

# HTML 5

## - Aktueller Technologieüberblick -

Prof. Dr.-Ing. Thomas Wiedemann  
email: [wiedem@informatik.htw-dresden.de](mailto:wiedem@informatik.htw-dresden.de)



HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)  
Fachbereich Informatik/Mathematik

- HTML 5 – Technologieüberblick
  - Zeitliche Einwicklung
  - Basissyntax, Streichlisten und neue Tags
  - **Neue Formularfunktionen**
  - Canvas – 2D- und 3D-Visualisierungen
  - Multimedia und GPS im Griff
  - Offline-Speicher, FileWeb-Worker
  - Empfehlungen für die nächsten Jahre

Quellen : Hinweis: Aufgrund der sehr hohen Aktualität und ständigen Weiterentwicklung sei generell auf die Online-Materialien verwiesen:

- [www.whatwg.org](http://www.whatwg.org) [www.w3.org/TR/html5](http://www.w3.org/TR/html5) ,
- [http://www.w3schools.com/html/html5\\_intro.asp](http://www.w3schools.com/html/html5_intro.asp) weiterhin

[Krö] Peter Kröner. „HTML5 - Webseiten innovativ und zukunftssicher“, 2. Auflage

- das aktuell verwendete HTML 4.01 (bzw. die davon abgeleiteten XHTML Versionen) stammen aus dem Jahr **1999** (bzw. HTML 4.0 aus 1997)
- Umfang der 4.01 Spezifikation ca. 400 Seiten (HTML 5 aktuell ca. 2000 Seiten !!)

## Warum kein neuer Standard seit 1998 ?

- Doch schon – mit XHTML !
  - XHTML ist HTML 4.01 mit den Regeln von XML
  - XML ist wiederum ebenfalls abgeleitet aus dem SGML-Standard und stellt eine Sprache zur Definition von Datenauszeichnungssprachen dar (-> VL)
  - Standardisierungsbemühungen seit 2000 bis 2009???
- **Probleme mit XHTML:**
  - Relativ strenge Syntaxregeln und „nicht verzeihende“ Parser – bei einem Fehler wird NICHTS angezeigt !
  - Ein HTML 4 –Parser (Browser) versucht noch etwas zu retten (sog. „Tag Soup Parser – fischt aus dem Tag-Müll noch etwas Vorzeigbares heraus ... )

## Probleme mit XHTML 1.0 und Konsequenzen :

- Basisidee HTML im XML-Format kommt vom W3C-Konsortium und war an sich nicht schlecht :
  - einheitliche XML-Parser für alle Datenformate UND Internetpräsentation
  - **bessere (strengere) und genauere Regeln**
  - Aber Rechnung ohne den Wirt (=Webdesigner -> Browserhersteller) gemacht!  
Problem der Entwickler und Browser-Programmierer mit dem XML ...
- Im Sinne einer Übergangslösung bietet der XHTML 1.0 –Standard deshalb **zwei Verarbeitungsoptionen** an :
  - XHTML wird doch nach den Regeln von HTML 4 verarbeitet (-> funktioniert, aber macht den Sinn von XHTML – die klarere Syntax - kaputt !)
  - XHTML wird „richtig“ mit XML-Parser interpretiert -> die ganze Seite funktioniert bei nur einem einzigen Fehler ÜBERHAUPT nicht !
  - Microsoft : IE 6.0 bis IE 8.0 verstehen richtiges XHTML nicht !!
  - **Fazit : CHAOS !!!!**

## Probleme mit XHTML 2.0 und Konsequenzen :

- Nach dem nur halb umgesetzten XHTML 1.0 (auch XHTML ½ genannt) versuchte das W3C mit XHTML 2.0 das Chaos zu beseitigen :
- PLANUNG zu XHTML 2.0 -> Keine Alternativen mehr – **NUR NOCH XML !**
- -> **offene Rebellion der Browserhersteller** im Jahr 2004 :
  - Gründung der **WHATWG** (Web Hypertext Application Technology Working Group)
    - eigene Arbeit am Web Applications Standard (Vorläufer von HTML 5)
- **Reaktion des W3C**
  - Auflösung der XHTML 2.0 –Arbeitsgruppe Mitte 2009
  - 2010 letzte Veröffentl. XHTML 2.0 als “Notizen” ohne Standardcharakt.
  - XHTML 2.0 ist damit TOT ! (bzw. nicht ganz, weil HTML 5 kann auch als XML verpackt werden ...)
  - Gründung eigene W3C-Arbeitsgruppe zu HTML 5
- **Fazit** : ab Ende 2011 existieren 2 sich teilweise widersprechende Arbeitsgruppen zu HTML5 – die **WHATWG** und die **W3C-HTML5-AG** !

# Status und Zukunft von HTML 5

- Widersprüchliche Planungen von **WHATWG** und **W3C-HTML5-AG**

## ▪ **WHATWG**

- Plante in 2011 keine Standardisierung vor 2022 ( Hintergrund : W3C-Standard dann , wenn zwei völlig unabhängige, aber vollständige Implementierungen in aktuellen Browsern mit erfolgreichem Bestehen von mindestens 20.000 Einzeltests existieren !! )
- **WHATWG entwickelt HTML als "Living Standard".**
- “A living standard is never fully complete, but always updated and improved. New features can be added, but old functionality can not be removed.”
- WHATWG Living Standard published in 2012 and is continuously updated.

## ▪ **W3C-HTML5-AG:**

- W3C entwickelt HTML5 and XHTML5 Standard als "snapshot" der WHATWG-Definitionen
- W3C HTML5 recommendation released 28. October 2014.
- see [http://www.w3.org/QA/2012/09/getting\\_html5\\_to\\_recommendatio.htm](http://www.w3.org/QA/2012/09/getting_html5_to_recommendatio.htm)

**Hoffnung** : Einigung beider Gremien !!!!??

- „Möge der Bessere oder das jeweils bessere Konzept siegen !!!!“

## Das HTML 4 Format

- hatte nur das **Befehlsformat** definiert, aber nicht die Reaktion des Browsers,

## HTML 5

- definiert nun auch relativ genau die gewünschte **Reaktion des Browsers**
- damit zukünftig **HOFFENTLICH konsistenteres Verhalten der Browser**
- **Deshalb ist auch der Standard deutlich länger** (mit 2000 statt 400 Seiten)

## Gemeinsamkeiten

- Die Grundideen von HTML-Tags, die HTML-Struktur und CSS bleiben bestehen.
- HTML geht meist evolutionär vor : es erweitert bestehende Befehle !
- Einige Regeln werde sogar einfacher !

**Fazit :** **HTML5 ist eine abwärtskompatible Verbesserung von HTML**

**Aktueller Status** (bzw. kann man HTML 5 jetzt schon einsetzen ???)

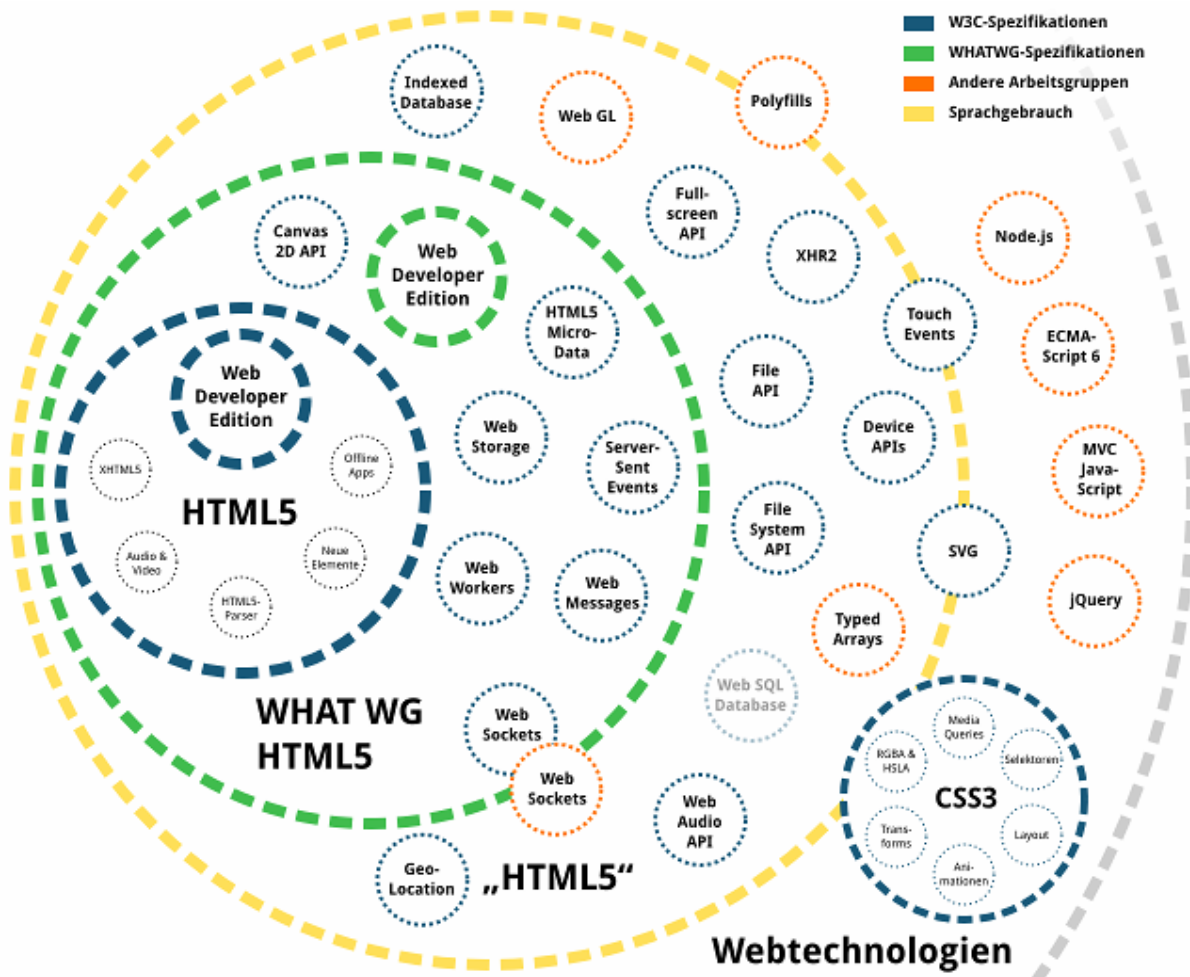
- JA, der Standard ist Browser-Hersteller-getrieben und vieles funktioniert schon !
- mit den üblichen Browser-Tests sollte es gehen

Folgende globale Anforderungen (und Probleme) werden mit HTML 5 adressiert:

- statt Webseiten entwickeln wir zunehmend **Webapplikationen** :
  - Das Look & Feel und die Funktionalität sollte dem wirklicher (Desktop-) Applikationen entsprechen,
  - **Offline-Betrieb** : auch ohne Internetverbindung sollten diese funktionieren
    - Daten müssen auf dem Client durch den Browser zwischengespeichert werden (-> Sicherheit / Vertrauen / Ausspähen durch andere Apps???)
    - Leistungsfähigere Formularelemente und fest eingebaute Unterstützung bei der Formularvalidierung auf dem Client
    - weitere, eingebaute Unterstützung wie Rechtschreibkontrolle etc.
- Leistungsfähige **Zeichenfläche (Canvas)** für 2D und ggf. auch 3D
- bessere Einbindung von **Multimediatechnologien**
  - verlässliche Anzeige (Abspielen) der meisten Multimediaformate
  - sogar Editieren von Bildern und Videos im Browser OHNE Flash



# Die „HTML5“ – Standards



HTML 5 ist

- kein streng definierter Standard, sondern setzt sich aus einer Menge von Subdefinitionen und in Beziehung stehendem Browsertechnologien zusammen :

HTML5 = W3C – Kern

WHAT WG HTML = HTML 5 Standard der WHATWG

„HTML5“ – kein direkter Standard, sondern in Relation zum HTML 5-Standard stehende Technologien

## HTML 5 - Start (wie bisher) mit Doctype

**<!DOCTYPE html>**

- wird von jedem Browser verstanden,
- Zeichensatztag **<meta charset="utf-8">** möglichst weit oben !

## Ebenso mit XML-Syntax

(XHTML 5 ist echtes XML, d.h. jeder Fehler im Dok. führt zum **Abbruch** und zur Fehleranzeige!!)

```
<?xml version="1.0" encoding="utf-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Ein XHTML5-Dokument</title>
  </head>
  <body> <p>
    Ich bin ein echtes XHTML-Dokument!
  </p></body>
</html>
```

sieht man von der XML-Problematik bei XHTML 5 ab, so gilt :

- Die Spezifikation ist deutlich **flexibler** (bzw. lockerer),
- Es wurde vieles **GENERALISIERT** oder **VEREINHEITLICHT** !
- Viele Ausnahmen und Abweichungen vom Regelfall wurden entfernt !
- Es können sehr viele Tags und Attribute ohne Probleme weggelassen werden
  - selbst `<body>` kann weg !
- Beim Weglassen gelten relativ sinnvolle **Defaultregeln** !
- Sinnvolle Elemente, welche in XHTML, aber nicht in den Browsern verbannt wurden, sind wieder im HTML5 –Standard !

# Minimalismus mit HTML 5

- Neue Festlegungen zu Pflicht/ Optional – ergibt in der Folge (Quelle [Krö] ) :

Element	Start-Tag	End-Tag
<html>	Optional	Optional
<head>	Optional	Optional
<body>	Optional <sup>6</sup>	Optional
<li>	Pflicht	Optional
<dt>	Pflicht	Optional
<dt>	Pflicht	Optional
<p>	Pflicht	Optional
<colgroup>	Optional	Optional
<thead>	Pflicht	Optional
<tbody>	Optional	Optional
<tfoot>	Pflicht	Optional
<tr>	Pflicht	Optional
<th>	Pflicht	Optional
<td>	Pflicht	Optional
<rt>	Pflicht	Optional
<rp>	Pflicht	Optional
<optgroup>	Pflicht	Optional
<option>	Pflicht	Optional

## Langform (gültiges HTML) :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Hallo Welt!</title>
    </head>
    <body>
        <p>Hallo Welt!</p>
    </body>
</html>
```

## Kurzform (gültiges HTML5):

```
<!DOCTYPE html>
    <meta charset="utf-8">
    <title>Hallo Welt!</title>
    <p>Hallo Welt!
```

# Neue Tag-Kategorien

Die Tags und Attribute wurden neu sortiert und in Gruppen eingeteilt:

- **Metadata:** Metadaten (z. B. `<base>` und `<tile>`)
- **Flow:** Inhalt des Dokuments (z. B. `<p>`-, `<img>`- und `<strong>`-Elemente, aber auch bloßer Text)
- **Sectioning** : neue semantische Elemente zur Einteilung eines Dokuments (z. B. `<section>` und `<nav>` )
- **Heading:** Überschriften (z. B. `<h1>` und `<hgroup>`)
- **Phrasing:** Auszeichnungen und Elemente auf Text-Ebene (z. B. `<span>` und `<img>`)
- **Embedded:** Externe Inhalte, die in das Dokument eingebettet werden (z. B. `<iframe>` und `<video>`)
- **Interactive:** Elemente für die Nutzerinteraktion (z. B. `<a>` und `<input>`)

## Ersatzlos und absolut gestrichen :

- Frames (nur Iframes gibt es weiterhin)
- Doctype-Varianten Strict, Transitional und Frameset
- reine Präsentationselemente wie `<tt>` und `<big>`
- reine Design-Attribute wie `bgcolor` und `align`
- das `<acronym>`-Element (ersetzt durch `<abbr>`) -> Probleme mit IE
- das `<applet>`-Element (ersetzt durch `<object>`)

## Gestrichen, aber spezifiziert

- einige Tags sind gestrichen, werden aber zwecks Browser-Verhalten weiterhin spezifiziert !
- z.B. wird `bgcolor`-Attribut ausführlichst beschrieben, gehört aber nicht mehr zum Standard !!!!!

## Nicht gestrichen, aber auch nicht empfohlen

- alte DOM-Methoden `document.write()` und `document.writeln()`
  - sind teilweise problematisch (Timing / Zugriff)
  - verhindern die asynchrone Ausführung von Skripten
  - **funktionieren in XHTML5 überhaupt nicht**
- konnten jedoch aufgrund Verbreitung nicht entfernt werden, doch von ihrer Verwendung wird dringend abgeraten.

# Beispiele von sinnvollen Erweiterungen

## Im Bereich Content-Auszeichnung:

- Das `<a>` - Link-Attribut darf nun alle möglichen Inhalte umschliessen:

```
<a href="a.html">
```

```
  <h1>Lorem Ipsum</h1> <p> und so weiter ...</p>
```

```
</a>
```

- Das `rel`-Attribut in Links erlaubt nun viele Linkstypen (->Suchmaschinen)
  - `Rel=„alternate“` (Alternative Repräsentation), `archives` (Archivmaterial)
  - `„author“` (Link zum Autor), `bookmark` , `external`,
  - `first`, `last` (Liste von Links), `help`, `icon`, `index` , `license` ...

## Editmodus und Rechtschreibprüfung (noch stark browserabhängig )

```
<p contenteditable="true" spellcheck="true">
```

```
  Das <code>spellcheck</code>-Attribut sorgt  
  für Rechtschreibprüfung
```

```
</p>
```

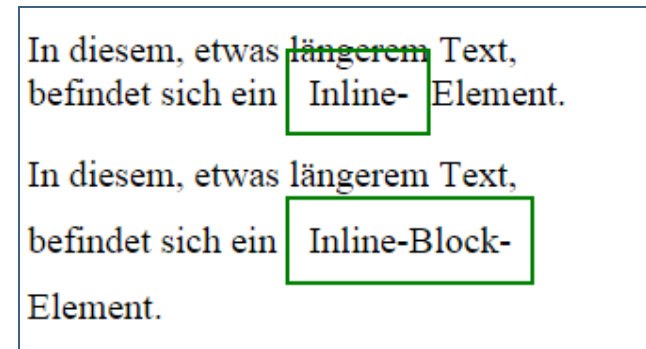
# Stärkere Bedeutung und Erweiterungen des display-Attributes

**Das universelle display-Attribut hat die größte Bedeutung für das Layout !**

Details/Quelle vgl. <https://wiki.selfhtml.org/wiki/CSS/Eigenschaften/Anzeige/display>

**Bereits seit CSS 1.0 definiert und immer erweitert mit den folgenden Optionen:**

- **display : inline;** (Defaultwert) - Element erzeugt Box in der Zeile (frei verschiebbar, Größe wird nur durch Elementinhalt bestimmt, width / height haben keine Wirkung!!) - Typische Vertreter: `<span>`, `<em>`, `<strong>`
  - Hinweis: Damit werden unbekannte Tags (->IE) immer als inline-Box angezeigt
- **display : block;** - Element erzeugt vertikal angeordnete Box (Breite 100% oder über width / height definiert)  
Typische Vertreter: `<div>`, `<p>`, `<article>`
- **display : inline-block;** - kombiniert inline und block box  
- innerhalb einer Zeile ist eine Box mit width/height definierbar  
Typische Vertreter: `<input>`, `<textarea>` (siehe Beispiel rechts)
- **display : none ;** - keine Anzeige

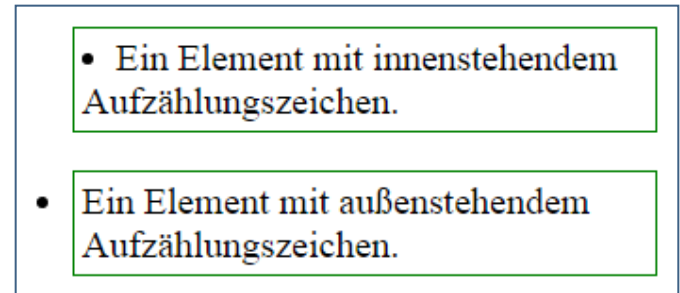




## Neue Optionen mit CSS 3.0:

- **display : list-item;** - erzeugt 2 Boxen für Aufzählungszeichen und Text in der Zeile (mit wählbarer Position für Listenzeichen)

```
... <style> p { display: list-item; }  
  .v1 { list-style-position: inside; } </style>  
... <body><p class="v1">Ein Element mit innen... </p>  
  <p>Ein Element mit außen ... </p> </body>
```

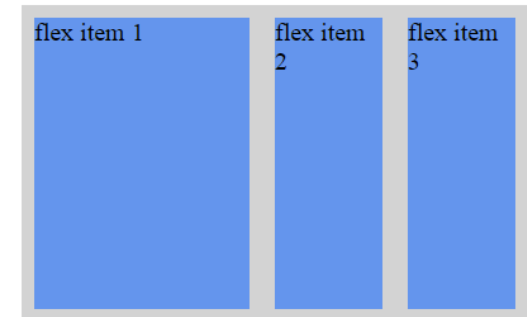


- **display : run-in ;**
  - Kombiniert block oder inline – Modus je nach Kontext der umgebenden Elemente (bei einem block-Nachfahren wird auch das mit run-in – gekennzeichnete Element als block behandelt, anderenfalls erfolgt eine Anzeige im inline-Modus)
  - Noch keine 100% Browserumsetzung (war raus in CSS 2.1, in CSS 3.0 drin .)

# Erweiterungen des display-Attributes - das Flexbox-Layout

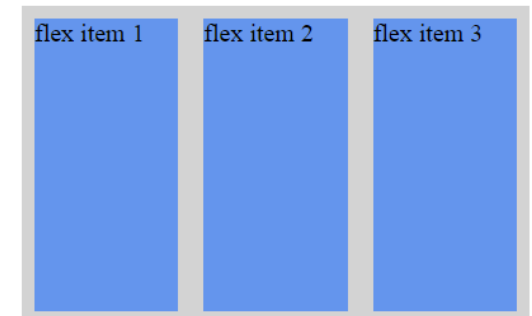
- **display : flex; (oder auch inline-flex)** (display: box / flexbox sind veraltet)
- Details: <https://wiki.selfhtml.org/wiki/CSS/Eigenschaften/Gr%C3%B6%C3%9Fenangaben/Flexbox>
- definiert Flex-Container mit Flex-Items, sehr neu – unterstützt ab IE11, Android 4.4
- ggf. für alte Browser mit Präfix -webkit -ms-flex
- erlaubt deutlich flexiblere Anpassungen bei Responsive Design (-> entspr. VL)

```
<style> .flex-container {display: -webkit-flex; display: flex;
    width: 400px; height: 250px; background-color: grey; }
.flex-item { background-color: blue; margin: 10px; }
.item1 { -webkit-flex: 2; flex: 2; } <!-- relative Maße --!>
.item2 { -webkit-flex: 1; flex: 1; } <!-- relative Maße --!>
.item3 { -webkit-flex: 1; flex: 1; }
</style></head> <body>
<div class="flex-container">
  <div class="flex-item item1">flex item 1</div>
  <div class="flex-item item2">flex item 2</div>
  <div class="flex-item item3">flex item 3</div>
</div>
```



mit

```
.item1 { -webkit-flex:2; flex:2; }
```



Die wichtigsten Umstellungen (in Anlehnung an [http://www.w3schools.com/html/html5\\_migration.asp](http://www.w3schools.com/html/html5_migration.asp) )

- **Änderung des Doctype's**

**<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">**

⇒ **<!DOCTYPE html>**

- **Änderung des HTML5 Encodings**

**<meta http-equiv="Content-Type" content="text/html; charset=utf-8">**

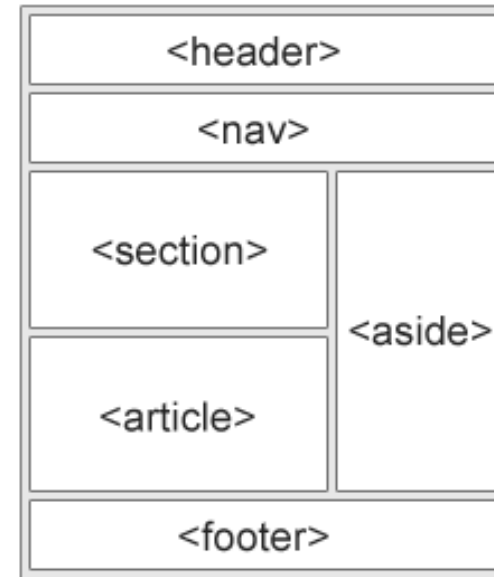
⇒ **<meta charset="utf-8">**

# Migration von HTML 4 zu HTML 5 - Layoutbereiche

Die wichtigsten Umstellungen (in Anlehnung an [http://www.w3schools.com/html/html5\\_migration.asp](http://www.w3schools.com/html/html5_migration.asp) )

- Umstellung der bisherigen **CSS-id**'s auf die neuen **HTML5-Tags**

<code>&lt;div id="header"&gt;</code>	⇒	<code>&lt;header&gt;</code>
<code>&lt;div id="menu"&gt;</code>	⇒	<code>&lt;nav&gt;</code>
<code>&lt;div id="content"&gt;</code>	⇒	<code>&lt;section&gt;</code>
<code>&lt;div id="post"&gt;</code>	⇒	<code>&lt;article&gt;</code>
<code>&lt;div id="sidebar"&gt;</code>	⇒	<code>&lt;aside&gt;</code>
<code>&lt;div id="footer"&gt;</code>	⇒	<code>&lt;footer&gt;</code>



- Analoge Umstellung der bisherigen **CSS-Styles** auf die neuen **HTML5-Tags**

```
div#header,div#footer,div#content,div#post { border:1px solid grey; ... }
```

```
header,footer,section,article { border:1px solid grey; ... }
```

# Migration von HTML 4 zu HTML 5 – Support alter Browser

Die wichtigsten Umstellungen (in Anlehnung an [http://www.w3schools.com/html/html5\\_migration.asp](http://www.w3schools.com/html/html5_migration.asp) )

- Die neuen Tagtypen (siehe Seite vorher) sollten alten Browsern zur Sicherheit als Block-Typen bekannt gemacht (sonst Rendering als inline ...)

⇒ **header, section, footer, aside, nav, main, article, figure**  
**{ display: block; }**

- Ältere IE-Versionen (vor/bis Version 9) können auf unbekannte (=neue) Tags keine CSS-Formate anwenden -> Abhilfe schafft eine spezielle JavaScript-Extension **html5shiv**

```
<head> <title>Styling HTML5</title>
<!--[if lt IE 9]>
<script src='http://html5shiv.googlecode.com/svn/trunk/html5.js'></script>
<![endif]-->
```

⇒ Historie zu Shiv : <http://www.paulirish.com/2011/the-history-of-the-html5-shiv/>

## Wichtigste (und überfällige) Änderungen durch HTML 5 in diesem Bereich :


- sehr viele neue Input-Felder-Typen mit fest eingebauter Validierungsoption (->ff.)
- zur Zeit am besten unterstützt von Opera und Chrome
- Browser ohne entsprechende Unterstützung zeigen normale Textfelder an

## Globale Neuentwicklungen (meist universell einsetzbar)

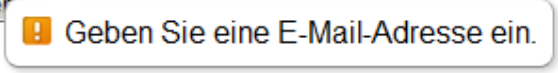
- neues Attribut **data-\*** erlaubt adhoc-“Erfindung“ eigener Data-Tags
  - Ersatz von \* durch eigene Namen
  - zur Ablage von Steuerparametern oder Hidden-Parameters,
  - z.B. `<input ... Data-id="KundenId" Data-PLZ="01069"`

# Neue Formular-Elementtypen

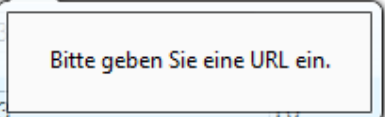
## Neue Inputfelder :

- **E-mail:** `<input type="email" name="email1" />`
  - validiert auf @ (nicht bei IE9??), Chrome :
  - schaltet ggf. Tastatur um ...(Smartphone)
- **URL** `<input type="url" name="user_url" >`
  - checkt auf http:// (Rest ist egal ... ?)
- **Colorpicker:** `<input type="color" name="user_color" />` Color: 
- **Number** `<input type="number" name="points" min="1" max="10" />`
  - mit Up/Down-Buttons und regionaler Fehlermeldung (sehr gut!)
- **Range** mit Slider-Anzeige : 
- **Pattern** `<input type="text" name="country_code" pattern="[A-z]{3}" />`
- **Search-Inputfeld** (wie normales Textfeld, nur ggf. spezielles Layout des BS (z.B. Apple!!))

E-mail:

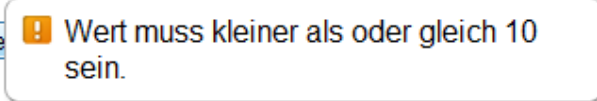
Sender 

FF8: Homepage:

Points: 3 

Points: 1

Points:

Sende 

Search-Input:

Normales Text-Input:

- **Output-Element** – generiert Anzeigen oder Berechnungen

```
<form oninput="x.value=parseInt(a.value)+parseInt(b.value)">0
```

```
<input type="range" name="a" value="50" />100
```

```
+<input type="number" name="b" value="50" />
```

```
=<output name="x" for="a b"></output>
```

- **Balkenanzeige** Info: `<meter value="0.85">85%</meter>`

Info: 

- **Data-List** : bindet Auswahlliste an Input (+Validierung gegen Liste)

Webpage: `<input type="url" list="url_list" name="link" />`

  
`<datalist id="url_list"> <option label="Google" value="http://www.google.com" />`

`<option label="Microsoft" value="http://www.microsoft.com" /> </datalist>`

- **Keygen** zur Generierung sicherer Kommunikationsschlüssel

Username: `<input type="text" name="usr_name" />`

Encryption: `<keygen name="security" />`



-> `usr_name=tom&security=MIICQDCCASgwgge ...` (2048 bit-Schlüssel)



## Neue Form-Attribute:

- Autocomplete
- Novalidate

## Neue Input- Attribute:

- autocomplete
- autofocus
- form
- form overrides  
(formaction, formenctype, formmethod, formnovalidate, formtarget)
- height and width
- list
- min, max and step
- multiple
- pattern (regexp)
- placeholder
- required

# Drag & Drop - API

Zur Erstellung von hochinteraktiven Anwendungen ist die **Drag & Drop-Funktionalität** neu verfügbar

1. Drag & Drop bei zu „dragendem“ Element erlauben und definieren mit

```
<script> function drag(ev) { ev.dataTransfer.setData("text", ev.target.id); }  
</script>  
<img draggable="true" ondragstart="drag(event)" >
```

2. In potentiellen Zielelementen das „dropen“ erlauben und drop definieren:

```
<script> function allowDrop(ev) { ev.preventDefault(); }  
        function drop(ev) { ev.preventDefault();  
                            var data = ev.dataTransfer.getData("text");  
                            ev.target.appendChild(document.getElementById(data)); }  
<!-- html code -->  
<div id="div1" ondragover="allowDrop(event)" ondrop="drop(event)" ></div>
```

Komplettes Beispiel: [http://www.w3schools.com/html/html5\\_draganddrop.asp](http://www.w3schools.com/html/html5_draganddrop.asp)

## Zitat zur Einführung des Webworker – Begriffs (nach [Krö] S. 420):

„Web Workers machen JavaScript zwar nicht schneller, aber sie sorgen dafür, dass der Surfer nicht mehr merkt, wie langsam es ist.“

## Hintergrund :

- Die Javascript-Ausführung mehrerer Skript erfolgt **SEQUENTIELL** !
- Blockierende oder sehr langsame Skripte blockieren das Gesamtsystem !

## Lösung:

- Parallele Ausführung der Skripte mit Webworkers (mit beliebiger externer JS-Datei)

- Worker-Object handelt Ausführung quasi in neuen Prozess
- Achtung: ggf. neue Detailprobleme durch Timing (später gestartetes Skript kann eher fertig sein !!!)

```
// quadratwurzel.html
var start = document.getElementById('start');
// Worker erzeugen
var meinWorker = new Worker('quadratwurzel.js');
// Worker das Startsignal geben
start.onclick = function(){
  meinWorker.postMessage("");
};
// Erfolgsmeldung empfangen
meinWorker.onmessage = function(){
  alert('Fertig');
};
```

## Problem: temporär fehlende Internetverbindung - Lösungen ?

### 1. Webstorage für Daten mit

- dauerhaften Local Storage : `localStorage`.Iname="Smith";  
document.write(„Name: " + `localStorage`.Iname);
  - Temporären Session-Storage : `sessionStorage`.lastname="Smith";  
document.write(`sessionStorage`.lastname);
- `</script>`

### 2. Application- Cache – Storage

- dient zum Vorhalten von **Ressourcen** für den Offline- Fall !
- nichts gemein mit dem normalen Browsercache : präzise kontrollierbar, versioniert und synchronisiert eigenständig,
- eine Ressource muss nicht erst durch den Besucher aufgerufen werden,
- Cache Manifest-Datei definiert zu speichernde Ressourcen:

```
<! Einbindung in HTML-Datei mit >  
<html manifest="cache-manifest.manifest">
```

#### CACHE MANIFEST

```
# Erste Zeile ist Pflicht  
# Ressourcen für Offline-Betrieb :  
img/foo.png  
scripts/bar.js  
# Alles folgende NICHT cachen  
NETWORK:  
news.html
```

# Server-Sent-Events (SSE)

Für häufige, aber kleinen Aktualisierungen gibt es die neuen Server-Sent-Events, bei welchen der Server von sich aus Updates sendet. Dies reduziert den Netzwerktraffic, da kein ständiges Pollen mehr nötig ist.

1. Initialisierung eines **Requests** und einer **Callback-Routine** auf Clientseite

```
<script> if(typeof(EventSource) !== "undefined") {  
    var source = new EventSource("demo_sse.php");  
    source.onmessage = function(event) {  
        document.getElementById("result").innerHTML += event.data + "<br>"; };  
} else { document.getElementById("result").innerHTML = "Sorry –no SSE ... "; }
```

2. Ädequates Skript auf Serverseite mit Mimetype **text/event-stream**

```
<?php header('Content-Type: text/event-stream');  
header('Cache-Control: no-cache');  
$time = date('r'); echo "data: The server time is: {$time}\n\n";  
flush();  
  
?>
```

# HTML5 Element `<canvas>`

- ist ein HTML5-Element und erlaubt es dynamisch Grafiken erzeugen
- bedeutet auf deutsch Leinwand oder Arbeitsfläche
- wird von allen aktuellen Browsern unterstützt (IE erst ab Version 9)
- wurde ursprünglich von Apple für deren HTML-Bibliothek Webkit entwickelt

## Anwendung

- HTML-typische Verwendung mit öffnenden und schließenden Tag
- dazwischen kommt ein Fallback-Inhalt
- Breite und Höhe werden als Attribute angegeben
- `<canvas>` besitzt keine `alt-` und `src-`Attribute
- zur Identifikation/Steuerung für JavaScript wird eine ID vergeben
- außerdem notwendig: JavaScript-Funktion, die das Zeichnen übernimmt

# Beispiel <canvas>

```
<canvas id="canvas1" width="200" height="150">
```

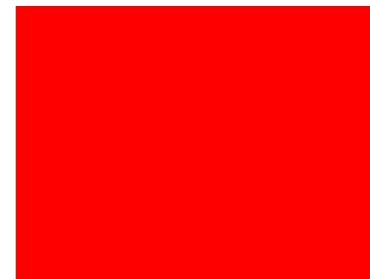
Your browser does not understand canvas-elements

```
</canvas>
```

```
function draw_rect(){  
  var canvas = document.getElementById('canvas1');  
  if(canvas.getContext){  
    var context = canvas.getContext('2d');  
    context.fillStyle = "rgb(255, 0, 0)";  
    context.fillRect(0, 0, canvas.width, canvas.height);  
  }  
}
```

Funktion ausführen (im <body>-Bereich), z.B.:

```
<body onload=" draw_rect()">
```



# Specific characteristics / actual limitations

- `<canvas>` unterstützt nur eine primitive Form: Das Rechteck!
- für komplexere geometrische Formen müssen Pfade generiert werden
- zum Zeichnen von Rechtecken stehen drei Funktionen zur Verfügung:
  - `fillRect(x,y,width,height)` > Rechteck
  - `strokeRect(x,y,width,height)` > Rechteck mit Kontur
  - `clearRect(x,y,width,height)` > Transparenter Bereich

Tutorials und Beispiele zu `<canvas>`

<http://www.html5andcss3.org/html5canvas.php>

[https://developer.mozilla.org/en/Canvas\\_tutorial](https://developer.mozilla.org/en/Canvas_tutorial)



- eine 2. Option für 2D-Grafiken ist das (alte) SVG-Format
- erste Version von 2001 (aber Streit mit Microsoft und nicht im IE)
- aktuell Arbeiten an SVG 2.0 (bis Ende 2016?)
- SVG ist XML-basiert -> die Syntax ist genau einzuhalten

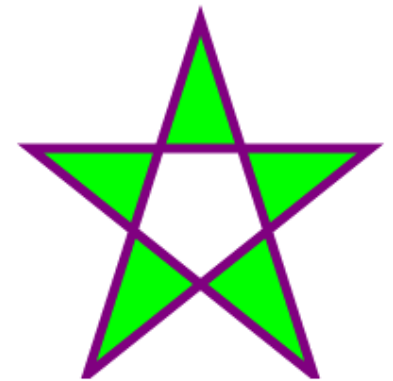
```
<svg width="300" height="200">
```

```
<polygon points="100,10 40,198 190,78 10,78 160,198"
```

```
style="fill:lime;stroke:purple;stroke-width:5;fill-rule:evenodd;" />
```

Sorry, your browser does not support inline SVG.

```
</svg>
```



Tutorials und Beispiele zu <svg>

[http://www.w3schools.com/html/html5\\_svg.asp](http://www.w3schools.com/html/html5_svg.asp)

[http://www.w3schools.com/graphics/svg\\_intro.asp](http://www.w3schools.com/graphics/svg_intro.asp)

# 3D- WebGL

- eine neue Option zur 3D-Darstellung ist der WebGL-Standard
- WebGL auf Basis von OpenGL ES (ES=Embedded Systems)
- relativ stark vereinfachtes OpenGL (OpenGL Light)
- bis auf IE schon relativ gut durch alle modernen Browser unterstützt
- Def. ebenfalls mit canvas-tag, aber danach komplette Programmierung mit JS:

```
<script type="application/javascript">
```

```
function WebGLStart() {
```

```
    initGL();    initShaders();
```

```
    var canvas = document.getElementById("WebGL-canvas");
```

```
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
```

```
// Drawing
```

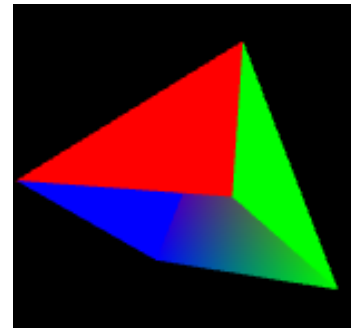
```
    drawScene();
```

```
}
```

```
...
```

```
<body onload="WebGLStart()">
```

```
<canvas id="WebGL-canvas" width="500" height="500"></canvas>
```



ca. 200 Codezeilen für  
einfaches Dreieck mit  
interaktiver Animation

# Vergleich und Bewertung der Grafikoptionen

## Eigenschaften zur Einsatzbewertung:

### 2D-Canvas

- Canvas-Size-abhängig
- Rendering-Optionen (z.B. bei Text) noch schwach im Vergleich zu SVG
- Auch für einfache (2D)-Spiele geeignet

### SVG

- echtes Vektorformat damit größenunabhängig (z.B. auch für große Google-Maps Applikationen)
- sehr gutes (Farb- und Style-Optionen++), allerdings auch langsames Rendering
- nicht geeignet für Spiele u.ä.

### WebGL

- setzt OpenGL-fähige Hardware voraus (bei Mobile Geräten nur teilweise)
- Programmierung vergleichsweise aufwändig. Dank HW-Beschleunigung auch bei großen Szenen schnell

# Multimedia (HTML5-Video/Audio)

- Media Elemente `<video>` und `<audio>` ermöglichen erstmals(!) bewegte Bilder nebst Ton ohne Plug-ins von Dritten zuverlässig in Websites einzubetten
- damit teilweises Ersetzen von Flash, welches bekanntlich als Plug-in installiert werden muss, möglich
- `<audio>`-Element dient dem Einbinden von Sounds oder Audiostreams, z.B.:

```
<audio src="audio.ogg" controls>
```

```
<a href="audio.ogg">audio.ogg herunterladen</a>
```

```
</audio>
```

- `<video>`- Element zum einbinden von Videos, z.B.:

```
<video src="video.ogg" width="427" height="240" controls>
```

```
<a href="video.ogg">video.ogg herunterladen</a>
```

```
</video>
```



# Geolocation-API

- erweitert das `window.navigator`-Objekt um ein Objekt `geolocation`
- bietet drei Methoden für das Auslesen von Geolocation-Informationen
  - eine zum Auslesen des Standorts
  - zwei zum Einrichten und Steuern einer dauerhaften Überwachung von Positionsveränderungen
- Bestimmung der aktuellen Position mit der Methode `getCurrentPosition()`,  
Beispiel:

```
navigator.geolocation.getCurrentPosition(function(position){  
/* "position" enthält die Geolocation-Informationen */    });
```

Die Schnittstelle verwendet folgende technische Quellen für die Position:

- IP-Adresse (jedoch +/- 2 ... 50 km je nach IP-Pool)
- WLAN-Netzwerke
- Funk-Signale (Mobilfunk)
- GPS-Sender (nur bei `,enableHighAccuracy'`)

# Geolocation-API -Anwendungsbeispiel

```
<script> var x = document.getElementById("demo");  
function getLocation() {  
  if (navigator.geolocation) {  
    navigator.geolocation.getCurrentPosition(showPosition, showError);  
  } else { x.innerHTML = "Geolocation not supported by this browser."  
  }  
}
```

```
function showPosition(position) {  
  var latlon = position.coords.latitude + "," + position.coords.longitude;  
  var img_url = "http://maps.googleapis.com/maps/api/staticmap?  
    center="+latlon+"&zoom=14&size=400x300&sensor=false";  
  document.getElementById("mapholder").innerHTML =  
  "<img src='"+img_url+"'>";  
}
```

# Geolocation-API -Anwendungsbeispiel -Fehlerhandling

```
<script> ...  
function showError(error) {  
    switch(error.code) {  
        case error.PERMISSION_DENIED:  
            x.innerHTML = "User denied the request for Geolocation." ;    break;  
        case error.POSITION_UNAVAILABLE:  
            x.innerHTML = "Location information is unavailable." ;        break;  
        case error.TIMEOUT:  
            x.innerHTML = "The request to get user location timed out." ;    break;  
        case error.UNKNOWN_ERROR:  
            x.innerHTML = "An unknown error occurred." ;                break;  
    }  
} </script>
```

# Positionsänderungen überwachen

- periodisch die aktuellen Geokoordinaten einholen, mit Methode `watchPosition()`
- Beispiel:  
// Überwachung starten  

```
var timerId = navigator.geolocation.watchPosition(
erfolgCallback, fehlerCallback, options);
```

  
// Überwachung stoppen  

```
document.getElementById('stopButton').addEventListener('click',
function(){
navigator.geolocation.clearWatch(timerId);
}, false);
```
- Ist ggf. nicht in allen Browsern implementiert -> mit Javascript nachbauen



# Allgemeine HTML5-Style-Empfehlungen

In Anlehnung an [http://www.w3schools.com/html/html5\\_syntax.asp](http://www.w3schools.com/html/html5_syntax.asp)

sollten folgende Styleguide-Empfehlungen beachtet werden:

- obwohl viele Tags weggelassen werden können sollte die Basisstruktur eines HTML4-Dokumentes (vorerst) beibehalten werden (ältere Apps, Screenreader ...) – also mit `<html><head><body> ...</body></html>`
- Tag sollten nach Möglichkeit immer auch geschlossen werden
- Eine XML-Wohlgeformtheit sollte (immer noch) angestrebt werden.

## Für die Lesbarkeit sinnvoll:

- generelle Kleinschreibung der Tags (bei Filenamen (speziell unter UNIX) und JavaScript-Code ist die Groß/Kleinschreibung aber ggf. relevant !!)
- Attribute weiter mit “...” schreiben (bei Spaces sowieso nötig!)
- Nach W3Schools (Link siehe oben)

**„With HTML5, you must create your own Best Practice, Style Guide and Coding Conventions.”**

# Fazit : HTML5 – Empfehlungen

Aktuell sollten folgende Regeln befolgt werden :

- genaue **Analyse der aktuellen Spezifikationen**
  - im Gegensatz zu den alten Specs sollten die HTML 5-Specs genauer und verbindlicher für die Browser sein – das Spec-Studium macht also mehr Sinn
- Einsatz von HTML auf Basis **Kosten/Nutzen-Verhältnis**
  - zunächst sollten nur wirklich sinnvolle oder überragende neue Funktionen verwendet werden (z.B. neue Formular- oder CANVAS-Funktionen)
- die schnellen Updatezyklen machen **TESTEN** noch wichtiger
  - alle wichtigen Browser in den letzten 3..5 Versionen testen !
- schaffe **Rückfallebenen** (Graceful Degradation und Progressive Enhancement)
  - auch bei Ausfall von neuen Funktionen sollte die Site noch benutzbar bleiben
  - Step-by-step-Vorgehensweise vermeidet größere Reinfälle (erst eine relativ unwichtige Siteebene mit HTML 5 aufpeppen, dann bei Erfolg 4 Wochen später den Rest ... )

## Viel Glück und Erfolg mit HTML 5 !