADJUSTMENTS ------------------------------ */
@media screen, projection
{
/* No layout-dependent adjustments necessary */
}
```

Special adjustments for Internet Explorer are not required in this case, as the subtemplates are a fixed part of the framework. The adjustments are already all built into the file *iehacks.css*.

5 Tools & Tips

5.1 Tools

In addition to the YAML framework itself and the sample layouts, the download package provides you with further tools in the *tools/* folder to make your work even easier.

5.1.1 Dynamically Generated Dummy Text

js/ftod.js

This compact Javascript tool generates dummy text (Lorem ipsum...) within any DIV container. Two dynamically added text links can add or remove the paragraphs.

The script is used in various layout samples in the *examples/* folder of the download package. The use of the script is quite simple. Link it in the web page header:

```
<script type="text/javascript" src="your_path/ftod.js"> </script>
```

Immediately below, we configure the tool to define which areas of the page should be filled with the dummy text. These HTML elements need unique IDs so the script can find them.

In the layout samples, these are the containers `#col1_content` to `#col3_content` of the three content columns of the basic layout.

```
<script type="text/javascript">
window.onload=function(){ AddFillerLink( "col1_content", "col2_content",
"col3_content"); }
</script>
```

5.1.2 Dreamweaver Styles

Dreamweaver is one of the most popular software tools for professional web design on the market. However, it still has problems up to Version 7 with the WYSIWYG display of YAML-based layouts.

Dreamweaver MX 2004 (V7.0)

tools/dreamweaver_7/base_dw7.css

Conveniently, *Dreamweaver* does provide for alternative stylesheets just for drafting new pages in WYSIWYG mode. These stylesheets are only used in the editor, and can compensate for Dreamweaver's difficulty in displaying sophisticated CSS layouts.

For Dreamweaver MX 2004, you will find an alternative base stylesheet *base_dw7.css* in the *tools/dreamweaver_7/* folder, which comments out the problematic declarations. Instructions for use with Dreamweaver are included in the file *readme.txt*.

Dreamweaver 8

The current version has further improved the display quality in the editor's WYSIWYG mode, so that there's usually no need for any special adjustments of the CSS components. The only known problem currently is with the processing of the `@media` rules: Dreamweaver may overwrite the screen rules with those for print.

Should this happen, it is usually easiest to hide the print stylesheets from the editor. The file *readme.txt* in the *tools/dreamweaver_8/* folder describes the necessary steps.

5.2 Tips on Designing Flexible Layouts

In closing, a few more things to note when developing flexible layouts.

5.2.1 Dealing with Large Elements

It is important to fully understand the functioning of a column layout using *floats*. The static column `#col3` "flows around" the two *float* columns `#col1` and `#col2` (even if this is not obvious in the layout).

Background: Internet Explorer is the only browser that still has problems dealing with elements which are too wide for the static `#col3`. In this case, the entire `#col3` is shoved below the *float* columns -- or even hidden entirely. The layout is destroyed, and the web page is barely usable. Various solutions for this problem are available in [Section 3.5.2](#).

All other modern browsers allow the too-wide elements to merely overflow into the neighboring columns: the layout remains intact. Web designers must watch out for this problem, especially in flexible layouts, as even a minimum layout width must guarantee content enough space in its container.

5.2.2 Small Screens

Flexible layouts adjust themselves to the width available. The formatting (margins, sizes) of all content elements should orient themselves to a sensible minimum width.

An often-used lower limit for screen display is the SVGA resolution of 800x600 pixels. This resolution leaves a viewport of usable space of circa 760 pixels, as vertical scrollbars and even window borders themselves reduce the available space. This is important to note, as horizontal scrollbars should be avoided if at all possible.

All content elements (headings, tables, forms, graphics) should be created to fit into this minimum width, so that the layout is displayed without errors or overlapping.

For even smaller resolutions (like on PDAs and other mobile devices), a new stylesheet can be quite handy -- made accessible only via the CSS rule `@media handheld`. Linearized columns are better suited to tiny displays: the containers will then appear one after the other, just as in the print stylesheet.

5.2.3 Flexible Side Columns

The width of the static column `#col3` in flexible layouts normally results automatically from the total width of the browser window minus the widths of the displaying *float* columns. Should the *float* columns `#col1` or `#col2` also require flexible widths, they should be measured in *EM* or *percent*.

Yet when using *EM* for *float* columns, please note: the *float* columns always extend themselves toward the static column `#col3`. If the user zooms in on the text, `#col3` will eventually become too narrow to read, as the font size in each container increases, and the width of the *float* containers increases proportionally to the font size as it is oriented to *EM*. The float columns force `#col3` to become narrower and narrower as they expand.

As a consequence, I recommend percentage values for flexible *float* columns. The proportions will then remain constant, independent of font and window size.

5.3 Known Problems
5.3.1 Internet-Explorer 5.x: Collapsing Margin on #col3

	IE 5.x/Win	IE 5.x/Mac	IE 6	IE 7
Bug active	Yes	Unknown	Yes *)	Yes *)

*) This bug is actually present in these browser versions, but can be countered by the special IE clearing (see [Section 2.7: The Clearing of Column #col3](#)).

Description: The column `#col3` is defined with `width:auto`. Internet Explorer duly gives this container the property *hasLayout = false*.

In the event that in a three-column layout, the left column is the shortest while the right column is the longest, IE collapses the left margin of `#col3`.

This means that any border on `#col3` (graphic-free column separator!) between `#col1` and `#col3` slips over to the left side of the page. Any background defined for `#col3` will be stretched out to the left edge of the page. This widening has no influence on the actual content of the DIV (text, images, etc.), as `#col3` is set to be behind the side columns via the `z-index`. The bug can be observed on the following test page.

Testpage: [ie_bug.html](#) (only visible in IE5.x!)

Workaround 1: The visual effects of this bug can be avoided by using an image for the left-side column separator, and defining this as a background image for another container, like `#main`. Furthermore, `#col3` should have no background assigned, neither graphic nor color (see [Section 4.8: Draft Layout "3col_advanced"](#)). If required, these can be assigned to `#main` or `.page`.

Workaround 2: Alternatively, you can avoid the problem by activating *hasLayout = true* for `#col3` from within the adjustment file for Internet Explorer:

```
...
#col3 {height: 1%;}
...
```

This CSS hack is, however, incompatible with the graphic-free column separators. Should you choose this method, you must then use background images (Faux Columns technique) instead.

Note: YAML Version 2.5 eliminated this bug in IE 6 and IE 7. As IE 5.x is no longer very popular, this bug does not often cause problems -- especially as it does not actually hinder access to the web page.

5.3.2 Mozilla & Firefox

Mozilla browsers up to version 1.7.0 (and Firefox up to version 1.0) contained a Float Clearing Bug. This prevented the column separator of `#col3` from reaching all the way to the footer if one of the side columns were longer than the center column. This had no effect on graphics that were assigned as background images.

Bugfix: the bug was fixed in the July 2004 with version 1.7.1, and is no longer relevant.

5.3.3 Netscape

Netscape 6 & 7: the browser versions 6.x are based on unfinished beta versions of Mozilla and are extremely faulty. Although version 7 officially uses the rendering engine of Firefox 1.0.1 or 1.0.2, there are great CSS compatibility problems here too -- especially with the versions 7.0 and 7.1.

YAML officially supports Netscape version 8 and up. YAML-based layouts had no display errors in this version.

Netscape 7: Overflow Bug

The markup-free clearing using `overflow:hidden` causes display errors up to Netscape 7.1 when used on static boxes. This means that the content in subtemplates is hidden. The following workaround counters this bug.

Workaround: in general, it is enough to float the container in question. In this case, you must assure that the container occupies the complete available width to avoid bothersome side effects in your layout.

Should you use subtemplates and require support for these old Gecko engines, you can use the CSS class `.subcolumns_oldgecko` instead of .subcolumns. This alternative CSS class incorporates the float hack described above.

Note: if, when using `.subcolumns_oldgecko`, subsequent content is displayed next to the subtemplate rather than after it (so far only seen in tables), assign the property `display:inline` to those elements.

5.3.4 Opera

Opera 9.01 Bug: Opera's version 9.01 contains a hover bug, which collapses the margins between a clearing element and the next element. The current version, 9.02, has fixed this bug.

Workaround: instead of using margins, you can create whitespace with padding or borders. This avoids the problem entirely.

Opera 6: although Opera 6 in principle should be able to display YAML-based layouts correctly, certain conditions can lead to unpredictable phenomena such as unclickable areas, etc. These browser bugs cannot be countered with reasonable measures. Happily, the browser is no longer widespread enough to require our attention.

5.4 Add-ons
5.4.1 General

The functionality of the framework can be expanded with add-ons. These are complementary elements whose function is either entirely independent of YAML or only useful in specific situations.

Official add-ons are kept in the folder *yaml/add-ons/* and are available in the download package as well as in the project template *Simple Project*.

5.4.2 Add-on: Accessible Tabs & SyncHeight

Tabs have become extremely popular widgets on many websites, as many small blocks of information can be nicely sorted into limited space. JavaScript frameworks like jQuery can create the visual impression of tabs with only very few lines of code, and many other plugins are available with countless configurations for visual presentation.

The problem that even the best scripts have is that the user does not get feedback on activation. Most tab scripts change the presentation of the tabs and the visibility of their content, but leave the user where he was - at the tab he'd clicked - with no idea of what just happened.

Dirk Ginader acted on my suggestion and has recently developed the jQuery plugin "*Accessible Tabs*" and tested it thoroughly for accessibility. As the topic is so complicated, I ask you to read his own blog post (in English and German) on its functionality and configuration.

The second little plugin with the name *"SyncHeight"* serves to synchronize the height of any chosen containers via JavaScript. When using it in combination with the tab plugin, it becomes possible to orient the height of the visible tab to the maximum amount of content of all the tabs. The following example demonstrates the simple use of the Accessible Tabs as well as the optional synchronization of the tabs with the SyncHeight plugin.

/examples/09_add-ons/accessible_tabs.html

The required files (jQuery plugin, tab stylesheet) are in the folder *yaml/add-ons/accessible-tabs/*. The SyncHeight plugin is in the folder *yaml/add-ons/syncheight/*.

Further information on both plugins is on the following websites.

- Accessible Tabs mit jQuery (german)
- Accessible Tabs with jQuery (english)
- Plugin-Home: Accessible Tabs
- Plugin-Home: SyncHeight

5.4.3 Add-on: Microformats

Microformats are special tags to semantically structure the contents of a website: events, business cards, links, etc., so that they can be read and recognized by machines as well as humans. The content itself is not changed, only the markup.

Without special formatting, microformats are generally invisible for users without special settings -- which hinders their wide use and usefulness. It is better to make microformats immediately visible and recognizable to your visitors.

The folder *yaml/add-ons/microformats/* contains the stylesheet *microformats.css*, which contains standard formatting for the most often-used microformats.

The use of this add-on is demonstrated in the following layout sample:

/examples/09_add-ons/styled_microformats.html

Further information on microformats can be found on the following websites:

- microformats.org
- Microformats Wiki
- Mikroformate (german)
- mikroformate.org (german)
- The BigPicture on Microformats

5.4.4 Add-on: RTL Support

This add-on provides all the relevant components for working with YAML in Hebrew or Arabic languages whose text flows from right to left.

The plugin's files are in the folder *yaml/add-ons/rtl-support/*. Within that folder are the subfolders *core/* with the adjustments of the core elements of the framework: *base-rtl.css* and *iehacks-rtl.css*. The folder *navigation/* contains the adjustments for the standard included menus.

Basic Principles

The support of *right-to-left* languages comes from special stylesheets which overwrite existing CSS rules to control text direction and content positioning. These stylesheets thus only contain incremental changes and include the suffix **-rtl** in the filename: the adjustments necessary for the file *base.css* are found in the file *base-rtl.css*.

The activation of the changed text flow from right to left takes two steps. The first step is to change the text direction with the dir attribute of the HEAD element of the page: `<head dir="rtl">`. The second step is to merely link the RTL stylesheets into your layout. Within the *base-rtl.css*, the CSS property `direction: rtl;` is assigned again to the BODY element.

Along with the text direction, many other CSS properties must be changed -- margins, indentations, etc., to present the text correctly.

A sample of *base-rtl.css*:

```
ul, ol, dl { margin: 0 1em 1em 0; }
li {
  margin-left: 0;
  margin-right: 0.8em;
}
```

The corresponding section of *base.css*:

```
ul, ol, dl { margin: 0 0 1em 1em } /* LTR */
li {
  margin-left: 0.8em; /* LTR */
  line-height: 1.5em;
}
```

Properties within a regular stylesheet which are overwritten with other values for the RTL display are always marked with the comment /* LTR */.

Example

The use of the add-on is demonstrated in the following example:

/examples/09_add-ons/rtl_support.html

The corresopnding central stylesheet *layout_rtl_support.css* always links the normal YAML components first and then the RTL variation to overwrite them.

```
/* import core styles | Basis-Stylesheets einbinden */
@import url(../../../yaml/core/base.css);
@import url(../../../yaml/add-ons/rtl-support/core/base-rtl.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 rtl-support changes | RTL-Support Anpassungen einbinden */
@import url(../../../yaml/add-ons/rtl-support/navigation/nav_shinybuttons-
rtl.css);
@import url(screen/basemod-rtl.css);
@import url(screen/content-rtl.css);

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

Note: YAML is a tool for developing website layouts. The modular CSS structure as well as the support for RTL languages via complementary stylesheets with only the necessary changes is based on this concept.

When the layout development of any particular site is finished, the CSS components should be optimized for production: comments removed and files combined to minimize the number of HTTP requests.

6 Changelog

6.1 Changes in 3.x
6.1.1 Changes in Version 3.3.1 [18.06.11]

YAML Core

Bugfixes

- *yaml-focusfix.js* - fixed a small bug in classname detection
- *base.css* and *base-rtl.css* - skiplink CSS produced a huge horizontal scrollbar in FF4
- *markup.html* - corrected wrong filename to include "yaml-focusfix.js"

General

Updates

- Updated jQuery Library to current stable release 1.6.1
- Updated "Accessible Tabs" add-on to current stable release 1.9.1

6.1.2 Changes in Version 3.3 [12.10.10]

YAML Core

New

- *base.css* - Support for HTML5 Elements: set block-model to allow structural formatting (`article, aside, canvas, details, figcaption, figure, footer, header, hgroup, menu, nav, section, summary`)

Improvements & Feature Changes

- *base.css* - removed last remaining positioning properties for `#nav, #main, #header` and `#topnav`
- *base.css* - changed containing floats solution for `.floatbox` to contain floats without clipping: `display:table; width: 100%;`
- *base.css* - changed containing floats solution for subtemplates to: `display:table; width: 100%;`
- *base.css* - moved non-general parts of the italics fix to `iehacks.css` to save some bytes in modern browsers.
- *iehacks.css* - trigger *hasLayout* for subcolumns content boxes in print layout (containing floats).
- *yaml-focusfix.js* - rewritten JS code (thanks to Mathias 'molily' Schäfer for contribution)

General

Improvements

- *forms.css* - changed clearing for form rows (`.type-text` etc.) and fieldsets to *Clearfix* to ease CSS3 form styling

- *content_default.css* - new default values for `<sub>` and `<sup>`
- removed `@charset` rule from all stylesheets to avoid problems with several CSS minimizer tools.

6.1.3 Changes in Version 3.2.1 [10.01.10]

YAML Core

Improvements

- *yaml-focusfix.js* - The JS file *webkit-focusfix.js* (introduced in v3.2) was renamed due to the extended support for more browsers
- *yaml-focusfix.js* - no more pollution of global namespace
- *yaml-focusfix.js* - added IE8 support and fallback solution for older webkit browsers
- *iehacks.css* - simplified Clearfix adjustments for IE 5.x - 7.0

General

Improvements

- *content_default.css* - better contrast on `a:focus {}` (keyboard acessibility)
- *forms.css* - improved robustness for "columnar" and "full" form layout (avoiding float drops)

Bugfixes

- *forms.css* - included fix for IE7 auto-padding bug when using buttons

Examples

Improvements

- *equal_height_boxes.html* - better accessibility for complex example (hidden more links within content boxes)

Bugfixes

- *flexible_grids2.html* - added new skiplink styling to its *basemod_grids2.css*
- *equal_height_boxes.html* - added new skiplink styling to its *basemod_equal_heights.css*

6.1.4 Changes in Version 3.2 [27.10.09]

YAML-Core

New

- *base.css* - merged *base.css* and *print_base.css* (smaller filesize)
- *base.css* - New subtemplate-set (20%, 40%, 60% and 80%), equalized mode is available
- *base.css* - new skip link solution, that allows overlaying
- *js/webkit-focusfix.js* - JavaScript based fix for focus problems in Webkit-browsers (Safari, Google Chrome)

Improvements

- *base.css* - Split up media types to "all", "screen, projection" and "print", helps to avoid several problems in print layout.
- *base.css* - using child selectors for equalize-definition saved about 400 bytes of code
- *base.css* - moved visual print settings (fontsize & hidden containers) to print-stylesheets
- *iehacks.css* - improved code for robustness of all major layout elements
- slightly better optimized slim-versions of core-files

Bugfixes

- *base.css* - removed `<dfn>` from the hidden elements again
- *iehacks.css* - fixed a bug that made subtemplates invisible in IE 5.01
- *slim_iehacks.css* - Clearfix hack was broken in IE7

Dropped Features

- *base.css* - removed code to force vertical scrollbars in FF, Safari & Opera (replaced by CSS3 solution in user files)
- *iehacks.css* - removed compatibility code for `#page_margins` and `#page` IDs.
- *iehacks.css* - Column Backgrounds with `#col3` border-styling isn't available anymore, due to accessibility and maintenance issues in IE

General

New

- *forms.css* - added `.full` class as an option to get full width for `<input>`, `<select>` and `<textarea>` elements in small columns or within subtemplates.
- *content_default.css* - added styles for `<big>`, `<small>`, `<dfn>` and `<tt>`

Improvements

- *forms.css* - `.yform` class can be added to any element. It's not bundled with `<form>` element anymore.
- *forms.css* - ajdusted fieldset- & div-paddings to avoid clipping of element outlines and dropshadows in Safari.
- *forms.css* - cleaner and easier fix for fieldset/legend problem in all IE's (including IE8).
- *forms.css* - Formatting for "reset" and "submit" buttons changed from IDs to classes to allow multiple forms on a webpage. Styling available via `input[type=reset]` or `input.reset` to older support IE versions (IE5.x & IE6).
- *content_default.css* - added a fix to `<sub>`, `<sub>` to prevent the visual increase of line-height.
- *nav_slidingdoor.css* - Removed predefined indent margin of 50px. Indention has to be set by the user in *basemod.css*
- *nav_shinybuttons.css* - Removed predefined indent padding of 50px. Indention has to be set by the user in *basemod.css*
- *tools/dreamweaver_7/base.css* - updated to recent codebase.
- updated jQuery library to version 1.3.2

Bugfixes

- *forms.css* - corrected issue in Firefox 1.x & 2.x where form labels weren't shown correctly in columnar display (FF-Bug)
- *forms.css* - no more jumping checkboxes & radiobuttons in IE8 and Opera
- *basemod_draft.css* - changed predefined selectors `#page_margins` and `#page` into `.page_margins` and `.page`
- *content_default.css* - nested lists (`ol`, `ul`) will be displayed correctly now.
- *markup_draft.html* - moved charset metatag in front of title element to allow UTF-8 there

Dropped Features

- *debug.css* - removed debug-stylesheet from *yaml/* folder. This feature is replaced by YAML's new [Debug Application](#)

Add-ons

New

- Accessible-Tabs plugin for jQuery
- SyncHeight plugin for jQuery

Improvements

- Microformats - added missing icons: xfn-child.png, xfn-parent.png, xfn-small.png and xfn-spouse.png
- RTL-Support - [*iehacks-rtl.css*] added an option to force the vertical scrollbar to the right in Internet Explorer (disabled by default)

Examples

New

- All examples - added WAI-ARIA landmark roles for accessibility improvement
- All examples - added optional CSS3-based code to force vertical scrollbars (overflow-y)
- *multicolumnar_forms.html* - demonstrates two easy ways to create flexible multicolumnar forms
- *accessible_tabs.html* - example for the usage of the Accessible-Tabs and the SyncHeight add-on
- *3col_liquid_faux_columns.html* - demonstrates "Liquid Faux Columns" technique

Improvements

- *3col_gfxborder.html* - changed ID's to classes to allow multiple usage
- *building_forms.html* - JavaScript Detection added
- *equal_height_boxes.html* - added a second usage example (simple)
- *dynamic_layout_switching.html* - added JavaScript detection code and option to "*show all columns*"

Removed

- *3col_column_backgrounds.html* - this feature isn't supported anymore due to accessibility issues

Bugfixes

- *2col_right_13.html* - corrected fix for 3-pixel-bug in IE-patch file
- *dynamic_layout_switching.html* - corrected fix for 3-pixel-bug in IE-patch file
- *equal_height_boxes.html* - still used `#page_margins` and `#page` ID's.
- *index.html* - link to last example corrected
- several CSS files were still not saved in UTF-8
- UTF-8 BOM signature removed from some files in *examples/04_layout_styling/*

6.1.5 Changes in Version 3.1 [20.01.09]

New Functions and Extensions

- **Form Construction Kit**
  YAML now includes a complete building kit of HTML and CSS building blocks for building forms. For more information, see [Section 3.10](#).
- **Support for content boxes of the same height**
  YAML's subtemplates can now produce boxes of the same height - that of the container with the most content. All with CSS. For more information, see [Section 4.5](#).
- **A better standard template for designing content**
  The *content_default.css* has been completely reworked and expanded and now contains a nearly complete set of set formats for texts, tables, and images.
- **Add-on: support for microformats**
  This add-on provides visual emphasis for the most important microformats.
- **Add-on: support for RTL languages**
  This add-on provides complete support for working with right-to-left languages (Hebrew and Arabic languages). It includes the required adjustments of the core files and navigation elements.
- **Layout examples included**
  The number of included examples has been completely reworked and expanded. 7 new sample layouts demonstrate the use of the new functions.
- **Download package and Simple Project**
  The JavaScript library [jQuery](#) version 1.3 is now included in both packages in the folder *js/lib/*.

Changes and Corrections

(X)HTML Markup

- **IDs #page_margins and #page are now CSS classes**
  This change allows the classes `.page_margins` and `.page` to be used more than once per page, thus broadly expanding the design possibilities. The change has no influence on existing YAML layouts and their updates. The necessary IE adjustments for the IDs are still included.
- **Unification of the navigation elements**
  The markup of the included navigation blocks has been unified. All IDs were transformed into CSS classes and all classes named in a uniform manner.

Core Files [base.css]

- **Expansion of the reset block**
  The blockquote and the quote elements are now included in the reset block. Any browser-predefined quotation marks are removed in favor of a uniform presentation.
- **Bugfix for the unfortunate rendering of select elements in Firefox**
  The reset block has been expanded to include 1px padding for select elements.
- **Generic layout classes removed**
  The CSS classes `.hidecol1`, .hidecol2 and `.hideboth` must be adjusted for specific layouts, so they belong in the user CSS. The new layout example *"dynamic_layout_switching"* demonstrates their use.

- **Reworking of classes for hidden content**
  The declarations for width and height were removed from the classes `.skip`, `.hideme`, and `.print` so they could more easily be made visible. The `DFN` element was added to the hidden elements.

Core Files [iehacks.css]

- **[fix] Reworking of the z-index values for content columns**
  Now Internet Explorer 6 users can again mark content in `#col3` in all column variations.

Core Files [print_base.css]

- **Font family no longer set**
  The font specification "Times New Roman" has been removed from the print stylesheet. The print version now uses the same font as the screen layout. The switch to the **pt** (point) unit of measurement remains, the standard size is still set to 10pt.
- **Marking abbreviations and links**
  The CSS rules for visibly marking abbreviations and linked urls have been moved to the print stylesheet (*print_xyz_draft.css*) to be turned on or off.
- **Introducing the CSS class .noprint**
  Dynamic content elements or areas that are too difficult to select via CSS can now be hidden from the print version with the `.noprint` class.

Navigation [shiny_buttons.css]

- **[fix] Collapsing margin**
  The collapsing of the navigation's left margin in Internet Explorer 5.x and 6.0 was corrected.

Graphics

- **File size optimization**
  All included graphics files of the YAML folders and the layout examples were optimized for size and the majority converted to PNG images.
- **Included Photoshop templates**
  All Photoshop templates (for example, the gfxborder graphics) are now collected in one central folder: *examples/photoshop-files/*.

6.1.6   Changes in Version 3.0.6 [09.06.08]

Changes and Corrections

Core-Files

- **[fix] Missing .print definitions added**
  Elements with class `.print` were invisible in print layout. This is fixed now.
- **[fix] Opera & attribute selectors**
  Opera 9.2x seems to have a bug concerning the optimized attribut selectors (missing brackets) in *slim_print_base.css*. Although all definitions are within `@media print` this bug leads to rendering problems in screenlayout. This bug is fixed now.

- **[new] Improved debug-stylesheet**
  YAML's debug stylesheet now visualizes xhtml page structure, subtemplates and special markup for content elements.

6.1.7 Changes in Version 3.0.5 [24.05.08]

Changes and Corrections

- **Update of license conditions**
  For free use under Creative Commons License (CC-A 2.0) naming the author is no longer necessary. A back link to YAML homepage is sufficient. The license texts of commercial licenses (YAML-C) have been revised and clarified in relation to reproduction and redistribution.
- **Overall code clean-up**
  CSS scripting layout (white spaces within brackets and semicolon behind last properties) within YAML files was equalised.
- **Better optimised slim-files**
  All core files where optimised for even smaller slim-variants for production use.
- **@media rule for screen layouts changed**
  basemod- and IE-patch files contain all relevant CSS rules for the screen layout. The `@media` rules within these files have been changed to `screen, projection`. Therefore, the screen layout shouldn't have an impact on the print layout anymore.

Core-Files [base.css]

- **Positionierung of #topnav**
  Container `#topnav` gets property `position:absolute` only within #header. In any other case it's a static container with text-alignment to the right.
- **Subtemplates**
  CSS class `.subcolumns_oldgecko` doesn't get `overflow:hidden` anymore. Containing Floats is done via `float:left`.
- **z-index**
  Specific z-index values were be removed from *base.css* (more simple fix for IE Clearing bug in *iehacks.css*). This should help a lot to avoid problems with e.g. dropdown menus.

Core-Files [iehacks.css]

- **Bugfix for IE-Clearing-Bug using negative z-index**
  The IE-Clearing-Bug can't be fixed completely in IE 5.x. But now there is a more simple way to avoid problems in IE by using one single negative z-index for `#col3`, than defining positive values for all content columns.

Layout examples [Folder /examples/]

- **New "Special Interest" example: "3col_fullheight"**
  In this example the overall layout height will be expanded to the bottom of the viewport, if page content is'nt enough to fill the screen. This example is labeled as "special interest" as a concept proof for for experienced users only.
- **Improved JS-Expression for min-/max-width in IE 5.5 and IE6**
  The new JS-expression works without problems in Quirks Mode and EM based values will be calculated - as they should - from parent elements font size.

- **Small screen layoutadjustments**
Some minor changes were made in the screen layout (*basemod.css*) to bring it mor in line with the YAML Builder and the *Simple Project Example*.

6.1.8 Changes in Version 3.0.4 [27.11.07]

Changes and Corrections

Core-Files [base.css]

- **Better solution to force vertical scrollbars in Firefox**
The new solution forces vertical scrollbars with `html { height: 100%; margin-bottom: 1px; }` that leeds to only 1 Pixel overlap and is much more pleasing.
- **Changes in the reset-CSS block**
The `cite` element was removed from the block. The `blockquote` element doesn't get predefined properties `font-size` and `width` anymore. This can be done by the user within *content.css* file.
- **Container #header**
The container `#header` gets `clear:both` property. Now it is easily possible to switch position of `#header` and `#nav` in the markup without negative interactions of floating navigation elements on `#header`.
- **Generic classes for l ayout switching**
The naming of the classes `.hideleft` and `.hideright` wasn't semantic, as the source order of column containers is indepentend from their position in screen layout. Thererfore the classes are renamed to `.hidecol1` and `.hidecol2`, which makes them clearly assigned. The class `.hidenone` is obsolete and was removed.

Core-Files [iehacks.css]

- **Improved layout stability**
Better stability for flexible columns in IE5.x + IE6 by adding `#main {position:relative}`. This avoids wrong positioning of columns after resizing the browser window while using IE-Expressions.
- **Bugfix added for List-Numbering-Bug**
Affects all IE versions 5.01 - 7.0: If a list element of an ordered list gets a property that activates *hasLayout*, IE doesn't correctly assign numbers to the list items.

Screenlayout Draft [content_default.css]

- **Fix for Gecko problems and font-size reset of monospaced elements**
Elements using monospaced font (textarea, tt, pre, code) will get a standard font-size of 13px instead of 16px when resetting font-size. A bugfix was added.

Layout examples [Directory /examples/]

- **[3col_2-1-3]**
improved CSS design without activating IE/3-Pixel-Bug,
changed imported print-stylesheet to *print_100_draft.css.*
- **[3col_3-1-2]**
improved CSS design without activating IE/3-Pixel-Bug,
changed imported print-stylesheet to *print_100_draft.css.*

- **[3col_fixed_seo]**
  Bugfix for wrong `min-width` behavior of Safari 3,
  better CSS design without activating IE/3-Pixel-Bug
- **[3col_gfxborder]**
  Naming of edge graphics and -containers changed to be more consistent.
- **Namespace added to <html> element in all layout examples**
- **3-column-examples (03_3col_layouts)**
  Fixed lateral paddings of content containers within the columns according to their position
  on screen.

6.1.9 Changes in Version 3.0.3 [18.08.07]

Changes and Corrections

Core Files

- **[iehacks.css] Bugfix for input elements in IE6**
  The new bugfix for the Italics-Bug in V3.0 had a side effect, that input elements were
  arbitrary extended in IE6. A fix was added and *slim_iehacks.css* was updated.

6.1.10   Changes in Version 3.0.2 [01.08.07]

Download Package & Documentation
- [Doc en/de] some URL's corrected.
- [Doc de] section numbering corrected
- [Doc de] Section 1.4: futher links added
- [CSSDoc-Comments] beautified indenting of comments in css files

Änderungen und Korrekturen

Core Files

- **[base.css] fix for missing scrollbars in Opera 9.x**
  Negative margins of classes `.skip`, `.hideme` and `.print` were reduced to -1000em to
  avoid this bug.
- **[iehacks.css] CSS bugfixes for different media**
  Bugfixes for the *Doubled Float Margin Bug* and the *Expanding Box Problem* only affect output
  on screen via `@media screen` rule.
- **[print_base.css] print preview in IE6 & linearization of subtemplates**
  Subtemplates are linearized by default. The print preview in IE6 is now more stable.

Navigation Elements

- Adjusted background colors of list elements in *nav_slidingdoor.css* and *nav_shinybuttons.css*.

Other
- Some small adjustmens in the layout examples (page titles changed)

6.1.11   Changes in Version 3.0.1 [15.07.07]

Core Files

- **[fixed] A small rounding bug in Subtemplates**
  In v3.0 the 33- and the 66-percent Subtemplates container of had wrong widhts.

6.1.12   Changes in Version 3.0 [09.07.07]

Download Package & Documentation

- **Bilingual Documentation**
  The extensive documentation as well as all comments in the framework's CSS files are now available in English and in German.
- **Comprehensive Restructuring of the Download Package**
  The download package now distinguishes clearly between the actual framework, the documentation, and layout examples and tools. The structure of the framework was reworked.
- **Optimized Stylesheets for Use in Production**
  The core files of the framework were optimized for use on the live server: they now contain no comments and the file size was greatly reduced.
- **Conversion of all Files to Character Encoding "UTF-8"**
  All framework files were converted to UTF-8 character encoding. As the comments in the files are now available in more than one language, this step was logical and unavoidable.
- **CSS Comments according to the CSSDOC Standard**
  The CSSDOC Standard offers a machine- and human-readable format for comments within CSS files.
- **Better Support of Dreamweaver 7 and 8**
  For Dreamweaver 7 (MX 2004), an alternative base stylesheet is included, which enables a nearly error-free display of YAML-based layouts in the WYSIWYG design mode. A readme.txt is available for both Version 7 as well as for Version 8; this explains all the necessary adjustments for working with YAML.
- **Numerous New Sample Layouts**
  The number of included example layouts has increased greatly. All layout examples base on an appealing new design.

Changes and Corrections

Markup

- **[changed] Simplification of the (X)HTML Source Code**
  The class `.hold_floats` must no longer be explicitly assigned to `#page`: the bugfix is activated by default in the *iehacks.css* file.

Core Files

- **[new] Optimized Stylesheets for Production**
  The stylesheets in the *core/* folder of the YAML framework are also available in optimized form (smaller filesize). These versions have no comments and compromise between readability and smallest possible file size. This saves valuable bandwidth on the live server.

- **[new] Alternative Column Concept based on Classes**
  Four generic CSS classes allow an even simpler choice of which columns display in the basic layout.
- **[new] Generic CSS Classes for Hidden Content**
  The CSS classes `.hideme` and `.print` now provide two options for hiding content onscreen and yet keeping it available for screen readers and text browsers. The classes are defined in *base.css* and thus always available.
- **[new] Handling Oversized Elements in IE**
  IE5.x and 6.0 can now interpret the CSS class `.slidebox`, defined in *iehacks.css*, to let oversized elements merely overlap onto neighboring layout areas rather than destroying a page's layout.
- **[new] New Bugfix for IE Italics Bug**
  A new universal bugfix in *base.css* solves the problem with italic fonts in IE 5.x and 6.0. Previously, this bug was addressed in the content as it occurred.
- **[new] IE7 Bugfix for Print**
  IE7 has problems printing `#col3` because it does not have the property 'hasLayout' and correspondingly forces page breaks. The file *iehacks.css* now contains a suitable bugfix.
- **[new] Bugfix for Firefox 2 overflow:hidden Bug for Print**
  Firefox Version 2.x has problems dealing with the property `overflow:hidden` in printing. A suitable bugfix is now in the *print_base.css* file for the generic class `.floatbox`.
- **[changed] Min-/max-width Support for IE 5.x and IE6**
  The script solution via expressions was reworked, so that IE need no longer be set to Quirks Mode and can interpret EM-based values.
- **[changed] Subtemplates**
  The CSS of the block and content containers was simplified. The block container now encompasses the content by virtue of its *float* property. Oversized content elements are now no longer cut off. The compensation for rounding errors was also improved, so that `.subcolumns` itself is no longer an oversized container (> 100%). The alternative class `.subcolumns_oldgecko` allows support among old Gecko browsers (i.e. Netscape < Version 7.2).
- **[changed] Skip-Link Navigation**
  The skip-links are now immediately visible as soon as the tag navigation is activated. This behavior is required for layout accessibility.
- **[changed] Reworked Print Stylesheets**
  All layout-independent adjustments for printing were split off into an independent CSS component file *print_base.css*, which is loaded via the print stylesheet. This helps keep track of the styles and individual changes are more easily made.
- **[changed] Hover Effects for Links in IE7**
  Hover effects are no longer blocked in IE7 via *iehacks.css*.
- **[removed] Old IE Clearing (up to V2.4) is no longer supported**
  The CSS declarations for the old CSS class `.clear_columns` were removed from the *base.css* file.
- **[removed] Hacks for IE Mac Removed from the Project**
  IE/Mac interprets neither the normal style declarations nor the IE adjustments, as they are loaded via Conditional Comments and the `@media rule`. The Mac hacks (special comments) were rather confusing in the *iehacks.css* file, and were deleted. YAML supports this outdated browser by displaying all content without any CSS formatting at all.

Navigation Elements

- **[new] Navigation Elements Generally**
  All included navigation lists support the tab navigation correctly, including the emphasis on the currently active menu item.
- **[new] Navigation Elements Generally**
  The active menu item in any navigation element can be set either via the ID `#current` or now also via `strong`.
- **[new] Expansion of the vlist Navigation**
  The vlist navigation can now display four instead of the previous two navigation levels.
- **[removed] The Navigation "Sliding Door I" removed**
  The version "Sliding Door II" is still available and was renamed to *nav_sliding_door.css*.

Content Design

- **[new] New CSS Component content_default.css**
  The file *content_default.css* is located in the *yaml/screen/* folder and provides basic formatting for all standard content elements and can be incorporated if desired.
- **[new] Generic CSS Classes for Content Design**
  The *content_default.css* component offers three new CSS classes `.note`, `.important`, and `.warning` for emphasizing content.

Other

- **[new] Debugging Stylesheet**
  A new optional stylesheet *debug.css* makes layout debugging easier (see). Predefined CSS classes for displaying pixel grids, transparencies, or background colors allow a simple emphasis / test of layout elements. The stylesheet also warns the user, should the core stylesheet *iehacks.css* not load correctly.

7 License conditions

7.1 Current and future releases

YAML has been licensed under a Creative Commons Attribution 2.0 License (CC-A 2.0) since version 2.2. For commercial use, two alternative license models are available (see below).

7.2 Older releases

Older releases before version 2.2 were published exclusively under a Creative Commons Attribution 2.0 License (CC-A 2.0).

7.3 General Information

The Creative Commons license permits both the *non-commercial* and the *commercial* use of the framework on the condition that a backlink to the project homepage remains in the layout (see the next section).

Freelancers and web agencies, however, cannot always conform to the terms of use of the Creative Commons license, as customers seldom desire copyright notices of third party projects on their sites. In order to make the use of YAML possible in these cases, two license models for commercial use have been developed, alternative to the Creative Commons license. Both models are set up as single payments and include the use of any future releases.

7.4 YAML under Creative Commons License (CC-A 2.0)

The YAML framework is published under the Creative Commons Attribution 2.0 License, which permits both private and commercial use.

Condition: For the free use of the YAML framework, a backlink to the YAML homepage (http://www.yaml.de) in a suitable place (e.g.: footer of the website or in the imprint) is required.

A small thank you

In general it would be nice to get a short note when new YAML-based projects are released. If you are highly pleased with YAML and the forum support, perhaps you would like to take a look at my *Amazon wish list*?

7.5 YAML under Commercial License (YAML-C)

Two alternative license models are available for using the framework without the otherwise required backlink.

Commercial licenses can be orderd in the YAML Webshop at https://shop.yaml.de.

Project Related License	**59.50 EUR** **(incl. 19% Taxes)**

The purchaser receives the right to use the YAML framework without naming the author or linking back to the YAML page within *one* project, named by the purchaser. The license includes the use of any future releases.

Reproduction and redistribution or providing the code for downloading is only allowed with written permission of the author.

General License	**119.00 EUR** **(incl. 19% Taxes)**

The purchaser receives the general right to use the YAML framework without naming the author or linking back to the YAML page. This right is independent of any project and is permanent. The license includes the use any of future releases.

Reproduction and redistribution or providing the code for downloading is only allowed with written permission of the author.