



) HTML5)

Update auf den aktuellen Webstandard

Version 1.4.5 (01.05.2016, Autor: Frank Bongers, Webdimensions.de)
© 2016 by Orientation In Objects GmbH
Weinheimer Straße 68
68309 Mannheim
<http://www.oio.de>

Das vorliegende Dokument ist durch den Urheberschutz geschützt. Alle Rechte vorbehalten. Kein Teil dieses Dokuments darf ohne Genehmigung von Orientation in Objects GmbH in irgendeiner Form durch Fotokopie, Mikrofilm oder andere Verfahren reproduziert oder in eine für Maschinen, insbesondere Datenverarbeitungsanlagen verwendbare Sprache übertragen werden. Auch die Rechte der Wiedergabe durch Vortrag sind vorbehalten.

Die in diesem Dokument erwähnten Soft- und Hardwarebezeichnungen sind in den meisten Fällen eingetragene Warenzeichen und unterliegen als solche den gesetzlichen Bestimmungen.

Inhaltsverzeichnis

1.	EINFÜHRUNG	7
1.1.	WAS IST HTML5?	7
1.2.	INHALT DES SEMINARS	9
2.	DIE DOKUMENTSTRUKTUR	9
2.1.	DIE DOCTYPE-DEKLARATION	10
2.2.	DAS ROOT-ELEMENT	11
2.2.1.	Der Namensraum	12
2.2.2.	Das lang-Attribut und die Sprachdeklaration	12
2.3.	DAS <HEAD> ELEMENT	13
2.3.1.	Meta-Elemente in HTML5	14
2.3.2.	Das <script>-Element in HTML5	14
2.3.3.	Das <link>-Element in HTML5	14
2.3.4.	HTML5 vs. XHTML5	15
3.	NEUE SEMANTISCHE ELEMENTE	17
3.1.	DOKUMENTHIERARCHIE – DIE „OUTLINE“	18
3.1.1.	Das <section> Element	20
3.1.2.	Das <article> Element	21
3.1.3.	Das <aside> Element	23
3.1.4.	Das <header> Element	26
3.1.5.	Das <hgroup>-Element	27
3.1.6.	Das <footer>-Element	29
3.1.7.	Das <nav>-Element	31
3.2.	INLIEELEMENTE	32
3.2.1.	Das <mark>-Element	33
3.2.2.	Das <time>-Element	34
4.	FORMULARE IN HTML5	36
4.1.	HTML5 <FORM>-TAG	36
4.1.1.	Attribute des Form-Elements	37
4.2.	HTML5 <INPUT> TAG	38
4.2.1.	Attribute des <input>-Tags	38
4.2.2.	Modernizr und Input-Attribute	43
4.2.3.	Das type-Attribut von <input>	43
4.2.4.	Modernizr und Inputtypen	44
4.3.	INPUTTYP: TEXT	44
4.4.	INPUTTYP: BUTTON	44
4.5.	INPUTTYP: CHECKBOX	45
4.6.	INPUTTYP: COLOR	45
4.7.	INPUTTYP: DATE, MONTH, WEEK, TIME, DATETIME	45
4.8.	INPUTTYP: EMAIL	47
4.9.	INPUTTYP: FILE	48
4.10.	INPUTTYP: HIDDEN	48

4.11.	INPUTTYP: IMAGE	48
4.12.	INPUTTYP: NUMBER	49
4.12.1.	Zusatzattribute für Inputtyp number	49
4.13.	INPUTTYP: PASSWORD.....	49
4.14.	INPUTTYP: RADIO	50
4.15.	INPUTTYP: RANGE	50
4.15.1.	Zusatzattribute für Inputtyp range	51
4.16.	INPUTTYP: RESET.....	51
4.17.	INPUTTYP: SEARCH	51
4.18.	INPUTTYP: SUBMIT.....	52
4.19.	INPUTTYP: TEL	52
4.20.	INPUTTYP: URL.....	52
4.21.	VALIDIERUNG VON FORMULARFELDERN	53
4.21.1.	Unterdrücken der Validierung.....	54
5.	MEDIENEINBINDUNG – AUDIO UND VIDEO	55
5.1.	AUDIO EINBINDEN MIT <AUDIO>	55
5.1.1.	Boolesche Attribute von AUDIO	55
5.1.2.	Funktionen der JavaScript-API.....	56
5.1.3.	Einbinden von Audiodateien mit SOURCE-Element	57
5.1.4.	Modernizr und <audio>	58
5.2.	VIDEO EINBINDEN MIT <VIDEO>	59
5.2.1.	Modernizr und <video>	60
5.3.	ATTRIBUTE FÜR <AUDIO> UND <VIDEO>	61
5.3.1.	Gemeinsame Attribute <audio> und <video>.....	61
5.3.2.	Spezielle Attribute für <video>	62
5.4.	FALLBACK VERHALTEN	63
5.4.1.	Fallback auf Quicktime.....	63
5.4.2.	Fallback auf andere Plugins.....	64
5.5.	API FÜR VIDEO UND AUDIO.....	65
5.5.1.	Methoden	65
5.5.2.	Events	66
5.5.3.	Eigenschaften	68
6.	DAS CANVAS-ELEMENT	68
6.1.1.	Der Drawing-Context und seine Methoden	69
6.1.2.	Koordinaten und Zeichenpfade	71
6.1.3.	Modernizr und <canvas>.....	73
7.	EINBETTUNG VON SVG UND MATHML	73
7.1.	BEIPIEL MATHML.....	74
7.2.	BEISPIEL SVG.....	75
7.2.1.	Modernizr und SVG.....	75
8.	GEOLOCATION API.....	76
8.1.	PROPERTIES DES POSITIONSOBJEKT-OBJEKTS.....	79
8.1.1.	Modernizr und Geolocation	80
9.	DRAG AND DROP API.....	81
9.1.	DRAG & DROP-EVENTS.....	83

9.1.1. Modernizr und Drag & Drop	84
10. VALIDIEREN VON HTML5-DOKUMENTEN	86
10.1. VALIDATOR.NU.....	86
10.2. W3C-VALIDATOR.....	86
11. TOOLS UND HILFSMITTEL.....	87
11.1. ZEN-CODING	87
11.2. REMY SHARPS HTML5-SHIV.....	90
11.3. MODERNIZR	92
11.3.1. Download und Konfiguration	92
11.3.2. Einbinden von Modernizr	93
11.3.3. Auswertung der Featureprüfung über CSS-Klassen	94
11.3.4. Auswertung der Featureprüfung über JavaScript.....	95
11.4. HTML5 BOILERPLATE.....	97
11.4.1. HTML-Template	97
11.4.2. CSS-Dateien	98
11.4.3. JavaScript-Dateien.....	99
11.4.4. Weitere Ressourcen.....	99
11.5. INITIALIZR.....	100
12. ÜBERBLICK ÜBER DIE SPEZIFIKATIONEN.....	101
12.1. SPEZIFIKATIONEN FÜR ANWENDUNGSPROGRAMMIERER	101
12.2. SPEZIFIKATIONEN FÜR AUTOREN.....	101
12.3. WEITERE SPEZIFIKATIONEN	102
13. LITERATUR.....	103

1. Einführung

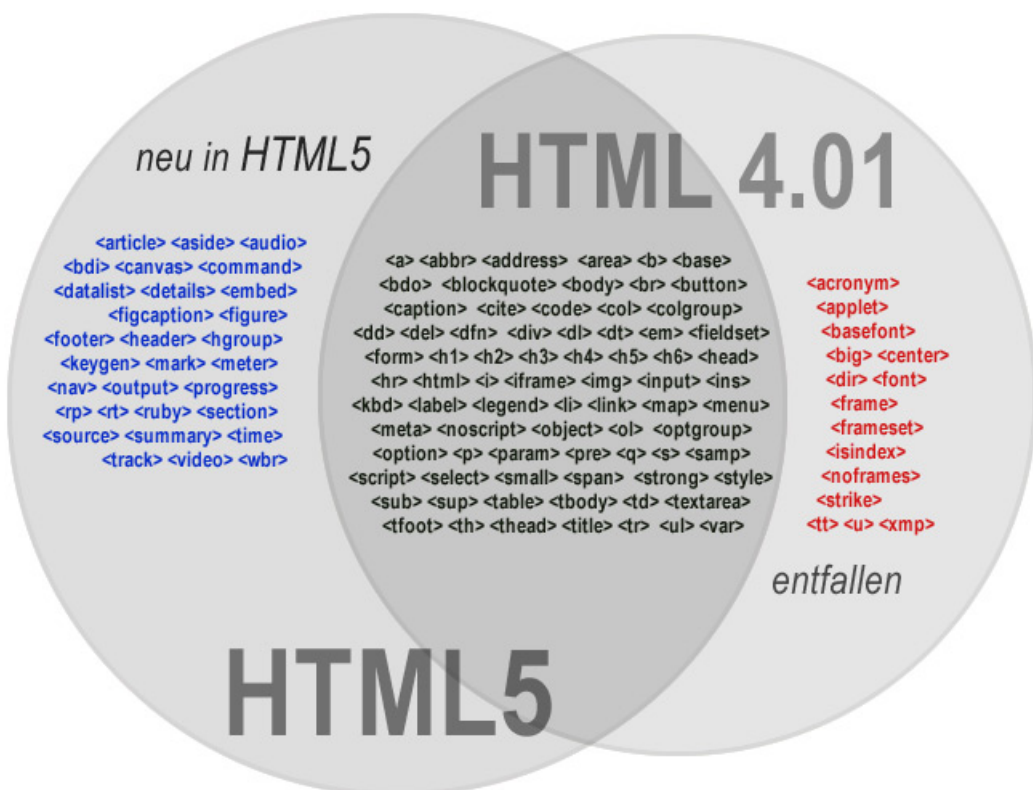
1.1. Was ist HTML5?

HTML5 ist die Antwort auf aktuelle Anforderungen des Webdesigns, die sich aus den Forderungen von **Rich Internet Applications** ergeben. Dies verringert die Erfordernis von Plugins zur Einbindung interaktiver Medien und berücksichtigt gleichzeitig Aspekte der Barrierefreiheit (ARIA).



HTML5 – die Markupsprache

HTML5 tritt dabei als **Markupsprache** die direkte Nachfolge von HTML 4.01 und XHTML 1.0 an. Um die **Abwärtskompatibilität** zu wahren übernimmt HTML5 alle Elemente aus HTML 4.01 und lässt lediglich überkommene und veraltete Tags und Attribute außen vor.



HTML5 – die API-Sammlung

Zum Anderem umreißt der Begriff HTML5 im weiteren Sinne auch eine **Bündelung verschiedener APIs** und anderer Aspekte, die zeitgenössische Webanwendungen ausmachen.



Hiermit dient die Markupsprache als Schnittstelle nicht nur zur Einbindung sondern auch (mittels JavaScript) zur Steuerung von Audio, Video, SVG- oder Canvasgrafiken, sowie für Dienste wie (unter anderem) Geolocation, lokale Datenspeicherung und erweiterte Ajax-Funktionalität. Neben SVG können auch andere XML-Formate wie MathML unmittelbar in HTML5-Dokumente eingebunden werden.

Zusammenfassend zeichnet sich HTML5 gegenüber HTML 4.01 und XHTML 1.0 folgendermaßen aus:

- ✓ Beibehaltung des bisherigen Vokabulars zwecks Abwärtskompatibilität
- ✓ Hinzufügung neuer Elemente, um zeitgemäßen semantischen Forderungen gerecht werden zu können
- ✓ Hinzufügen neuer Elemente, um zeitgemäßen Formen der Medieneinbindung gerecht werden zu können
- ✓ Schaffung von Schnittstellen, welche der Mediensteuerung und der Anbindung neuer Dienste an HTML5-Dokumente ermöglichen

1.2. Inhalt des Seminars

Thema dieses eintägigen Seminars sind primär die **Markup-Aspekte von HTML5**. Behandelt werden also vordringlich die dort hinzugekommenen Elemente und Attribute und es wird auf Änderungen eingegangen, die sich aus der Abschaffung von Elementen und Attributen aus HTML 4.01 bzw. deren abweichender Interpretation ergeben.

Die Satelliten-Spezifikationen von HTML5 sowie deren Schnittstellen können nur ansatzweise im Rahmen eines eintägigen Seminars behandelt werden – so kann zwar beispielsweise auf die Einbettung von SVG und MathML eingegangen, nicht aber auf die Struktur und Erstellung von Inhalten in diesen Sprachen.

Ebenso können die in HTML5 vorgesehenen Schnittstellen zur Steuerung von Medien oder Einbindung von Diensten wie beispielsweise Geolocation gestreift werden, nicht jedoch die Programmierung entsprechender Anwendungen behandelt. In allen Fällen muss zur Vertiefung auf die einschlägigen Spezifikationen verwiesen werden.

✓ Einen Überblick über die Spezifikationen zu HTML5 finden Sie im Anhang.

2. Die Dokumentstruktur

Die Grundstruktur eines HTML-Dokuments ist in HTML5 gegenüber den Vorgängerversionen HTML 4.01 bzw. XHTML 1.0 nicht wesentlich verändert, eher lediglich vereinfacht worden. Die bereits bekannten Strukturen sind jedoch in HTML5 nach wie vor gültig, werden aber im Detail durch einen HTML5-Parser gegebenenfalls ignoriert. Dies ermöglicht die Weiterverwendung von **Legacy-Markup** im Umfeld von HTML5.

Eine einfache Grundstruktur in HTML5 sieht wie folgt aus:

```
<!DOCTYPE HTML>
<html lang="de">
<head>
  <meta charset="UTF-8">
  <title>HTML5 Document</title>
</head>
<body>
```

```
<p>Ein einfaches Dokument nach HTML5-Standard.</p>
</body>
</html>
```

- ✓ Nach wie vor tragen die vorgeschriebenen, obligatorischen Strukturelemente eines HTML-Dokuments die Namen HTML, HEAD, TITLE und BODY.

Zum Vergleich ein **inhaltlich analoges** Dokument, das in *XHTML 1.0 transitional* abgefasst ist – dieses Vokabular entspricht im Prinzip einer Untergruppe von HTML5 (und ist in dem Sinne semantisch gültiges HTML5, als dass alle dort als überflüssig erachteten Bestandteile ignoriert würden). Die Zeilenumbrüche in der Doctype-Declaration sind satzbedingt:

```
<!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">
<head>
    <meta http-equiv="Content-Type"
        content="text/html; charset=UTF-8" />
    <title>XHTML 1.0 Dokument</title>
</head>
<body>
    <p>Ein einfaches Dokument in XHTML.</p>
</body>
</html>
```

Die Vereinfachungen in HTML5 gegenüber XHTML sind augenfällig. Betrachten wir sie im Detail.

2.1. Die Doctype-Deklaration

Eine besonders starke Vereinfachung betrifft die Dokumenttyp-Deklaration, die in HTML5 lediglich aus einer Zeile besteht:

```
<!DOCTYPE html>
```

Weder eine Versionsnummer, noch ein URL zu einer DTD wird hier genannt; lediglich der Bezeichner des Wurzelements wird konstatiert.

- ✓ Dies genügt, da die Deklaration nicht mehr „per se“ einer Validierung des Dokuments dient, worin deren ursprüngliche Aufgabe lag.

In der Hauptsache verwendeten (und verwenden) Browser die Doctype-Deklaration seit Jahren im Rahmen des sogenannten **Doctype-Sniffing**, um die geeignete Interpretation des Markups zu bestimmen.

Hiermit soll aktueller (standardkonformer) HTML-Code von (veraltetem oder ungültigem) Legacy-Code abgesondert werden. Entsprechend schaltet der Browser (mit ungewissem Ausgang) zwischen einem „Standards-Mode“ (aktueller Code) und einem „Quirks-Mode“ (alter Code) um. Hieraus ergeben sich teilweise erhebliche Darstellungsunterschiede.

- ✓ In allen aktuellen Browsern bewirkt das Vorhandensein der HTML5-Deklaration eine Verarbeitung des Dokuments im **Standards-Mode**.

2.2. Das Root-Element

Das Wurzelement von HTML5 heißt nach wie vor HTML und folgt hierin der aus SGML stammenden Tradition, dass das Root-Element den gleichen Namen wie die Sprache zu tragen hat. An dieser Stelle muss gesagt werden, dass HTML5 hingegen **nicht als von SGML abgeleitet** gilt, sondern (unter Einbeziehung des bislang gültigen Vokabulars) als quasi neue Sprachfamilie betrachtet wird. Es existiert daher keine DTD.

- ✓ Der Name des Wurzelements lautet nach wie vor `<html>`

Zur Erinnerung: In XHTML trägt das Wurzelement eine Namensraum-Deklaration `xmlns`, sowie das `xml:lang`-Attribut (und optional zusätzlich auch ein `lang`-Attribut):

```
<html xmlns="http://www.w3.org/1999/xhtml"
      lang="de" xml:lang="de">
```

In HTML5 ist das Wurzelement vereinfacht wie folgt:

```
<html lang="de">
```

Sowohl `xml:lang`- als auch `xmlns`-Attribut sind optional.

2.2.1. Der Namensraum

In XHTML ist das Setzen eines Namensraums *notwendig*, da XML-Elemente von Haus aus in *keinem* Namensraum sind. Um einen XHTML-Tag als „der Sprache HTML zugehörig“ zu kennzeichnen, *muss* daher dieser Namensraum deklariert werden:

`http://www.w3.org/1999/xhtml`.

In HTML 4.01 und dessen Vorgängern existiert *kein Namensraum*, da dieses Konzept in SGML nicht besteht. Eine Namensraumdeklaration ist daher *sinnlos* (und kann auch nicht vorgenommen werden). Hier also das entsprechende Wurzelement in HTML 4.01 – wie Sie sehen, gleicht es dem in HTML5:

```
<html lang="de">
```

In HTML5 ist eine explizite Namensraum-Deklaration nicht erforderlich, das Konzept des Namensraums *existiert* hingegen sehr wohl (ein feiner, aber bedeutsamer Unterschied!):

- ✓ Alle HTML5-Elemente sind **defaultmäßig** im HTML-Namensraum und dieser lautet nach wie vor "`http://www.w3.org/1999/xhtml`".

Eine `xmlns`-Deklaration gilt daher zwar nicht als semantischer Fehler, ist aber schlicht überflüssig.

2.2.2. Das lang-Attribut und die Sprachdeklaration

Die **natürliche Sprache**, in der die Inhalte des Dokuments abgefasst sind, wird mittels des `lang`- bzw. `xml:lang`-Attributs definiert. Dies kann dem Browser beim Lokalisieren helfen, wird von ihm ansonsten aber in der Regel nicht weiter beachtet (für Suchmaschinen kann die Information hingegen eine wertvolle Hilfe beim Katalogisieren darstellen).

Seit XHTML 1.0 wird das `xml:lang`-Attribut für diesen Zweck eingesetzt, das `lang`-Attribut kann (und dann in jedem Fall mit gleichem Wert) *zusätzlich* eingesetzt werden. (Nur im ausgesprochen ungebräuchlichen XHTML 1.1 darf ausschließlich nur `xml:lang` verwendet werden – dies kann als Grenzfall getrost ignoriert werden.)

Dies ist demnach ein Wurzelement für ein XHTML 1.0-Dokument:

```
<html lang="de" xml:lang="de">
```

Beachten Sie die Redundanz – beide Attribute dienen dem selben Zweck, nämlich der Benennung der Sprache des Dokuments. In HTML5 gilt nur das `lang`-Attribut:

```
<html lang="de">
```

- ✓ Lassen Sie `xml:lang` getrost weg und Ihr Dokument wird in HTML5, aber ebenso in HTML 4.01 und XHTML 1.0 gültig sein:

2.3. Das <head> Element

Das Head-Element eines HTML5-Dokuments enthält, neben dem obligatorischen Title-Container, wie seit jeher gewohnt, die **Metadaten**, die der Beschreibung der Datei dienen, außerdem Links zu externen Ressourcen wie Stylesheets und Scripten, sowie gegebenenfalls lokale CSS- und Script-Blöcke. So sieht dies typischerweise aus (XHTML):

```
<head>
  <title>Mein Blog</title>

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

  <link rel="stylesheet" type="text/css"
        href="mein_style.css" />

  <link rel="alternate" type="application/atom+xml"
        title="RSS Feed" href="/meinfeed/" />

  <link rel="search"
        type="application/opensearchdescription+xml"
        title="Open search" href="opensearch.xml" />

  <link rel="shortcut icon" href="/favicon.ico" />

  <script type="text/javascript"
        src="file.js"></script>
</head>
```

2.3.1. Meta-Elemente in HTML5

Betrachten wir das Meta-Element, das in HTML und XHTML dazu dient, den **Character-Code** des Dokuments zu bezeichnen:

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

Der Browser wird angewiesen, das Dokument zu behandeln, als ob es im UTF8-Format gespeichert ist (was dann auch wirklich der Fall sein sollte!).

In HTML5 ist dies auch wünschenswert, aber leichter formulierbar:

```
<meta charset="utf-8" />
```

✓ Praktischerweise funktioniert diese Schreibweise in allen Browsern.

2.3.2. Das <script>-Element in HTML5

Ebenfalls vereinfacht ist der Gebrauch des Script-Elements. Die bisherige Schreibweise bezeichnet das `type`-Attribut als obligatorisch, obwohl in der Praxis andere Programmiersprachen neben JavaScript höchst ungebrauchlich sind. Es *muss* bislang wie folgt geschrieben werden:

```
<script type="text/javascript" src="file.js"></script>
```

In HTML5 ist der Script-Tag auch unter Weglassung des `type`-Attribut legal. Wir können die obere Anweisung also verkürzen:

```
<script src="file.js"></script>
```

✓ Dies ist in jedem Browser erfolgreich, da JavaScript *grundsätzlich* den Defaulttyp für clientseitiges Scripting darstellt! (Soll eine andere Sprache verwendet werden, kann dies immer noch per `type`-Attribut bekannt gegeben werden.)

2.3.3. Das <link>-Element in HTML5

Vergleichbar mit dem Script-Tag kann auch der Link-Tag in HTML5 verkürzt werden. Auch hier ist in HTML 4.01 das `type`-Attribut vorgeschrieben:

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

Das `type`-Attribut ist in HTML5 auch in diesem Fall optional. Der Browser kann aus dem `rel`-Attribut die Art der Datei bereits erschließen:

```
<link rel="stylesheet" href="file.css">
```

- ✓ Dies ist naheliegend, weil neben CSS in der Praxis keine anderen Stylesprachen im Webbereich existieren.

2.3.4. HTML5 vs. XHTML5

Da eine Namensraumdeklaration in HTML5 nicht notwendig ist, ist *HTML5* und *XHTML5* an Hand des Wurzelements nicht unterscheidbar.

Lediglich im Dokumentinneren muss, wenn man die XHTML-Variante vorzieht, gemäß der XML-Syntax verfahren werden. Leere Elemente werden intern geschlossen (was in der HTML-Schreibweise falsch ist), Attribute dürfen nicht verkürzt und Attributwerte müssen stets mit Anführungszeichen markiert werden (in der HTML-Schreibweise dürfen sie u.U. weggelassen werden).

Beide Varianten sind in HTML5 gleichermaßen gestattet, sollten aber jeweils dokumentweit konsequent eingesetzt werden.

- ✓ Die XHTML-Schreibweise ist eigentlich nur dann nutzbringend, wenn Ihre Webdokumente im **XML-Kontext**, beispielsweise mit XSLT verarbeitet werden sollen.

Für das Meta-Element sind daher *alle* folgenden Schreibweisen legal.

- Nach **XML-Syntax**, sprich „XHTML5“, mit internem Abschluss:

```
<meta charset="utf-8" />
```

- Nach (vereinfachter) **HTML5-Schreibweise** *ohne* Anführungszeichen:

```
<meta charset=utf-8>
```

- Gemäß **HTML-Syntax** *mit* Anführungszeichen (lediglich „adretter“):

```
<meta charset="utf-8">
```

Ein HTML5-Parser kann mit *all diesen* Schreibweisen umgehen. Er macht von Haus aus also keinen Unterschied zwischen HTML5 und XHTML5.

- ✓ **Folgerung:** Es ist unproblematisch, beispielsweise aus einem XML-Workflow heraus, HTML5-Dokumente „in XML-Syntax“ (aka „XHTML5“) zu erstellen, da der Browser dies genauso akzeptiert.

XHTML5 ist HTML5:

Relevant ist die Auslieferung des Dokuments durch den Server. Wird es als `text/html` ausgeliefert, wird der HTML5-Parser es stets als „HTML5“ betrachten, ungeachtet der intern verfolgten Syntax (wie gesagt, beide Schreibweisen sind legal). Er arbeitet aber nicht als XML-Parser.

- ✓ Es besteht normalerweise kaum ein Grund, ein HTML-Dokument mit einem anderen Mime-Typ auszuliefern. Es steht Ihnen also stets frei, wahlweise HTML- oder XML-Syntax zu verwenden.

„Technisches“ XHTML wird wie folgt erzielt:

Wird ein HTML5-Dokument vom Server hingegen mit dem MIME-Typ `application/xhtml+xml` ausgeliefert, **dann** (und nur dann!) wird es vom HTML5-Parser als XHTML5 verstanden und **als XML behandelt**.

- ✓ In diesem Falle *muss* das Dokument auch der XML-Syntax und den Kriterien der Wohlgeformtheit Genüge tun.

„Technisches“ XHTML benötigen Sie jedoch *nur dann*, wenn Sie Fragmente aus anderen XML-Sprachen in Ihr HTML5-Dokument einbetten müssen, wie beispielsweise MathML oder SVG.

Sobald die in den Browsern implementierten HTML5-Parser hierauf eingerichtet sind, wird auch für den `text/html`-MIME-Typ die Einbettung funktionieren – es handelt sich um ein temporäres Problem. Bereits Firefox 4 unterstützt dies.

- ✓ **Achtung:** Da XHTML derzeit in der Regel als `text/html` ausgeliefert wird (u.a. weil verschiedene Versionen des Internet Explorers den XHTML-MIME-Typ nicht unterstützten), bringt die Forcierung von HTML5 als XHTML5 in der Praxis kaum Mehrwert.

3. Neue semantische Elemente

- ✓ HTML5 übernimmt, mit wenigen Ausnahmen das Tagvokabular aus HTML 4.01 und XHTML 1.0 und fügt diesem neue Elemente hinzu.

Alle neuen Elemente sind semantischer Natur, beschreiben also ihren Inhalt nach logischen und nicht nach Darstellungskriterien (ein Beispiel für ein *nicht-semantisches* Element wäre der berüchtigte ``-Tag, der in HTML5 aus gutem Grund abgeschafft wurde).

Elemente zur Beschreibung der Dokumentstruktur

Sie erweitern die Möglichkeiten der Beschreibung einer Informationsstruktur durch genauere Umreißung der Rolle einer Einzelinformation im Gefüge. Es ist nun möglich Teilen des Dokuments die Rolle als Header, Footer, Artikel oder Sektion zuzuweisen. Bislang war dies nur indirekt durch Vergabe von entsprechend benannten Identifiern oder Klassen an eigentlich neutrale Div-Elemente möglich:

Vorher: `<div class="article">Ein Artikel</div>`

Jetzt: `<article>Ein Artikel</article>`

All diese Elemente werden gewöhnlich auf der Blockebene verwendet.

Elemente zur semantischen Beschreibung von Inhalten

Auch für die Inlineebene werden neue Elemente zur Verfügung gestellt, die bisher nicht unterstützte Semantik berücksichtigen, wie das `<mark>`-Element, das Textmarkierungen kennzeichnet, oder das `<time>`-Element, das Zeitangaben enthält. Weitere Elemente erleichtern die Einbindung von Medien oder vergrößern das Funktionsspektrum von Formularen.

Vorher: `Hervorhebung`

Jetzt: `<mark>Hervorhebung</mark>`

Der semantische Tag `<mark>` bietet den Vorteil, einer (denkbaren) Defaultpräsentation für Hervorhebungen, wo man bisher grundsätzlich auf eine zu definierende CSS-Anweisung (wie hier auf `.highlight`) verweisen musste: Die CSS-Klasse hingegen hat lediglich eine „pseudosemantische“ Eigenschaft; sie kann zwar die Bedeutung illustrieren, ist ansonsten jedoch beliebig gewählt.

3.1. Dokumenthierarchie – die „Outline“

Die Gliederungsansicht eines Dokuments wird auch als dessen „**Outline**“ bezeichnet. Diese unterteilt ein Dokument in (explizite oder implizite) Abschnitte („Sektionen“), von denen jeder eigene Überschriften, potentiell sogar individuelle Kopf-, Fuß- und Marginalbereiche besitzen könnte. Hierbei könnte eine Überschriftenhierarchie jeweils abschnittbezogen und nicht (oder zusätzlich) dokumentbezogen zu realisieren sein.

Problem:

All dies sind Anforderungen, denen ein Autor mit den Möglichkeiten von HTML 4.01 auf *semantischer Ebene* nicht gerecht werden konnte – wenn es auch gewisse Konventionen und „Best Practices“ gab, um in deren Nähe zu kommen.

- ✓ Beispielsweise wurden und werden Abschnitte und Bereiche durch Div-Container umrissen, sind so aber nicht auf semantischer Ebene gekennzeichnet (siehe folgende Grafik).



Abb.: Struktur HTML 4.01 (Quelle d. Abb: www.alistapart.com)

In HTML5 wird diesem Umstand (wie ursprünglich für XHTML 2.0 geplant) Rechnung getragen und eine Reihe hierfür geeigneter Elemente eingeführt.

- ✓ Hier sind in erster Linie `<article>` und `<section>` zu nennen, aber auch die Elemente `<header>`, `<footer>`, `<nav>`, `<aside>` und `<hgroup>`.

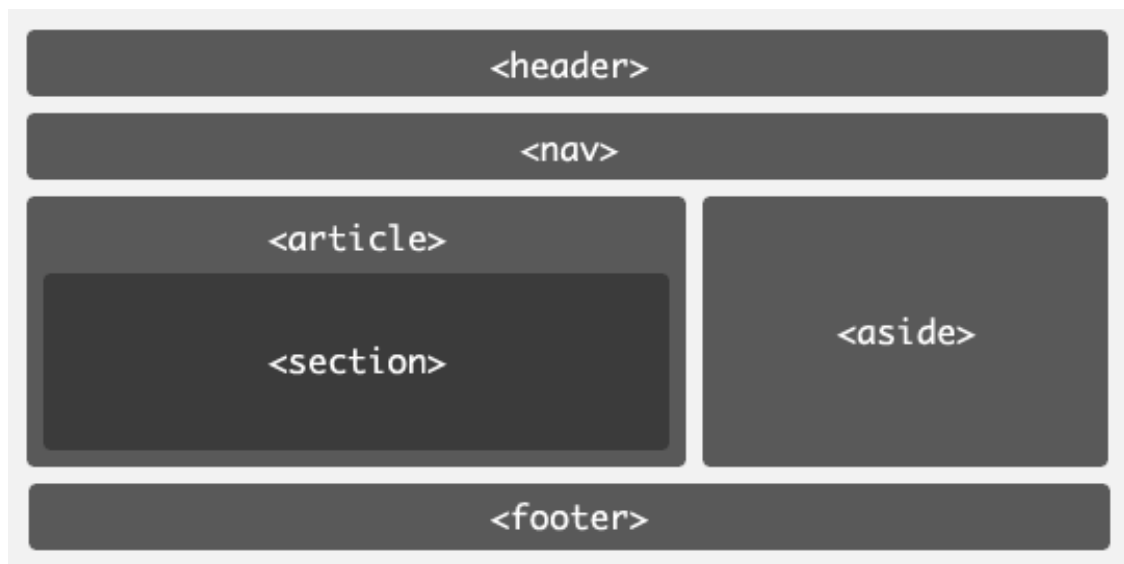


Abb.: Struktur HTML5 (Quelle d. Abb.: www.alistapart.com)

In HTML5 ist es nun möglich, einzelne Dokumentabschnitte separat mit Überschriftenhierarchien, Kopf- und Fußbereichen zu versehen – jeder Abschnitt darf also, zusätzlich zu einer Hauptüberschrift für das gesamte Dokument, seine eigene H1-Überschrift besitzen.

- ✓ In HTML 4.01 war man aufgrund der ungenügend definierten Outline-Semantik auf ein H1 pro Dokument beschränkt.

HTML5 behandelt hierbei `<article>` und `<section>` gleich, indem beide Elemente als Dokumentstrukturen behandelt werden, die als **eigenständige Abschnitte** betrachtet werden können.

- ✓ Durch die neuen Möglichkeiten der Strukturierung sind die Tags H2 bis H6 theoretisch überflüssig.

```
<!-- HTML 4: implizite Outline: -->
<h1>Überschrift Ebene 1</h1>
<p>Textabsatz Ebene 1</p>
<h2>Überschrift Ebene 2</h2>
<p>Textabsatz Ebene 2</p>
<h3>Überschrift Ebene 3</h3>
<p>Textabsatz Ebene 3</p>
```

... wird somit in HTML5 zu

```
<!-- HTML5: explizite Outline: -->
<h1>Überschrift Ebene 1</h1>
<p>Textabsatz Ebene 1</p>
<section>
  <h1>Überschrift Ebene 2</h1>
  <p>Textabsatz Ebene 2</p>
  <section>
    <h1>Überschrift Ebene 3</h1>
    <p>Textabsatz Ebene 3</p>
  </section>
</section>
```

Die Elemente H1 - H6 sind weiterhin für die Überschriftenebenen zuständig, allerdings beginnt die Zählung jedesmal aufs Neue, wenn ein SECTION-Element geöffnet wird. Hierbei wird jeweils eine Hierarchie-Ebene tiefer angesetzt.

Die **Outline** des voranstehenden Fragments sieht so aus:

- 1. Überschrift Ebene 1
 - 1. Überschrift Ebene 2
 - 1. Überschrift Ebene 3

Die Outline Ihres Dokuments können Sie mit Hilfe eines online verfügbaren Tools betrachten, des **HTML5-Outliners**. Der Outliner gleicht einem Validator, indem er einen URL, einen Fileupload oder einen Direkinput von Quelltext akzeptiert. Er generiert aus dem übergebenen HTML eine hierarchische Outline, die die Gliederungsebene widerspiegelt.

HTML5-Outliner: <http://gsnedders.html5.org/outliner/>

3.1.1. Das <section> Element

Eine "Section", wie sie durch den <section>-Tag beschrieben wird, kann am ehesten mit einem **logischen Abschnitt** eines Textes, also einem „Kapitel“ bzw. einer Gliederungsebene, verglichen werden.

Jede Section *kann* ihren eigenen Kopf und Fußteil besitzen, **soll** aber zumindest eine Überschrift besitzen.

- ✓ Es ist zu beachten, dass eine Sektion **kein Ersatz für den Div-Container** ist, also nicht einfach an dessen Stelle tritt. Besitzt ein

Abschnitt keine Überschrift, so soll ein Div-Container eingesetzt werden, um den Abschnitt zu kennzeichnen.

```
<!-- Eine Sektion besitzt eine Überschrift: -->
<section>
<h1>Eine beliebige Überschrift</h1>
... der Inhalt des Abschnitts...
</section>
```

Faustregeln für den Einsatz von `<div>` vs. `<section>`:

- Dient ein Container ausschließlich zum Binden von CSS an einen Dokumentabschnitt, so verwendet man an dieser Stelle `<div>`.
- Markiert der Container eine explizite Gliederungsebene, so verwendet man `<section>`. Es beginnt eine neue Überschriftenhierarchie.
- Sofern zu Beginn des zu bildenden Abschnitts keine Überschrift steht, verwendet man an dieser Stelle `<div>`.
- Andere Tags wie `<article>`, `<aside>` oder `<nav>` vorziehen, wo semantisch angebracht.

3.1.2. Das `<article>` Element

Ein „Artikel“ ist laut offizieller Lesart ein Dokumentbereich, der auch alleinstehend Sinn macht (hier wird regelmäßig der Vergleich mit einem Eintrag eines RSS-Feeds angestellt) – im Gegensatz zur Section, die eher einem Unterabschnitt (aka „Kapitel“) in einem längeren Informationsblock gleichkommt.

✓ Der BODY eines Dokuments entspricht einem impliziten „Artikel“.

Bei einem „Artikel“ macht es Sinn, dass er über eigenständige Überschriftenhierarchien (inklusive einer H1), sowie über Kopf, Fuß- und Marginalbereiche verfügen kann.

Üblicherweise erhält in HTML 4.01 das Dokument selbst die H1, sodass für „Artikel“ jeweils bestenfalls H2 zur Verfügung stehen. Hier sind in einem HTML-Dokument eine Reihe von Feedeinträgen enthalten:

```
<div class="feedeintrag">
  <p class="erschienen">1. Mai 2011</p>
  <h2>
    <a href="#"
      rel="bookmark"
      title="auf Eintrag verweisen">
      Reisetagebuch
    </a>
  </h2>
  ...
</div>
```

Ersetzt man den Div-Container mit CSS-Klasse durch ein `<article>`-Element, so kann jedem ARTICLE (der jetzt auch semantisch als Artikel gekennzeichnet ist) eine eigene H1 gegeben werden. Diese kann zusammen mit dem VÖ-Datum in einen Artikelkopf geschrieben werden:

```
<article>
  <header>
    <p class="erschienen">1. Mai 2011</p>
    <h1>
      <a href="#"
        rel="bookmark"
        title="auf Eintrag verweisen">
        Reisetagebuch
      </a>
    </h1>
  </header>
  ...
</article>
```

Hüllt man das Datum im Textabsatz des Headers noch in den hierfür gedachten `<time>`-Tag (siehe dort), so schöpft man die Möglichkeiten von HTML5 noch besser aus:

```
<article>
  <header>
    <p class="erschienen">
      <time datetime="2011-05-01" pubdate>
        1. Mai 2011</time>
    </p>
    <h1>
      <a href="#"
        rel="bookmark"
        title="auf Eintrag verweisen">
```

```

        Reisetagebuch
    </a>
</h1>
</header>
...
</article>

```

3.1.3. Das <aside> Element

Das <aside>-Element ist für „Nebeninformationen“ im Sinne einer **Marginalie** (oder auch einer Sidebar) gedacht. Hierbei kann es sich um Nebeninformationen zu einem Artikel oder Blogbeitrag handeln, aber auch außerhalb eines Artikels global auf die ganze Webseite beziehen.

- ✓ Nach aktueller Lesart kann <aside> am ehesten mit einem sogenannten „Pullquote“ verglichen werden, also einem erläuternden „Nebentext“, der sich auf eine Position im Haupttext bezieht.

Ein <aside>-Block kann also einem Dokumentabschnitt wie einem Artikel zugeordnet sein

```

<article>
  <h1>Blogeintrag</h1>
  <p>Text des Eintrags....</p>
  <aside>
    <!-- Im Artikel
           bezieht sich ASIDE auf den umliegenden Artikel.
    -->
    <h1>Linkliste</h1>
    <ol>
      <li><a href="#">Thema 1</a></li>
      <li><a href="#">Thema 2</a></li>
    </ol>
  </aside>
</article>

```

Ebenso kann <aside> auf Dokumentebene eingesetzt werden und in diesem Falle als Kontextspalte oder „Sidebar“ interpretiert werden:

```

<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="UTF-8">

```

```
<title>Beispiel für Aside-Element</title>
</head>
<body>
  <header>
    ...
  </header>
  <article>
    ...
  </article>
  <aside>
    <!-- Auf der obersten Gliederungsebene
         bezieht sich ASIDE auf das Gesamtdokument.
    -->
    ...
  </aside>
  <footer>
    ...
  </footer>
</body>
</html>
```

✓ **Achtung – nicht im Sinne von „Layout“ mißzuverstehen:**
Auf welche Art eine in `<aside>` untergebrachte „Nebeninformation“ dargestellt wird – ob als Sidebar oder anders – ist ausdrücklich *nicht* festgelegt. Das Element bezieht sich in seiner Semantik auf die Darstellung einer **Relation zum Hauptinhalt**, nicht auf den layouttechnischen Einsatz.

Hinweis: Prinzipiell wird der ASIDE-Inhalt als für das Verständnis des Hauptinhalts, auf den Bezug genommen wird, als „entbehrlich“ betrachtet. Mobilgeräte könnten daher ASIDE-Container möglicherweise aus dem Layout entfernen, wenn dies erforderlich ist.

Wollen Sie ein ASIDE mit einer **ARIA-Rolle** korrekt beschreiben, so können Sie ein `role`-Attribut mit der Rolle „complementary“ setzen:

```
<aside role="complementary">
  <p>Korrekt per ARIA-role ausgezeichnet...</p>
</aside>
```


3.1.4. Das <main>-Element

Auch für den „Hauptinhalt“ des Dokuments existiert ein explizites Markup-Element, namens `<main>`. Es soll definitiv nur **ein einziges Mal** erscheinen, und dann auf der *obersten* Outline-Ebene (also *nicht* als Inhalt anderer Sektionen oder Artikel, bzw. von Header-, Footer- oder Nav-Bereichen).

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="UTF-8">
  <title>Beispiel für Main-Element</title>
</head>
<body>
  <header>
    ...
  </header>
  <main role="main">
    <h1>Der Hauptinhalt des Dokuments</h1>
    <p> ... </p>
  </main>
  <footer>
    ...
  </footer>
</body>
</html>
```

Im obigen Beispiel enthält der Main-Container eine H1-Überschrift. Es ist wichtig zu wissen, dass `<main>` nicht als „Sectioning-Container“ gilt, also transparent für die Outline ist - ein `<section>`-Element an gleicher Stelle würde die enthaltene Überschrift um eine Ebene in der Outline herabstufen. In diesem Fall definiert die H1 also die oberste Outline-Instanz mit dem Body als „Sectioning Root“.

- ✓ Sie können die H1 natürlich auch im Header-Element des Dokuments platzieren. Der Main-Inhalt könnte dann durch Sections partitioniert werden und sollte in diesem Fall *keine* eigene H1 besitzen.

Zu beachten ist das `role`-Attribut, das im Beispiel explizit gesetzt ist, und das die **ARIA-Rolle** des Elements als „*main content*“ deutlich macht. Diese Rolle entspricht der Default-Rolle. Wenn Sie ein `role`-Attribut setzen, muss es daher mit diesem Wert geschehen.

- ✓ Das `role`-Attribut wird aus Gründen der **Abwärtskompatibilität** für diejenigen Browser gesetzt, die `<main>` nicht kennen und daher die Default-ARIA-Rolle *nicht zuweisen*. Im Prinzip ist es also redundant.

3.1.5. Das `<header>` Element

Ein mit `<header>` bezeichneter Bereich enthält Informationen, die am Anfang eines Dokuments, eines Artikels oder einer Sektion stehen sollen. Hierzu gehören beispielsweise Überschriften oder ein einleitender Absatz, wie ein „Abstract“, aber auch beliebige Inline-Elemente.

Natürlich konnte man auch in HTML 4.01 einen „Kopfbereich“ festlegen, allerdings nicht mit zwingender semantischer Bedeutung:

```
<div class="blogeintrag">
  <div class="artikelheader">
    <h2>Mein Blogeintrag</h2>
    <p class="abstract">
      Hier ist eine Zusammenfassung...</p>
    </div>
    ...
  </div>
```

Markiert man den Blogeintrag als Artikel, kann man die Headerinformationen in einen `<header>`-Container zusammenfassen:

```
<article>
  <header>
    <h1>Mein Blogeintrag</h1>
    <p class="abstract">
      Hier ist eine Zusammenfassung...</p>
    </header>
    ...
</article>
```

- ✓ Das `<header>`-Element ermöglicht, *zusätzlich* zum eigentlichen Überschriften-Container *Hx* weitere Inhalte quasi in der Überschriftenrolle einzusetzen (im obigen Beispiel ein P-Tag in der Meta-Rolle „abstract“). Ist dies nicht erforderlich, genügt also ein einfacher *Hx*-Tag, so ist der Header-Container nicht erforderlich.

Auch auf der obersten Outline-Ebene des Dokuments darf das `<header>`-Element verwendet werden. Wenn Sie ein HEADER-Element in diesem Sinne als „Dokument-Header“ einsetzen, können Sie ihm per `role`-Attribut die **ARIA-Rolle** „banner“ zuweisen, die seine Aufgabe im Dokumentkontext verdeutlicht:

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="UTF-8">
  <title>Beispiel für Header-Element</title>
</head>
<body>
  <header role="banner">
    <h1>Das Thema dieses Dokuments ist... </h1>
  </header>
  <main> ... </main>
  <footer> ... </footer>
</body>
</html>
```

3.1.6. Das `<hgroup>`-Element

Überschriften können, innerhalb oder außerhalb eines Headers, in ein `<hgroup>`-Element eingeschlossen werden, sofern eine Kombination aus mehreren Überschriften auftritt und das damit automatisch verbundene **implizite Outlineverhalten** unerwünscht ist.

Besitzt ein Abschnitt nur *eine* Überschrift, wird man üblicherweise eine H1 verwenden (die Outline-Ebene ergibt sich implizit):

```
<article>
  <h1>Titel des Artikels</h1>
  <p>Text des Artikels...</p>
</article>
```

Zwei aufeinanderfolgende Überschriften stellen zunächst zwar kein *semantisches* Problem dar, solange die hierarchisch vorgegebene Reihenfolge eingehalten wird:

```
<article>
  <h1>Titel des Artikels</h1>
  <h2>Untertitel des Artikels</h2>
```

```
<p>Text des Artikels...</p>
</article>
```

... praktisch jedoch befindet man sich im P nach der H2 jedoch eine Gliederungsebene *tiefer*, als eigentlich beabsichtigt! Die erste Ebene besteht nur aus der H1-Überschrift (hier greift also die implizite Outline).

Manchmal will man mit zwei Überschriften für *dieselbe* Gliederungsebene arbeiten und diese vielleicht obendrein vertauschen – hier soll, der Untertitel optisch über (im Quelltext also *vor*) der Hauptüberschrift erscheinen. Diese Umsetzung jedoch ist bereits *semantisch* problematisch:

```
<article>
  <h2 class="subheader">Untertitel des Artikels</h2>
  <h1 class="mainheader">Titel des Artikels</h1>
  <p>Text des Artikels...</p>
</article>
```

Auch die gut gemeinte zusätzliche Rollenkennzeichnung durch CSS-Klassen hilft nicht weiter: Es muss „gruppiert“ werden.

✓ **Gruppiert** man mehrere Überschriften mit dem `<hgroup>`-Element, richtet sich die semantische Bedeutung der Gruppe nach ihrer *höchstrangigen* Überschrift, egal, wo diese im Container steht.

Ein `<hgroup>`-Element mit einer H1 gilt also stets *wie eine H1*, auch wenn vor (oder nach) der enthaltenen H1 noch andersrangige Überschriften stehen sollten. Die semantische Rangordnung ist somit (künstlich) wiederhergestellt:

```
<article>
  <hgroup>
    <h2>Untertitel des Artikels</h2>
    <h1>Titel des Artikels</h1>
  </hgroup>
  <p>Text des Artikels...</p>
</article>
```

Im folgenden Beispiel übernimmt das `<hgroup>`-Element semantisch die Rolle einer H2-Überschrift:

```
<section>
  <hgroup>
    <h3>Untertitel</h3>
    <h2>Haupttitel</h2>
  </hgroup>
  <p>Text des Artikels...</p>
</section>
```

```
        <h4>Nebentitel</h4>
    </hgroup>
</section>
```

- ✓ **Merke:** In der Outline erscheint nur die **ranghöchste** Überschrift einer `<hgroup>`-Überschriftengruppe.

3.1.7. Das `<footer>`-Element

Meist am Ende eines Artikels oder einer Webseite steht das FOOTER-Element. Es enthält Auskünfte über Autoren, das Copyright oder weitere Informationen, die zur Webseite oder dem Artikel gehören (Sektionen erhalten normalerweise keinen Footer, obwohl dies durchaus erlaubt ist. Nebenbei, für Kontaktinformationen im Rahmen eines Footers sollten Sie das `<address>`-Element einsetzen.)

- ✓ **Hinweis:** Ein Footer leitet keine neue Gliederungsebene ein, ist also kein „Sectioning-Content“. Er darf allerdings seinerseits „Sectioning-Content“ enthalten, wie Abschnitte in Form von Sektionen oder Navigationen (dabei aber *keinen* eigenen Header und Footer). Abgesehen davon ist sämtlicher Flow-Content gestattet.

Ein Footer muss *nicht* notwendigerweise am Ende seines Abschnittes stehen – es kommt in der Tat nicht auf seine Position im Quellcode, sondern auf seine semantische Rolle an.

- ✓ Verstehen wir den Footer am einfachsten als „sichtbare Metainformation“ zu seinem Abschnitt.

In HTML 4.01 konnte man einen Footer nur „pseudo-semantisch“ kennzeichnen:

```
<div id="footer">
    <p>Beispiel für eine Fußzeile</p>
</div>
```

Stattdessen kann man in HTML5 wie folgt schreiben:


```
<footer>
    <p>Beispiel für eine Fußzeile</p>
</footer>
```

Wenn Sie ein FOOTER-Element als *Dokument-Footer* einsetzen, können Sie ihm per `role`-Attribut die **ARIA-Rolle** „contentinfo“ zuweisen, die seine Aufgabe im Dokumentkontext verdeutlicht:

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="UTF-8">
  <title>Beispiel für Footer-Element</title>
</head>
<body>
  <header> ... </header>
  <main> ... </main>
  <footer role="contentinfo">
    &copy; <time datetime="2014">2014</2014>
  </footer>
</body>
</html>
```

Fat Footer

Dies funktioniert auch mit den heute sehr verbreiteten „Fat Footern“. In Folge ein Beispiel, wie es so ähnlich beim W3C zu finden ist – im Inneren des Footer-Bereichs befindet sich eine dreispaltig gesetzte Navigation, deren erste zwei Spalten mit dem neuen `<nav>`-Tag gekennzeichnet sind und die jeweils über eine Überschrift verfügen.

- ✓  **Erinnern wir uns, dass `<nav>` eine „spezialisierte Sektion“ ist, die eine Navigation enthält, aber auch eine eigene Überschrift besitzen kann. (Eine genaue Beschreibung folgt sogleich.)**

Dieser Footer enthält entsprechend drei explizite Abschnitte, von denen zwei als `<nav>` und einer als `<section>` vorliegen, sowie einen abschließenden Textabsatz `<p>` auf oberster Ebene. (Auch der Footer selbst könnte eine eigene Überschrift besitzen.)

```
<footer>

  <nav>
    <h1>Navigation</h1>
    <ul>
      <li><a href="/">Home</a></li>
```

```
<li><a href="/standards/">Standards</a></li>
<li><a href="/participate/">Participate</a></li>
<li><a href="/membership">Membership</a></li>
<li><a href="/Consortium/">About W3C</a></li>
</ul>
</nav>

<nav>
  <h1>Contact W3C</h1>
  <ul>
    <li><a href="/contact">Contact</a></li>
    <li><a href="/Help/">Help and FAQ</a></li>
    <li><a href="/sup">Donate</a></li>
    <li><a href="/siteindex">Site Map</a></li>
  </ul>
</nav>

<section>
  <h1>W3C Updates</h1>
  <ul>
    <li><a href="twitter.com/W3C">Twitter</a></li>
    <li><a href="identi.ca/w3c">Identi.ca</a></li>
  </ul>
</section>

<p class="copyright">Copyright © 2011 W3C</p>

</footer>
```

Die Outline des vorstehenden Codeabschnittes sieht so aus:

1. *Untitled Section*
 1. Navigation
 2. Contact W3C
 3. W3C Updates

3.1.8. Das <nav>-Element

Der mit dem <nav>-Tag markierte Dokumentabschnitt enthält eine Navigation, wobei (laut Spezifikation) nur „größere“ oder „wesentliche“

Navigationselemente („*major navigation blocks*“) entsprechend markiert werden sollen.

- ✓ Es dürfen explizit **mehrere** solcher Elemente pro Dokument auftreten, auch innerhalb einer Sektion.

So wäre ein Navigationscontainer in HTML 4.01 ausgezeichnet worden:

```
<div id="nav">
  <ul>
    <li><a href="#">home</a></li>
    <li><a href="#">blog</a></li>
    <li><a href="#">gallery</a></li>
    <li><a href="#">about</a></li>
  </ul>
</div>
```

HTML5 verbessert die semantischen Möglichkeiten und kennzeichnet den Bereich eindeutig als Navigation:

```
<nav>
  <ul>
    <li><a href="#">home</a></li>
    <li><a href="#">blog</a></li>
    <li><a href="#">gallery</a></li>
    <li><a href="#">about</a></li>
  </ul>
</nav>
```

- ✓ Sobald Screenreader das `<nav>`-Element und damit seine Rolle im Dokument verstehen, werden die bisher aus Barrierefreiheitsgründen üblichen Skiplinks überflüssig.

Aside :-)

Derzeit sind Screenreader in der Lage, aus den Überschriften eine navigierbare Outline eines Dokuments zu generieren. In Zukunft *könnten* explizite Navigationselemente hier ebenfalls berücksichtigt werden.

3.2. Inlineelemente

In HTML5 werden eine Reihe neuer Inlineelemente eingeführt. Andere, von HTML 4.01 übernommene, Tags werden in ihrer Rolle umgedeutet.

Hierzu zählen beispielsweise auch die in HTML 4.01 letztthin verpönten ``, `<i>` und `<small>`-Tags, die in HTML5 mit neuer Semantik wiederauferstehen:

- ✓ `<i>` gilt nun als Markup im Sinne von „fremdsprachiger Ausdruck“, „Fachbegriff“ oder „typographisch kursiv gesetzt“.
- ✓ `` dient zur Kennzeichnung von aus „stilistischen Gründen“ hervorgehobenen Passagen, wie „Schlüsselbegriffe“ oder für „typographisch fett gesetzt“.
- ✓ `` verschiebt sich in seiner Bedeutung von „betont“ zu „durch Aussprache hervorgehoben“.
- ✓ `` verschiebt sich von „stark betont“ zu einer „Hervorhebung von Wichtigkeit“.
- ✓ `<small>` dient nicht mehr der rein präsentativen Verkleinerung des Fonts, sondern der Kennzeichnung als „Kleingedrucktes“ (durchaus im juristischen Sinne).

3.2.1. Das `<mark>`-Element

Das `<mark>`-Element steht in HTML5 als neues Inlineelement zur Verfügung, das, neben `` und ``, zur Hervorhebung von Textpassagen dient.

- ✓ Hierbei trägt `<mark>` absichtlich keine semantische Nebenbedeutungen wie „Betonung“ oder „starke Betonung“.

`<p>Man könnte sagen, dass HTML5 durchaus neue Aspekte bietet.</p>`

Ein möglicher Einsatz von `<mark>` besteht darin, Textpassagen für den Leser hervorzuheben, die für ihn von besonderem Interesse sind, beispielsweise im Text auftretende Suchbegriffe.

Dem Element muss allerdings irgendeine Präsentation zugewiesen werden:

```
/* so könnte eine Hervorhebung definiert sein */

mark {
    background-color:#ff9;
    color:#000;
    font-style:italic;
    font-weight:bold;
}
```

- ✓ Gebrauch: `` sollte zur Hervorhebung eingesetzt werden, wenn es um die Kennzeichnung von *Wichtigkeit* einer Passage geht, `` zur *Betonung*, wie es in der gesprochenen Rede geschieht. In allen anderen Fällen kann `<mark>` eingesetzt werden.

3.2.2. Das `<time>`-Element

Bislang konnte man **Zeitangaben** in HTML nicht gesondert markieren sondern war auf pseudosemantische Kennzeichnung über CSS-Klassen angewiesen:

```
<div class="blogeintrag">
  <p><span class="datum">
    1. Mai 2014</span></p>
  <h2>Reisetagebuch</h2>
  ...
</div>
```

In HTML5 existiert für diese Zwecke das Inlineelement `<time>`. Dieses besitzt ein Attribut `datetime`, das ein maschinenlesbares Datumsformat, wie einen Zeitstempel, aufnehmen kann.

```
<article>
  <p><time datetime="2014-05-01">
    1. Mai 2014</time></p>
  <h2>Reisetagebuch</h2>
  ...
</div>
```

In einem `<article>`-Element kann mittels eines `<time>`-Element ein Veröffentlichungsdatum genannt werden. Hierfür muss das Element zusätzlich das (leere) `pubdate`-Attribut erhalten.

- ✓ Jeder Artikel darf mehrere `<time>`-Elemente besitzen, von denen allerdings maximal einer als „publishing date“ fungieren kann.

```
<article>
<p><time datetime="2014-05-01" pubdate>
    1. Mai 2014</time></p>
    <h2>Reisetagebuch</h2>
    ...
</article>
```

4. Formulare in HTML5

Besonders im Bereich Formulare hat sich in HTML5 sehr viel getan. Nicht nur sind die Möglichkeiten bestehender Elemente erweitert worden, es wurden auch vollkommen neue Elemente und Attribute eingeführt bzw. der Wertebereich existierender Attribute erweitert. Dies betrifft vor allem das **Input-Element**.

Durch die Option, die Validierung eines Formular bereits auf HTML-Ebene definieren zu können und hierfür sogar Vergleichspattern in Form **regulärer Ausdrücke** zu erlauben, dringt HTML5 in eine Domäne vor, die bislang JavaScript vorbehalten war.

4.1. HTML5 <form>-Tag

Das Form-Element dient der Eingabe von Daten durch den Nutzer. Es enthält hierfür Eingabeelemente wie Input, Button, Textarea und weitere, wobei das Inhaltsmodell durch neu hinzugekommene Elemente erweitert wurde.

Beispiel für ein einfaches Formular:

```
<form action="form_test.php">
  Vorname:
  <input type="text"
        name="vorname" value="Peter"><br>
  Nachname:
  <input type="text"
        name="nachname" value="Panter"><br>
  <input type="submit"
        value="Abschicken">
</form>
```

Gegenüber HTML 4.01 wurden den Elementen neue Attribute hinzugefügt und die strenge Zuordnung von Inputs gelockert: Es ist nun möglich, ein Input-Element auch *außerhalb* eines Form-Containers zu definieren und diesem mit einem `form`-Attribut per Referenz zuzuordnen.

4.1.1. Attribute des Form-Elements

In HTML5 sind für das `<form>`-Element die folgenden Attribute definiert – gegenüber HTML 4.01 hinzugekommene Attribute sind fett hervorgehoben.

- ✓ Entfallen ist das `accept`-Attribut, das nur noch dem Input direkt zur Verfügung steht. Für `<form>` recycled wurde hingegen das in XHTML abgeschaffte `name`-Attribut.

Attribut	Wert	Beschreibung
<code>accept-charset</code>	<code>charset_list</code>	Eine Liste der Zeichensätze, die durch den Server unterstützt werden. Dies kann beispielsweise folgende Form haben: <code>accept-charset="ISO-8859-1, ISO-8859-2, ISO-8859-3"</code>
<code>action</code>	URL	Das Ziel des Datenversands
<code>autocomplete</code>	<code>on</code> <code>off</code>	Legt fest, ob die Autocomplete-Funktion des Browsers auf das Formular angewandt werden soll. (Ehemals proprietäres Attribut von IE.) Wird <code>autocomplete="on"</code> gesetzt, darf der Browser im Nutzerprofil gespeicherte Daten zum Füllen der Formularfelder heranziehen. Der Wert <code>autocomplete="off"</code> zwingt den Nutzer, die Eingaben in jedem Fall selbst vorzunehmen.
<code>enctype</code>	<code>application/x-www-form-urlencoded</code> <code>multipart/form-data</code> <code>text/plain</code>	Der Encoding-Typ, mit dem die Daten verschickt werden sollen. Defaultwert ist <code>enctype="application/x-www-form-urlencoded"</code> , bei dem übertragene Textdaten nach <i>RFC1738</i> encodiert werden.
<code>method</code>	<code>get</code> <code>post</code>	Die Methode des Datenversands (QueryString oder HTTP)
<code>name</code>	<code>form_name</code>	Ein Bezeichner vom Typ <i>name</i> , der dem Formular zugewiesen werden kann. Anmerkung: Abgeschafft in XHTML 1.0/1.1. Stattdessen sollte ein ID eingesetzt werden.
<code>novalidate</code>	<code>novalidate</code>	Setzt die Validierung des Formulars außer

		Kraft
target	_blank _self _parent _top framename	Legt einen Frame oder ein Browserfenster fest, in das der Server die Antwortseite ausgeben soll, die aus der Auswertung des action-URL resultiert

Attribute für `<form>` in HTML5

4.2. HTML5 `<input>` Tag

Auch in HTML5 definiert `<input>` ein allgemeines Eingabefeld, dessen Ausprägung durch das `type`-Attribut bestimmt wird. Dem Wertebereich von `type` wurde eine Vielzahl neuer Werte hinzugefügt.

4.2.1. Attribute des `<input>`-Tags

Gegenüber HTML 4.01 besitzt das Input-Element eine Reihe zusätzlicher Attribute.

- ✓ Einige andere Attribute hingegen entfallen: abgeschafft wurden die Attribute `align`, `ismap` und `usemap`.

Attribut	Wert	Beschreibung
accept	list_of_mime_types	Das <code>accept</code> -Attribut enthält eine kommaseparierte Liste all derjenigen Media-Typen (<i>content types</i> gemäß IANA-Registrierung), die der Server in Zusammenhang mit einem File-Upload akzeptieren soll. Das Attribut ist nur in Zusammenhang mit dem Input <code>type="file"</code> von Bedeutung.
alt	text	Das <code>alt</code> -Attribut enthält einen Textstring, der als Platzhalter für einen Grafik-Button mit <code>type="image"</code> dient, falls dieser nicht dargestellt wird.
autocomplete	on off	Legt fest, ob die Autocomplete-Funktion des Browsers auf das Eingabefeld angewandt werden soll.
autofocus	autofocus	Legt fest, dass dieses Feld nach dem Laden der Seite den Fokus erhalten soll. Nicht anwendbar auf <code>type="hidden"</code> .

<i>checked</i>	checked	Das <code>checked</code> -Attribut bestimmt, ob ein Inputelement des Typs <code>Checkbox</code> oder <code>Radiobutton</code> beim Laden des Formulars vorgewählt ist. Defaultzustand ist „nicht vorgewählt“.
<i>disabled</i>	disabled	Das <code>disabled</code> -Attribut deaktiviert ein Formulareingabeelement und unterdrückt die Versendung dessen Werts bei Abschicken der Formulardaten. Gleichzeitig wird durch das Attribut verhindert, dass der User den Zustand des Elements ändert (durch Werteingabe oder Anklicken). Nicht anwendbar auf <code>type="hidden"</code> .
form	formname	<i>IDREFS</i> -Attribut. Bezieht sich auf den ID eines (oder mehrerer) Formulare, dem/denen der Input funktional und datentechnisch zugeordnet wird. <i>Wirkung:</i> Der Input kann damit auch außerhalb des betreffenden Form-Containers liegen.
formaction	URL	Überschreibt den Wert des <code>action</code> -Attributs des Formulars. Nur anwendbar für <code>type="submit"</code> und <code>type="image"</code> . <i>Wirkung:</i> Jeder Submit-Button kann eine individuelle Action erhalten.
formenctype	application/x-www-form-urlencoded multipart/form-data text/plain	Überschreibt den Wert des <code>enctype</code> -Attributs des Formulars. Nur anwendbar für <code>type="submit"</code> und <code>type="image"</code> . <i>Wirkung:</i> Jeder Submit-Button kann ein individuelles Formencoding bewirken.
formmethod	get post	Überschreibt den Wert des <code>method</code> -Attributs des Formulars. Nur anwendbar für <code>type="submit"</code> und <code>type="image"</code> . <i>Wirkung:</i> Jeder Submit-Button kann eine individuelle Versandmethode erhalten.
formnovalidate	formnovalidate	Überschreibt den Wert des <code>novalidate</code> -Attributs des Formulars. Auf alle Typen anwendbar <i>Wirkung:</i> Das betreffende Feld wird von der Validierung in jedem Fall ausgenommen.
formtargetNew	_blank _self _parent _top framename	Überschreibt den Wert des <code>target</code> -Attributs des Formulars. Nur anwendbar für <code>type="submit"</code> und <code>type="image"</code> . <i>Wirkung:</i> Jeder Submit-Button kann ein eigenes Target angeben.
height	pixels %	Die Höhe einer Submitgrafik <code>type="image"</code> .
list	datalist-id	IDREF. Bezieht sich auf den ID eines

		<datalist>-Containers, aus dem der betreffende Input seine Daten holen soll.
max	number date	Legt den Maximalwert für numerische Inputs wie <code>range</code> oder <code>number</code> fest.
maxlength	number	Das <code>maxlength</code> -Attribut gibt für Eingabefelder vom Typ „text“ oder „password“ die maximal akzeptierte Länge des Eingabestrings an und verhindert Eingaben des Nutzers, die diese Länge überschreiten.
min	number date	Legt Minimalwert für numerische Inputs wie <code>range</code> oder <code>number</code> fest.
multiple	multiple	Erlaubt die Eingabe von mehreren Werten durch den Nutzer. Anwendbar auf die Typen <code>file</code> und <code>email</code> .
name	fieldname	Das <code>name</code> -Attribut legt den Bezeichner (<i>control name</i>) des Eingabefeldes innerhalb eines Formulars fest. Der Bezeichner findet bei der Datenübertragung als <code>name</code> -Part eines „name-value“-Pärchens Verwendung.
pattern	regexp_pattern	Legt ein <i>RegExp</i> -Pattern fest, dem die Nutzereingabe in das betreffende Feld Genüge tun muss. Beispiel: <code>pattern="[0-9]"</code> verlangt nach einer numerischen Eingabe zwischen 0 und 9.
placeholder	text	Gibt im Feld einen Hinweistext („Hint“) aus, der bei Klick in das Feld verschwindet. Anwendbar auf textbasierte Typen (gängig bei Typ „search“).
readonly	readonly	Das <code>readonly</code> -Attribut kann eingesetzt werden, um eine Änderung eines Vorgabewertes durch den Nutzer zu unterbinden. Die Anwendung erstreckt sich im Wesentlichen auf Texteingabefelder, die scriptgesteuert Werte ausgeben.
required	required	Bezeichnet die Eingabe in das Feld als obligatorisch. Das Formular wird bei fehlender Eingabe nicht versendet.
size	number of characters	Das <code>size</code> -Attribut gibt für Eingabefelder vom Typ „Text“ oder „Passwort“ die sichtbare Breite des Feldes in Zeichen an.
src	URL	Das <code>src</code> -Attribut nennt in Form eines URL die für einen grafischen Submit-Button vom Typ „Image“ zu zeigende Grafikressource.
step	number	Legt das Änderungsintervall für numerische Inputs wie <code>range</code> oder <code>number</code> fest.
type	button, checkbox, color, date, datetime, datetime-local, email,	Das <code>type</code> -Attribut bestimmt die Art des zu erzeugenden Eingabefeldes. Der Defaultwert in

	file, hidden, image, month , number , password, radio, range , reset, search , submit, tel , text, time , url , week	Abwesenheit des Attributs ist "text". (Neue Werte in der nebenstehenden Spalte fett hervorgehoben.)
value	value	Das value-Attribut bestimmt den Datenwert eines Eingabefeldes. Buttons: Legt die Aufschrift des Buttons fest. Checkbox und Radiobutton: Legt den Datenwert beim Versand fest. Ist für diese Typen obligatorisch. Typ hidden, password und text: Legt den Defaultwert des Feldes fest. <i>Achtung:</i> Nicht einsetzbar für type="file"
width	pixels %	Die Breite einer Submitgrafik type="image".

autofocus-Attribute

Das autofocus-Attribut setzt beim Laden den Focus automatisch in das betreffende Element.

```
<form>
  <input name="suche" autofocus>
  <input type="submit" value="Suche">
</form>
```

placeholder-Attribut

Das placeholder-Attribut setzt einen Platzhaltertext in ein textbasiertes Eingabefeld, der beim Klick in den Input verschwindet. Verlässt der Nutzer den Input ohne eine Eingabe vorgenommen zu haben, so taucht der Text wieder auf:

```
<form>
  <input name="suche"
    placeholder="Website durchsuchen">
  <input type="submit" value="Suchen">
</form>
```

- ✓ Das Attribut kann auch in einer Textarea eingesetzt werden. Wird es vom Browser nicht unterstützt, so ignoriert er es.

required-Attribut

Das `required`-Attribut deklariert ein Eingabefeld als "obligatorisch" mit einem Wert zu befüllen. Das Validierungsverhalten des Formulars lässt ein Verschicken der Daten nicht zu, solange nicht alle als "required" bezeichneten Inputs gefüllt wurden.

```
<form>
  <input type="search" name="suche" required>
  <input type="submit" value="Suchen">
</form>
```

- ✓ Browser können eine Defaultpräsentation für "obligatorische" Felder definieren.

pattern-Attribut

Mit Hilfe der `pattern`-Attributs kann ein regulärer Ausdruck übergeben werden, dem gegenüber die Feldeingabe geprüft wird. Das Validierungsverhalten des Formulars lässt einen Datenversand nicht zu, solange die Prüfung nicht positiv ist. In diesem Zusammenhang wird der `title`-Text für eine Fehlermeldung herangezogen:

```
<label> Bauteilnummer:
  <input pattern="[0-9][A-Z]{3}"
        name="bauteil"
        title="Eine Bauteilnummer besteht aus
              einer Ziffer gefolgt von drei
              Großbuchstaben."
  />
</label>
```

Ein Verstoß gegen die Regel könnte eine Fehlermeldung unter Verwendung des `title`-Textes zur Folge haben:

Beispiel: Inputtyp text (mit pattern)

Bauteil:

Please use the required format
Eine Bauteilnummer besteht aus einer
Ziffer gefolgt von drei Großbuchstaben.

4.2.2. Modernizr und Input-Attribute

- *CSS-Klassen:* Input-Attribute werden nicht über CSS-Klassen widergespiegelt
- *JavaScript-Property:* Modernizr.input[attribute]

4.2.3. Das type-Attribut von <input>

Besonders ins Gewicht fallen die neuen Werte des `type`-Attributs, mit deren Hilfe nun Inputs für unter anderem Datumswerte, Zeit- und Zahlenangaben oder URLs definiert werden können. Neu hinzugekommen sind `color`, `date`, `datetime`, `datetime-local`, `month`, `week`, `time`, `email`, `number`, `range`, `search`, `tel` und `url`.

Wert (Typ)	Beschreibung
<code>button</code>	Ein neutraler Button (zur Steuerung von Scripten)
<code>checkbox</code>	Eine Checkbox
<code>color</code>	Ein Farbwahlfeld (Colorpicker)
<code>date</code>	Ein Datepicker für Jahr, Monat und Tag (ohne Zeitzone)
<code>datetime</code>	Ein Datepicker für Zeitstempelwert (Jahr, Monat, Tag, Uhrzeit) als UTC-Zeit
<code>datetime-local</code>	Ein Datepicker für Zeitstempelwert (Jahr, Monat, Tag, Uhrzeit) in der lokalen Zeitzone
<code>month</code>	Ein Datepicker für den Monat
<code>week</code>	Ein Datepicker für die Kalenderwoche
<code>time</code>	Ein Datepicker für Uhrzeit (ohne Zeitzone)
<code>email</code>	Ein Texteingabefeld für E-Mailadressen
<code>file</code>	Ein Fileuploadfeld
<code>hidden</code>	Ein versteckter Input
<code>image</code>	Eine Grafik, die als Submitbutton dient
<code>number</code>	Ein Feld zur Eingabe von Zahlenwerten, als Numberspinner
<code>password</code>	Ein Passwortfeld mit Maskierung der Eingabe
<code>radio</code>	Ein Radiobutton
<code>range</code>	Ein Feld zur Eingabe von Zahlenwerten, als Schieberegler
<code>reset</code>	Ein Resetbutton; setzt alle Felder auf den Defaultwert zurück
<code>search</code>	Ein Texteingabefeld, das eine Suchanfrage aufnimmt

<code>submit</code>	Ein Submitbutton; leitet den Datenversand ein
<code>tel</code>	Ein Texteingabefeld, das eine Telefonnummer aufnimmt
<code>text</code>	Defaulttyp. Ein Texteingabefeld mit Defaultbreite von 20 Zeichen.
<code>url</code>	Ein Texteingabefeld, das einen URL aufnehmen soll

4.2.4. Modernizr und Inputtypen

- *CSS-Klassen:* Form-Inputs werden nicht über CSS-Klassen widergespiegelt
- *JavaScript-Property:* `Modernizr.inputtypes[type]`

4.3. Inputtyp: text

Defaulttyp. Erzeugt ein einfaches Texteingabefeld, dessen Breite (in Zeichen) durch das `size`-Attribut und dessen maximal akzeptierte Eingabestringlänge durch das `maxlength`-Attribut bestimmt werden. Alle Texteingabefelder eines Formulars müssen verschiedene `name`-Attributwerte besitzen.

Beispiel:

```
E-Mail: <input type="text" name="email"><br>
Pin-Nr: <input type="text" name="pin">
```

4.4. Inputtyp: button

Erzeugt einen neutralen Button ohne festgelegte Wirkung, dessen Beschriftung durch sein `value`-Attribut bestimmt wird. Dieser Buttontyp wird gewöhnlich eingesetzt, um Scripte zu steuern.

Beispiel:

```
<input type="button"
      value="Klick mich" onclick="tuwas()">
```

4.5. Inputtyp: checkbox

Erzeugt ein Boolesches Eingabefeld in Form einer Checkbox. Mehrere Checkboxes mit gleichem `name`-Attributwert (*control name*) bilden ein Array, in dem beliebig viele Checkboxes (keine bis alle) ausgewählt werden können. Die Werte aller gewählten Eingabefelder werden einem Property zugeordnet und gemeinsam übertragen. Eine vorzuwählende Checkbox wird mit dem Attribut `checked` markiert.

Beispiel:

```
<input type="checkbox"
      name="blume" value="rose">Rose<br>
<input type="checkbox"
      name="blume" value="tulpe">Tulpe
```

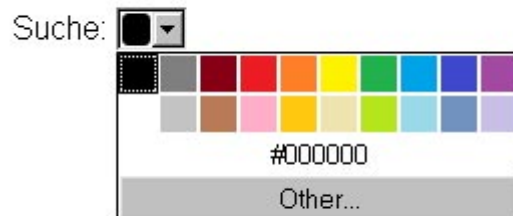
4.6. Inputtyp: color

Mit Hilfe des Typs `color` kann ein Farbwahlfeld in Form eines Colorpickers definiert werden. Die Umsetzung hängt vom Browser ab (derzeit wird der Typ nur in Opera 11 unterstützt). Rückgabewert ist eine Farbangabe in Form eines sechsstelligen Hexadezimalwerts.

Beispiel:

Farbe: `<input type="color" name="user_color" />`

Beispiel: Inputtyp color



4.7. Inputtyp: date, month, week, time, datetime

IN HTML5 wurde eine ganze Reihe spezialisierter Inputtypen zur Eingabe von Datumswerten eingeführt. Dies sind wie folgt:


- **date**

Eingabe von Tag, Monat und Jahr (Datepicker) month - Selects month and year

Datum:

Beispiel: Inputtyp date

Datum: Ausgabe:



Mon	Tue	Wed	Thu	Fri	Sat	Sun
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Today

- **week**

Eingabe von Monat und Jahr

Datum:

Beispiel: Inputtyp week

Datum: Ausgabe:



Week	Mon	Tue	Wed	Thu	Fri	Sat	Sun
35	29	30	31	1	2	3	4
36	5	6	7	8	9	10	11
37	12	13	14	15	16	17	18
38	19	20	21	22	23	24	25
39	26	27	28	29	30	1	2
40	3	4	5	6	7	8	9

Today

- **time**

Eingabe eines Zeitwerts (Stunden und Minuten)

Zeit:

Beispiel: Inputtyp time

Zeit: Ausgabe:

- **datetime**

Eingabe eines Zeitstempelwertes (Zeit, Tag, Monat Jahr im UTC-Format)

Zeitstempel: `<input type="datetime" name="zeitstempel">`

Beispiel: Inputtyp datetime

Zeitstempel: 2011-05-13 11:49 UTC Ausgabe: 2011-05-13T11:49Z

Mon	Tue	Wed	Thu	Fri	Sat	Sun
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Today

- **datetime-local**

Eingabe eines Zeitstempelwertes (Zeit, Tag, Monat Jahr, lokale Zeitzone)

Zeitstempel (lokalisiert):

`<input type="datetime-local" name="zeitstempel">`

Beispiel: Inputtyp datetime-local

Zeitstempel (lokalisiert): 2011-05-05 11:45 Ausgabe: 2011-05-05T11:45

Mon	Tue	Wed	Thu	Fri	Sat	Sun
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Today

4.8. Inputtyp: email

Der neue Typ "email" dient, wie der Name verrät, zur Eingabe von E-Mailadressen. Hierbei soll der Browser beim Datenversand automatisch eine Gültigkeitsvalidierung vornehmen. Von älteren Browsern wird der Input als Texteingabefeld wiedergegeben.

Beispiel:

```
<label for="email">E-Mail:</label>
<input type="email" name="email" id="email">
```

4.9. Inputtyp: file

Erzeugt ein Dateiupload-Feld, mit dem der Nutzer eine Datei durch eine Browse-Funktion auf seinem eigenen Rechner auswählen und beim Verschicken der Formulardaten mitversenden kann.

- ✓ **Achtung:** Für das `enctype`-Attribut des `<form>`-Containers muss `enctype="multipart/form-data"` gesetzt werden, damit ein Dateiupload ermöglicht wird.

Beispiel:

Upload Foto: `<input type="file" name="user_foto">`

4.10. Inputtyp: hidden

Erzeugt ein so genanntes „verstecktes“ Feld, dessen Wert vom Nutzer nicht unmittelbar geändert werden kann. Ein verstecktes Feld kann mittels seines `value`-Attributs mit einem Wert vorbelegt werden.

- ✓ Ansonsten können dynamisch erzeugte (z.B. berechnete) Werte per Script dort abgelegt werden. Dies ist beispielsweise bei kaskadierten Formularen von Nutzen, die im Vorfeld eingegebene Werte weiterreichen müssen.

Beispiel:

```
<input type="hidden" name="land" value="Deutschland">
```

4.11. Inputtyp: image

Bindet über das `src`-Attribut eine Grafik ein, die als Schaltfläche für den Submitvorgang dient. Aus Accessibilitygründen sollte ein Grafikbutton stets zusätzlich ein `alt`-Attribut mit passendem Wert erhalten.

Beispiel:

```
<input type="image"
      src="img_submit.gif" alt="Abschicken">
```

4.12. Inputtyp: number

Der Input vom Typ number dient zur Eingabe von Zahlen. Er wird vom Browser als Texteingabefeld mit am rechten Rand liegenden Inkrement- und Dekrementbuttons (*Numberspinner*) umgesetzt.

Beispiel:

Punkte: `<input type="number"
 name="punkte" min="1" max="10">`

Beispiel: Inputtyp number

Number: Ausgabe:

4.12.1. Zusatzattribute für Inputtyp number

Mit den folgenden Attributen kann der Eingabebereich festgelegt werden:

Attribut	Werttyp	Beschreibung
max	<i>number</i>	Legt den höchsten erlaubten Eingabewert fest
min	<i>number</i>	Legt den niedrigsten erlaubten Eingabewert fest
step	<i>number</i>	Legt das gestattete Eingabeintervall fest (für step="4" sind beispielsweise die Eingaben -4, 0, 8, 12 usw. erlaubt)
value	<i>number</i>	Legt den Defaultwert fest

4.13. Inputtyp: password

Erzeugt ein Passworteingabefeld, bei dem eingegebene Zeichen in der Darstellung durch Sterne (Asterisken) als Platzhalter ersetzt werden. Die interne Zwischenspeicherung (*current value* des Objekts) und Übertragung des eingegebenen Wertes erfolgt jedoch in Klartext. Das Passwortfeld gleicht ansonsten dem Eingabefeld vom Typ "text".

Beispiel:

```
<input type="password" name="pwd">
```

4.14. Inputtyp: radio

Erzeugt ein Boolesches Eingabefeld in Form eines Radiobuttons. Mehrere Radiobuttons mit gleichem `name`-Attributwert (*control name*) bilden ein Array, in dem genau ein Radiobutton ausgewählt werden kann. Ein vorzuzählender Radiobutton wird mit dem Attribut `checked` markiert.

- ✓ Nur der Wert des **gewählten Eingabefelds** wird übertragen (für eine sinnvolle Funktion müssen die Radiobuttons daher unterschiedliche *values* besitzen).

Beispiel:

```
<input type="radio" name="blume" value="rose">Rose<br>
<input type="radio" name="blume" value="tulpe">Tulpe
```

4.15. Inputtyp: range


Der Input vom Typ `range` dient zur Eingabe von numerischen Werten aus einem festzulegendem Bereich. Hierfür wird ein Start- und ein Endwert der möglichen Eingabe festgelegt. Der Input wird im Browser als Schieberegler (Slider) umgesetzt.

- ✓ Die Wertänderung erfolgt stufenlos (d.h. in Einerschritten), sofern nicht ein `step`-Attribut gesetzt ist, das eine Schrittweite definiert.

Beispiel:

```
<input type="range" name="punkte" min="1" max="10">
```

Beispiel: Inputtyp range

Range:  Ausgabe:

4.15.1. Zusatzattribute für Inputtyp range

Mit den folgenden Attributen kann der Eingabebereich festgelegt werden:

Attribut	Wert	Beschreibung
max	<i>number</i>	Legt den höchsten erlaubten Eingabewert fest
min	<i>number</i>	Legt den niedrigsten erlaubten Eingabewert fest
step	<i>number</i>	Legt das gestattete Eingabeintervall fest (für step="4" sind beispielsweise die Eingaben -4, 0, 8, 12 usw. erlaubt)
value	<i>number</i>	Legt den Defaultwert fest

4.16. Inputtyp: reset

Erzeugt einen Reset-Button, der vorgenommene Eingaben des Nutzers löscht und die Eingabefelder auf die Vorgabewerte zurücksetzt.

Die Aufschrift des Reset-Buttons lautet defaultmäßig (in Abhängigkeit von der Spracheinstellung) „**Reset**“, kann aber über das value-Attribut des Elements beliebig festgelegt werden.

Beispiel:

```
<input type="reset" value="Zurücksetzen">
```

4.17. Inputtyp: search

Der Inputtyp "search" wird für Suchanfragen eingesetzt. In diesem Zusammenhang macht der Einsatz des placeholder-Attributs Sinn, dass einen Hinweistext ins Feld schreibt, der beim Klick in das Feld verschwindet und, sofern keine Eingabe vorgenommen wurde, "onblur" wieder erscheint.

Ansonsten verhält sich das Feld wie ein reguläres Texteingabefeld.

Beispiel:

```
<input type="search"
      placeholder="Suchanfrage" name="search">
```

Beispiel: Inputtyp search

Suche:

Hinweis: Auf OS X-Systemen kann ein solches Suchfeld gemäß der Mac-Konvention, d.h. mit runden Ecken, dargestellt werden.

4.18. Inputtyp: submit

Erzeugt einen Submit-Button, der auf Klick einen Submit-Event erzeugt und den Dateninhalt des Formulars an die Adresse schickt, die das `action`-Attribut des umgebenden `<form>`-Containers bestimmt.

Die Aufschrift des Submit-Buttons lautet defaultmäßig (in Abhängigkeit von der Spracheinstellung) „Submit“, kann aber über das `value`-Attribut des Elements beliebig festgelegt werden.

Beispiel:

```
<form action="form_action.asp" method="get">  
Email: <input type="text" name="email"><br>  
      <input type="submit" value="Abschicken">  
</form>
```

4.19. Inputtyp: tel

Dieser Inputtyp wird für Felder eingesetzt, die eine Telefonnummer als Eingabe erhalten sollen. Optisch gleicht es einem gewöhnlichen Texteingabefeld.

Beispiel:

```
Mobile: <input type="tel" name="user_mobile" />
```

4.20. Inputtyp: url

Der Inputtyp `url` wird eingesetzt, wo der Nutzer eine Webadresse eingeben soll. Die Validierung erfolgt entsprechend den Regeln, denen Webadressen unterliegen. Ältere Browser geben den Typ als Texteingabefeld wieder.

Beispiel:

Homepage: `<input type="url" name="user_url" />`

4.21. Validierung von Formularfeldern

In HTML5 besteht das Standardverhalten in einer Validierung, sofern ein Feldtyp bestimmten Regeln unterliegt. Diese können sich aus dem Typ selbst (email, url, number) ergeben, oder über ein `pattern`-Attribut festgelegt werden.

```
<input pattern="[0-9][A-Z]{3}"
       name="bauteil"
       title="Eine Bauteilnummer besteht aus
             einer Ziffer gefolgt von drei
             Großbuchstaben."
/>
```

Beispiel: Inputtyp text (mit pattern)

Bauteil:

Please use the required format
Eine Bauteilnummer besteht aus einer
Ziffer gefolgt von drei Großbuchstaben.

Soll ein Feld eine Eingabe erhalten, so kann dies auch per `required`-Attribut bestimmt werden. In diesem Falle wird bei der Validierung geprüft, ob das Feld *leer* ist.

```
<input type="email"
       name="email"
       placeholder="Ihre E-Mail hier"
       required
       title="Bitte gültige E-Mail Adresse eingeben!">
```

Hier wird ein E-Mailfeld auf erfolgte Eingabe geprüft (Opera 11):

Beispiel: Inputtyp email

E-Mail:

This is a required field

... und dasselbe Feld nach erfolgter Eingabe bei Typverstoß:

Beispiel: Inputtyp email

E-Mail:

Please enter a valid email address

✓ Achtung – eine Validierung erfolgt nur, wenn der Input ein `name`-Attribut besitzt!

Bei einem Feld vom Typ *number* wird auch der Wert der `min`- und `max`-Attribute in Betracht gezogen:

```
<input type="number" name="points" min="1" max="10">
```

Beispiel: Inputtyp number

Number: Ausgabe: 30

Please enter a number less than or equal to 10

4.21.1. Unterdrücken der Validierung

Damit ein Feld oder das gesamte Formular nicht validiert wird, muss ausdrücklich das `novalidate`-Attribut gesetzt werden:

```
<form novalidate>
  <input type="email" id="addr" name="Email">
  <input type="submit" value="Abschicken">
</form>
```

5. Medieneinbindung – Audio und Video

In HTML5 werden einzubindende Videos und Audiofiles übergreifend als „Medien“ bezeichnet. Zur konkreten Einbindung existieren zwei spezialisierte Container-Elemente `<audio>` und `<video>`. Beide gleichen sich von der Handhabung und dem API zum Abspielen, wobei `<video>` allerdings über einen Darstellungsbereich mit Breite und Höhe verfügt, `<audio>` hingegen nicht.

- ✓ Beide Container können Inhalte besitzen, die entweder mittels des `<source>`-Tags eine Quelle nennen, oder Fallback-Inhalte bestimmen, falls der betreffende Mediencontainer nicht unterstützt wird.

5.1. Audio einbinden mit `<audio>`

Ein Audiofile kann sehr einfach eingebunden und automatisch sofort nach dem Laden abgespielt werden. Dieses hier ist dazu auf Endloswiedergabe (Loop) geschaltet:

```
<audio src="elvis.mp3" autoplay loop></audio>
```

- ✓ Achtung – ein Audio Element kann im FireFox nur abgespielt werden, wenn es im DOM eingehängt ist.

5.1.1. Boolesche Attribute von AUDIO

Allen **booleschen Attributen** kann auch der Attributname zusätzlich als Wert übergeben werden, um die Option zu aktivieren (z. B. `autoplay="autoplay"`. Dies ist dann XML-konform.).

Im Prinzip hätte jedoch jeder String, wie auch das bloße Setzen des Attributs ohne Wert, dieselbe Wirkung (`autoplay="true"` oder, wie oben, einfach `autoplay`).

Achtung – `autoplay="false"` *aktiviert* die Option daher ebenfalls! Um eine Option inaktiv zu machen, lässt man das Attribut daher weg.

- ✓ **preload:** *none | metadata | auto*
Soll gepuffert (vorausgeladen) werden?
- ✓ **autoplay** *bln*
Soll der Inhalt sofort abgespielt werden, sobald der Buffer voll ist?
- ✓ **loop** *bln*
Soll der Inhalt in einem Loop (mehrfach oder unendlich oft) abgespielt werden?
- ✓ **controls** *bln*
Soll der Browser Steuerungselemente wie Play, Pause, Position und Lautstärke anzeigen?

5.1.2. Funktionen der JavaScript-API

Im Rahmen der JavaScript-API stehen für Audio-Elemente mehrere Methoden und Eigenschaften zur Verfügung, mit denen das Element gesteuert werden kann.

- ✓ **play()**
Spielt den Inhalt der zugewiesenen Src des Audio-Elements ab.
- ✓ **pause()**
Pausiert das Abspielen der Datei
- ✓ **load()**
Lädt die Audiodatei neu (ohne sie abzuspielen)
- ✓ **canPlayType(type)**
Prüft, ob das übergebene Format abgespielt werden kann (Antwort: „*probably*“ (Unterstützung wahrscheinlich), „*maybe*“ (Unterstützung möglich) oder der leere String "" (Format nicht unterstützt).

Formate: audio/mpeg, audio/ogg, audio/mp4

**volume**

Dezimalzahl zwischen 0 und 1, mit der die Lautstärke gesteuert oder ausgelesen werden kann

**duration**

Die Abspieldauer der Audio-Datei.

**currentTime**

Die aktuelle Position in der spielenden (oder pausierten) Audio-Datei

5.1.3. Einbinden von Audiodateien mit SOURCE-Element

Anstelle des `src`-Attributs können auch `<source>`-Elemente im Containerinneren eingesetzt werden. Dies ist sinnvoll, sobald *mehrere alternative Formate* angeboten werden müssen, die jeweils von verschiedenen Browsern benötigt werden:

```
<audio controls="controls">
  <source src="elvis.mp3">
  <source src="elvis.ogg">
  <!-- weitere Inhalte als Fallback -->
</audio>
```



Achtung: Sobald Source-Elemente eingesetzt werden, sollte das `src`-Attribut in `<audio>` selbst entfallen.

Das dortige `src`-Attribut besitzt laut Spezifikation „Vorrang“ vor eventuellen inneren Source-Elementen und es wäre demzufolge nicht sichergestellt, dass diese *überhaupt* berücksichtigt würden!

Browser	Ogg Vorbis	MP3	WAV
FireFox 3.6+	✓		✓
Safari 5+		✓	✓
Chrome 6	✓	✓	
Opera 10.5+	✓		✓
Internet Explorer 9		✓	✓

Anmerkung: Firefox unterstützt MP3 aus Lizenzgründen nicht!

5.1.4. Modernizr und <audio>

- *CSS-Klassen:* .audio / .no-audio
- *JavaScript-Property:* Modernizr.audio
- Aktuell prüft Modernizr die **Audio-Formate** ogg, mp3, wav und m4a.

Einsatzbeispiel:

```
// Element erstellen:
var audio = new Audio();

// ins DOM einfügen, damit es abgespielt werden kann
document.body.appendChild(audio);

// prüfen, welcher Audiotyp unterstützt wird
audio.src = Modernizr.audio.ogg ? 'background.ogg' :
            Modernizr.audio.mp3 ? 'background.mp3' :
            'background.m4a';

// abspielen:
audio.play();
```

5.2. Video einbinden mit <video>

Dieser Befehl bindet ein Video im Format `.mov` ein:

```
<video src="test.mov"
      controls="controls"
      autoplay="autoplay"
      height="480"
      width="640"></video>
```

Im Prinzip genügt das. Leider sind die Dinge in der Regel nicht so einfach.

- ✓ Bislang (und dies mag auf absehbare Zeit so bleiben) besteht **keine Einigkeit über die Codecs**, die ein eingebettetes Video haben soll und darüber, welche Codecs vom Useragent unterstützt werden sollen.

Aus diesem Grunde ist "mehrgleisiges" Vorgehen anzuraten – das Video *muss* in **mehreren Formaten** angeboten werden.

- **WebM-Format** (Video-Codec *VP8* und Audio-Codec *Vorbis*)
WebM siehe: www.webmproject.org/
- **MP4-Container** (Video-Codec *H.264 baseline* und Audio-Codec *AAC*)
H.264 siehe: www.mpegif.org/
- **Ogg-Container** (Video-Codec *Theora* und Audio-Codec *Vorbis*)
Ogg siehe: www.xiph.org/ogg/

Alle drei alternativen Inhalte werden innerhalb eines `<video>`-Elements über je ein `<source>`-Element eingebettet. Alternativ kann noch ein Fallback im Flash-Format angeboten werden, um sicher zu gehen.

In diesem Fall wird das einzubindende Objekt mittels eines oder mehrerer `<source>`-Elemente im Inneren des Mediencontainers genannt. Das `src`-Attribut entfällt. Der Browser wählt das erstbeste unterstützte Format und ignoriert die weiteren:

Bindet ein Video mit zwei alternativen Formaten ein:

```
<video width="640" height="480" autoplay>
  <source src="test.mp4" type="video/mp4">
  <source src="test.ogv" type="video/ogg">
```

```
<!-- weitere Inhalte als Fallback -->
</video>
```

- ✓ Der Webserver muss entsprechend eingerichtet sein, um die Videodaten ausliefern zu können. Achten Sie darauf, dass in der Ini-Datei von Apache oder einem `.htaccess`-File die notwendigen Einstellungen vorgenommen sind (hier beispielhaft für das Ogg-Format):

```
AddType video/ogg .ogv
AddType audio/ogg .oga
```

5.2.1. Modernizr und `<video>`

- *CSS-Klassen:* `.video / .no-video`
- *JavaScript-Property:* `Modernizr.video`
- ✓ Aktuell prüft Modernizr die Formate ogg, webm und h264.

5.3. Attribute für <audio> und <video>

Beide Medienelemente <audio> und <video> betten eine abspielbare Datei ein, die geladen und gesteuert werden muss. Die Anforderungen weisen daher einige Parallelen auf. Aus diesem Grund greifen beide Elemente auf einen gemeinsamen Attributstamm zurück.

5.3.1. Gemeinsame Attribute <audio> und <video>

Für <audio> und <video> sind gemeinsamfolgende Attribute definiert:

Attribut	Wert	Beschreibung
src	URL	Optional. Ein URL, der auf die einzubindende Mediendatei verweist. Meist ist es günstiger, stattdessen mehrere <source>-Elemente im Inneren des Medienelements zu verwenden, da dies alternative Inhalte ermöglicht, falls ein Format nicht unterstützt wird.
autoplay	autoplay	Optional. Ein boolesches Attribut. Ist es gesetzt, wird das eingebettete Medium sofort nach dem Laden automatisch abgespielt.
loop	loop	Optional. Ein boolesches Attribut. Ist es gesetzt, wird das Abspielen solange wiederholt, bis dies abgebrochen wird – im Zweifelsfall endlos.
controls	controls	Optional. Ein boolesches Attribut, das entscheidet, ob der Browser die Defaultkonfiguration der Bedienelemente anzeigen soll.
preload	none metadata auto	Optional. Das preload-Attribut legt fest, ob der Browser die Datei vorausladen oder zumindest deren Metadaten in Erfahrung bringen soll.

Tabelle: Gemeinsame Attribute für <audio> und <video>

Hier ein Beispiel für ein eingebettetes Video, dessen **Metainformationen** downgeloadet werden, das aber nicht unverzüglich abgespielt wird:

```
<video src="test.ogg"
      width="320" height="240"
      preload="metadata"></video>
```

- ✓ Zu den **Metainformationen** gehören die Abmessungen, Dauer, Tonspurinformationen (Tracklistings) und eventuell die ersten Bildframes des Videos.

Hier wird der Browser angewiesen, keinen spontanen Download der Resource einzuleiten und auch keine Metadaten zu holen:

```
<video src=" test.ogg"
      width="320" height="240" preload="none"></video>
```

- ✓ Ein Laden dieses Videos würde erst beginnen, sobald der User das Abspielen veranlasst.

Mit `preload="auto"` veranlasst man den Browser, die Resource „gegebenenfalls“ vorzuladen (ohne dass, dies ist einschränkend anzumerken, dieses Verhalten strikt anweisen zu können), sie aber nicht sofort abzuspielen. Dies geht über das laden der Metadaten hinaus.

- ✓ Wird `preload` mit leerem String als Wert (oder ohne Wert) eingesetzt, so entspricht dies dem `auto`-Keyword.

Natürlich kann auch ein Download **und** automatisches Abspielen angeordnet werden (ob der User in diesem Moment *zuseht*, kann natürlich nicht geprüft werden):

```
<video src=" test.ogg "
      width="320" height="240" autoplay></video>
```

5.3.2. Spezielle Attribute für <video>

Für <video> gelten zusätzlich weitere Attribute:

Attribut	Wert	Beschreibung
audio	muted	Optional. Schaltet die Tonspur auf stumm. Derzeit ist "muted" der einzige, erlaubte Wert.
poster	URL	Optional. Nennt den URL eines Standbildes, das

		gezeigt wird, bis der erste Frame des eingebetteten Videos verfügbar ist. Auf I-Phone und I-Pad wird das Poster gezeigt, bis der User das Video startet (diese Geräte unterdrücken den ersten Frame).
width height	Pixelangabe, %	Legt die Abmessungen der Abspielfläche des Videos fest. Akzeptiert werden Ganzzahlen, die als Pixelwerte interpretiert werden und Prozentangaben, die sich auf die Breite des umgebenden Containers beziehen.

Tabelle: Elementspezifische Attribute für <video>

5.4. Fallback Verhalten

Unterstützt der Browser das `<audio>` oder `<video>`-Element nicht, so kann ihm im Inneren des Containers ein **Fallbackinhalt** zur Verfügung gestellt werden.

Dies funktioniert, weil solche Browser den Tag selbst (und eventuelle `<source>`-Elemente) ignorieren, den *restlichen Inhalt* aber sehr wohl auswerten können:

```
<video src="test.mov" controls="controls">
  ...
  <!-- Fallback kann beliebiger Inhalt sein; -->
  <p>Browser unterstützt Video-Element nicht!</p>
  <!-- Ende des Fallbacks -->
</video>
```

5.4.1. Fallback auf Quicktime

Für die meisten älteren Browser funktioniert der Fallback auf **Quicktime**-Inhalte. Hierfür muss lediglich das von Apple erhältliche JavaScript `AC_QuickTime.js` eingebunden werden:

```
<script src="AC_QuickTime.js"
  type="text/javascript"></script>
```

Quelle: <http://developer.apple.com/internet/licensejs.html>

Es wird eine Funktion aus dem geladenen Script aufgerufen, die einen passenden Object-Tag erstellt und in die Seite ausgibt:

```
<video controls="controls">
  <source src="test.mov" type="video/quicktime">
  <source src="test.3gp" type="video/3gpp">

  <!-- Fallback: -->
  <script type="text/javascript">

    // Funktion aus AC_QuickTime.js:
    QT_WriteOBJECT('test' , '320', '240', '');

  </script>

</video>
```

5.4.2. Fallback auf andere Plugins

Ein anderes, übliches Verfahren besteht im Einfügen eines Object-Tags ins Innere des Medienelements:

```
<video controls="controls">
  <source src="test.m3u8" type="vnd.apple.mpegURL">
  <source src="test.mp4" type="video/mp4">
  <source src="test.ogv" type="video/ogg">

  <!-- Fallback: -->

  <object
    classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
    codebase="http://www.apple.com/qtactivex/qtplugin.cab"
    height="320" width="240">
    <param name="src" value="test.mp4">
  </object>

</video>
```

Anmerkung: Der Wert des `classid`- und des `codebase`-Attributs *muss* auf den einzubindenden Medientyp zugeschnitten sein. Dasselbe gilt für die Parameter, die im Inneren des Objekts übergeben werden.

- ✓ Die Einbindung mit `<object>` ist daher nicht ganz trivial (und illustriert nochmals, warum das `<video>`-Element eigentlich eingeführt wurde).

5.5. API für Video und Audio

Der HTML5-Parser stellt für die Elemente `<video>` und `<audio>` eine DOM-Schnittstelle zur Verfügung, mit deren Hilfe die Elemente durch JavaScript gesteuert werden können.

5.5.1. Methoden

Über die API kann beispielsweise das Abspielen eines Mediums eingeleitet bzw. gestoppt werden. Hierzu dienen die API-Methoden `play()` und `pause()`.

Das Video-Element wird hierfür als **JavaScript-Objekt** referenziert. Am einfachsten versteht man es dazu mit einem Identifier:

```
<video src="test.ogg"
      width="320" height="240" id="v1"></video>
```

Anschließend kann es per Script adressiert werden (das Dokument muss fertig geladen sein):

```
var meinVideo = document.getElementById("v1");

function starten() {
    meinVideo.play(); // startet dasVideo
}

function stoppen() {
    meinVideo.pause(); // hält den Abspielvorgang an
}
```

Weitere Methoden:

✓ **load()**
Lädt die Videodatei neu (ohne sie abzuspielen)

✓ **canPlayType(type)**
Prüft, ob das übergebene Format abgespielt werden kann (Antwort: „probably“ (Unterstützung wahrscheinlich), „maybe“ (Unterstützung möglich) oder der leere String "" (Format nicht unterstützt).

Formate: video/ogg, video/mp4, video/webm

Mit Codec: video/ogg; codecs="theora, vorbis", video/mp4; codecs="avc1.4D401E, mp4a.40.2", video/webm; codecs="vp8.0, vorbis"

5.5.2. Events

Ein Methodenaufruf ist mit einem entsprechenden DOM-Ereignis verbunden – so bewirkt `play()` einen `play`-Event, `pause()` einen `pause`-Event. Andere Ereignisse werden von außen getriggert. Hier eine (unvollständige) Übersicht:

Ereignis	Beschreibung
loadstart	Das Laden der Ressource hat begonnen. <code>networkState</code> hat den Wert <code>NETWORK_LOADING</code>

progress	Das Laden der Ressource ist erfolgreich aufgenommen worden. networkState hat den Wert <i>NETWORK_LOADING</i>
suspend	Das Laden der Ressource ist unterbrochen worden. Die Ressource ist noch nicht komplett geladen. networkState hat den Wert <i>NETWORK_IDLE</i>
abort	Das Laden der Ressource ist abgebrochen worden. networkState hat entweder den Wert <i>NETWORK_EMPTY</i> oder <i>NETWORK_IDLE</i>
error	Es ist ein Fehler aufgetreten. networkState hat entweder den Wert <i>NETWORK_EMPTY</i> oder <i>NETWORK_IDLE</i>
stalled	Der Datenstrom beim Laden ist abgebrochen. networkState hat den Wert <i>NETWORK_LOADING</i> .
loadedmetadata	Die Metadaten der Ressource sind geladen (Abmessungen und Länge stehen fest). readyState erhält den Wert <i>HAVE_METADATA</i>
loadeddata	Die Ressource ist geladen und abspielbereit. readyState erhält den Wert <i>HAVE_CURRENT_DATA</i>
waiting	Das Abspielen der Ressource ist unterbrochen, wird aber fortgesetzt (weil z.B. auf den nächsten Frame gewartet wird). paused hat den Wert <i>false</i>
playing	Die Ressource wird abgespielt. readyState hat den Wert <i>HAVE_FUTURE_DATA</i>
ended	Das Abspielen der Ressource ist beendet, weil das Ende der Datei erreicht wurde. ended-Property erhält den Wert <i>true</i> .
play	Die play()-Methode wurde aufgerufen. paused erhält den Wert <i>false</i>
pause	Die pause()-Methode wurde aufgerufen. paused erhält den Wert <i>true</i>

Mehr Info:

www.whatwg.org/specs/web-apps/current-work/multipage/video.html

5.5.3. Eigenschaften

Mittels verschiedener DOM-Eigenschaften kann der Zustand des Videos erfasst werden (Auch die folgende Übersicht ist gekürzt):

Eigenschaft	Wert	Beschreibung
readyState	Integer	0: HAVE_NOTHING 1: HAVE_METADATA 2: HAVE_CURRENT_DATA 3: HAVE_FUTURE_DATA 4: HAVE_ENOUGH_DATA
paused	boolean	true: Video gestoppt false: Video läuft
ended	boolean	true: Video ist bis zum Ende gelaufen false: Ende noch nicht erreicht
networkState	Integer	0: NETWORK_EMPTY 1: NETWORK_IDLE 2: NETWORK_LOADING 3: NETWORK_NO_SOURCE

Mehr Info:

www.whatwg.org/specs/web-apps/current-work/multipage/video.html

6. Das Canvas-Element

Das in HTML5 offiziell gemachte `<canvas>`-Element stellt eine programmatisch erzeugte Bitmap-Grafik dar, die dazu dienen kann, ad hoc Balkengraphen, Gaming-Oberflächen oder andere Illustrationen zu erzeugen, die in die Webseite eingefügt werden sollen.

✓ Das Element selbst hat keine visuelle Präsentation.

Ein Canvas-Element wie das folgende (völlig korrekt geschrieben) wäre demnach nicht zu sehen:

```
<canvas></canvas>
```

Im Grunde kennzeichnet der Container nur einen Bereich, innerhalb dessen die Grafik gezeigt wird. Es ist sinnvoll, wie bei Grafiken allgemein üblich, diesen Bereich mit Abmessungen zu versehen:

```
<canvas width="300" height="225"></canvas>
```

Da beliebig viele Canvas-Elemente auf einer Seite enthalten sein können, sollte man sie jeweils mit einem ID versehen, um sie leichter über das DOM zugänglich zu machen:

```
<canvas id="meineCanvas"
        width="300" height="225"></canvas>
```

Das Canvas-Element kann nun über DOM-Methoden erfasst werden:

```
var dieCanvas = document.getElementById("meineCanvas");
```

Für den Fall, dass der aktuelle Browser Canvas nicht unterstützt, ist es möglich, in den Container Fallbackinhalte einzufügen. Ansonsten kann er auch leer bleiben.

```
<canvas id="meineCanvas" width="300" height="225">
    Ihr Browser unterstützt das Canvas-Element nicht!
</canvas>
```

6.1.1. Der Drawing-Context und seine Methoden

Die Befüllung eines Canvas-Bereiches erfolgt über JavaScript. Auf diesem Wege können Linien, Farbflächen, Farbverläufe definiert werden, oder Schrift und Images in die Canvas eingebettet werden.

Diese Funktion zeichnet ein Rechteck:

```
function zeichne() {

var dieCanvas = document.getElementById("meineCanvas");

var canvasContext = dieCanvas.getContext("2d");

canvasContext.fillRect(40, 20, 140, 100);

}
```

Zunächst wird hier eine DOM-Referenz auf das Canvas-Objekt gebildet und deren `getContext()`-Methode ausgeführt. Dieser wird der String `"2d"` übergeben (es existiert auch eine 3D-API, die jedoch noch nicht weit unterstützt ist). Auf den Rückgabewert der Methode (bezeichnen wir ihn als Zeichenkontext) kann nun die Methode `fillRect()` angewandt werden,

die ein Rechteck auf die Fläche zeichnet. Die vier Koordinatenwerte bezeichnen die obere linke (die ersten beiden) und die untere rechte Ecke (der dritte und vierte Wert) des zu zeichnenden Rechtecks.

✓ **Rechteck definieren:**

Die Methode `fillRect(a1, a2, b1, b2)` definiert und füllt ein Rechteck mit den Koordinatenpärchen `a1/a2` und `b1/b2`.

Sie bemerken, dass über das Aussehen (den „Fill-Style“) des Rechtecks keine Aussage gemacht wird. Es wird automatisch der „*Current Fill-Style*“ angewandt, der das Rechteck mit der Farbe Schwarz füllt. Natürlich kann auch eine andere Farbe angegeben werden.

✓ **Fläche füllen:**

Das `fillStyle`-Property kann eine CSS-Farbangabe, ein Füllpattern oder ein Farbverlauf sein (dazu später). Per Default hat es den Wert `#000`. Das `fillStyle`-Property gilt für einen Zeichenkontext jeweils so lange, bis es geändert wird.

✓ **Linie füllen:**

Canvas unterscheidet zwischen Flächen und Linienfüllungen. Für letztere existiert eine eigene Eigenschaft namens `strokeStyle`. Auch ein `strokeStyle` kann eine Farbe, ein Pattern oder ein Verlauf sein; er wird jedoch auf Umrisse angewendet und nicht zum Füllen von Bereichen.

Nicht nur das Erzeugen eines Rechtecks ist möglich, sondern auch das Gegenteil. Soll ein rechteckiger Bereich gelöscht (also geleert) werden, so kann dies mittels der Methode `clearRect()` geschehen.

✓ **Rechteckfläche leeren:**

Die Methode `clearRect(a1, a2, b1, b2)` löscht die Pixel innerhalb des Bereiches, der durch das Koordinatenpärchen `a1/a2` und `b1/b2` beschrieben wird.

Beispiel – halbtransparentes Rechteck

Rechteck definieren und füllen:

```
canvasContext.beginPath();  
canvasContext.fillStyle = "rgba(255, 255, 0, .5)";  
canvasContext.fillRect(15, 150, 120, 120);
```

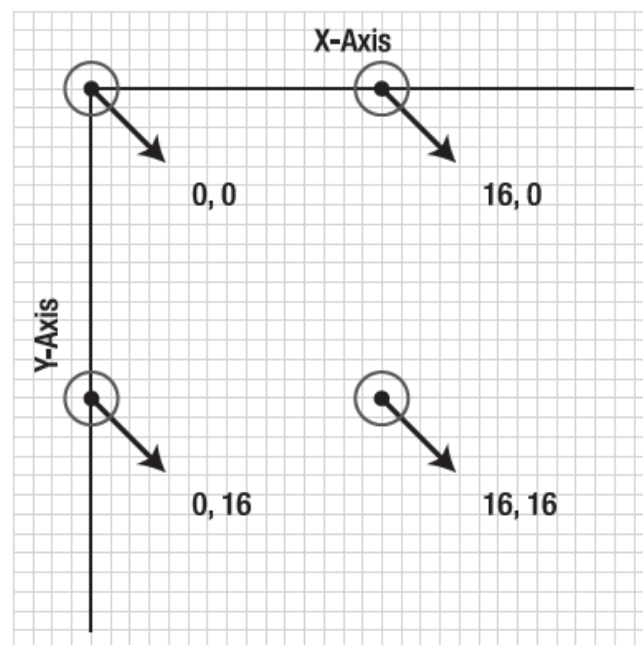
Beispiel – Kreis zeichnen

Kreis zeichnen und füllen:

```
canvasContext.beginPath();  
canvasContext.arc(75, 75, 50, 0, Math.PI*2, true);  
canvasContext.closePath();  
canvasContext.fillStyle = "#00A308"; // grün  
canvasContext.fill();
```

6.1.2. Koordinaten und Zeichenpfade

Über den Bereich einer Canvas kann man sich ein zweidimensionales Gitternetz gelegt denken. Die obere linke Ecke ist durch die Koordinate $(0, 0)$ beschrieben, die x-Achse zeigt nach rechts, die y-Achse nach unten.



Zum Zeichnen einer Linie stehen die folgenden beiden Methoden zur Verfügung:

Startpunkt der Linie setzen:

```
moveTo(x,y)
```

Endpunkt der Linie setzen (vom Startpunkt, bzw. ab letztem Endpunkt):

```
lineTo(x, y)
```

Betrachten wir ein Beispiel:

```
<canvas id="meineCanvas"
        width="500" height="500"></canvas>
```

```
var dieCanvas = document.getElementById("meineCanvas");
var canvasContext = dieCanvas.getContext("2d");
```

Auf die Fläche soll ein hellgraues Gitter (Farbe #eee) gelegt werden.

Hier die horizontalen Linien:

```
for (var x = 0.5; x < 500; x += 10) {
    canvasContext.moveTo(x, 0);
    canvasContext.lineTo(x, 500);
}
```

... und jetzt die vertikalen Linien :

```
for (var y = 0.5; y < 500; y += 10) {
    canvasContext.moveTo(0, y);
    canvasContext.lineTo(500, y);
}
```

✓ Zu beachten ist, dass wir während der ganzen Zeit im **selben Zeichenkontext** bleiben. Für die Canvas geschieht alles "im selben Zusammenhang", auch wenn das Öffnen mit `moveTo()` neu angesetzt wird.

Die Linien sind jetzt „skizziert“, aber noch unsichtbar, solange sie nicht mit `stroke()` „ausgeführt“ werden. Hierbei kann ihnen auch noch ein „Stil“ (Farbe) zugewiesen werden: Dies geschieht nun für den gesamten Kontext in `strokeStyle`, danach wird mit `stroke()` abgeschlossen:

```
canvasContext.strokeStyle = "#eee";
canvasContext.stroke();
```

Nun kann im vorhandenen Kontext weitergezeichnet werden. Hierfür muss der **Beginn** eines neuen Pfads definiert werden:


```
context.beginPath();
```

... anschließend kann mit `lineTo()` und `moveTo()` weitergearbeitet werden. Um den neuen Pfad sichtbar zu machen muss wiederum `stroke()` herangezogen werden. Dies würde automatisch auf den aktuellen `strokeStyle` zurückgreifen. Soll dieser eine andere Farbe erhalten, muss er neu gesetzt werden.

6.1.3. Modernizr und <canvas>

- *CSS-Klassen:* `.canvas` / `.no-canvas`
- *JavaScript-Property:* `Modernizr.canvas`

7. Einbettung von SVG und MathML

Die Einbettung von SVG und MathML-Daten in HTML5 ist in der Spezifikation bereits vorgesehen und kann ohne viel Aufwand geschehen. Hierbei wird der Datentyp bereits am Wurzelement erkannt. Die Namensräume der beiden Sprachen sind bereits integriert.

Folgende Regeln gelten für die Einbettung von SVG und MathML:

- Dem Element `<svg>` ist automatisch der SVG-Namensraum zugeordnet. Der Namensraum darf hier also nicht zusätzlich gesetzt werden.
- Dem Element `<math>` ist automatisch der MathML-Namensraum zugeordnet. Der Namensraum darf hier also nicht zusätzlich gesetzt werden.
- Codeblöcke in SVG bzw. MathML können verschachtelt werden.
- Innerhalb von SVG- und MathML-Blöcken wird CSS gemäß der XML-Regel verarbeitet, d.h. Selektoren für Klassen arbeiten case-sensitive. Es muss gegebenenfalls nach der kanonischen Regelung, also mit CamelCase-Schreibweise gearbeitet werden (z.B. für SVG `viewBox`).

- Elementnamen dürfen keine Doppelpunkte enthalten und können daher auch nicht mit Namensraumpräfixen versehen werden.

7.1. Beispiel MathML

Hier ein Code-Abschnitt im MathML, der die Lösung für eine quadratische Gleichung darstellt (Achtung – zur Einbettung in HTML5 darf **hier kein Namensraum** gesetzt sein!):

```
<math>
  <mi>x</mi>
  <mo>=</mo>
  <mfrac>
    <mrow>
      <mo>&minus;</mo>
      <mi>b</mi>
      <mo>&PlusMinus;</mo>
      <msqrt>
        <msup>
          <mi>b</mi>
          <mn>2</mn>
        </msup>
        <mo>&minus;</mo>
        <mn>4</mn>
        <mo>&InvisibleTimes;</mo>
        <mi>a</mi>
        <mo>&InvisibleTimes;</mo>
        <mi>c</mi>
      </msqrt>
    </mrow>
    <mrow>
      <mn>2</mn>
      <mo>&InvisibleTimes;</mo>
      <mi>a</mi>
    </mrow>
  </mfrac>
</math>
```

7.2. Beispiel SVG

Analog können in Form von SVG-Daten skalierbare Illustrationen eingebettet werden, ohne dass der HTML5-Code ausdrücklich als XHTML-Form vorliegen muss.

Im SVG können daher für eine Einbettung in eine Datei, die der HTML-Syntax folgt, ebenfalls Anführungszeichen um Attributwerte weggelassen werden, soweit sie nicht ausdrücklich erforderlich sind. (Die Auslassung ist möglich, weil die Verarbeitung mit dem HTML-Tokenizer erfolgt.)

```
<svg height=86 width=90
    viewBox='5 9 90 86'
    style='float: right;'>

    <path stroke=#F53F0C
        stroke-width=10
        fill=#F5C60C
        stroke-linejoin=round
        d='M 10,90 L 90,90 L 50,14 Z' />

    <line stroke=black
        stroke-width=10
        stroke-linecap=round
        x1=50 x2=50
        y1=45 y2=75 />

</svg>
```

7.2.1. Modernizr und SVG

Unterstützung von SVG durch den Browser

- *CSS-Klassen:* .svg / .no-svg
- *JavaScript-Property:* Modernizr.svg

Unterstützung von Inline-SVG durch den Browser

- *CSS-Klassen:* .inlinesvg / .no-inlinesvg

- *JavaScript-Property*: `Modernizr.inlinesvg`

8. Geolocation API

Mittels der **Geolocation-API** kann der momentane Aufenthaltsort des Users geographisch erfasst werden. Dies ist allerdings nicht ohne dessen Zustimmung möglich. Die Positionsbestimmung erfolgt über den Einwahlknoten, ist also naturgemäß nicht vollständig präzise.

- ✓ Für das Anbieten lokaler Dienste oder, mit gewissen Abstrichen, für Mobilanwendungen, ist der Dienst gut einsetzbar.

Die API erweitert das `navigator`-Objekt des Browsers um das `geolocation`-Objekt. Zur Positionsbestimmung steht dann dessen Methode `getCurrentPosition()` zur Verfügung. Diese nimmt zwei Callbackfunktionen entgegen:

```
navigator.geolocation
    .getCurrentPosition(
        success,
        error
    );
```

Der `success`-callback bekommt ein Positionsobjekt übergeben, dessen `coords`-Property (u.a) zwei Größen `latitude` und `longitude` enthält, aus denen sich die Position bestimmen lassen. Dies kann genutzt werden, um eine *Google-Map* anzusteuern und deren `LatLng`-Property zu befüllen:

```
function success(position) {
    //...

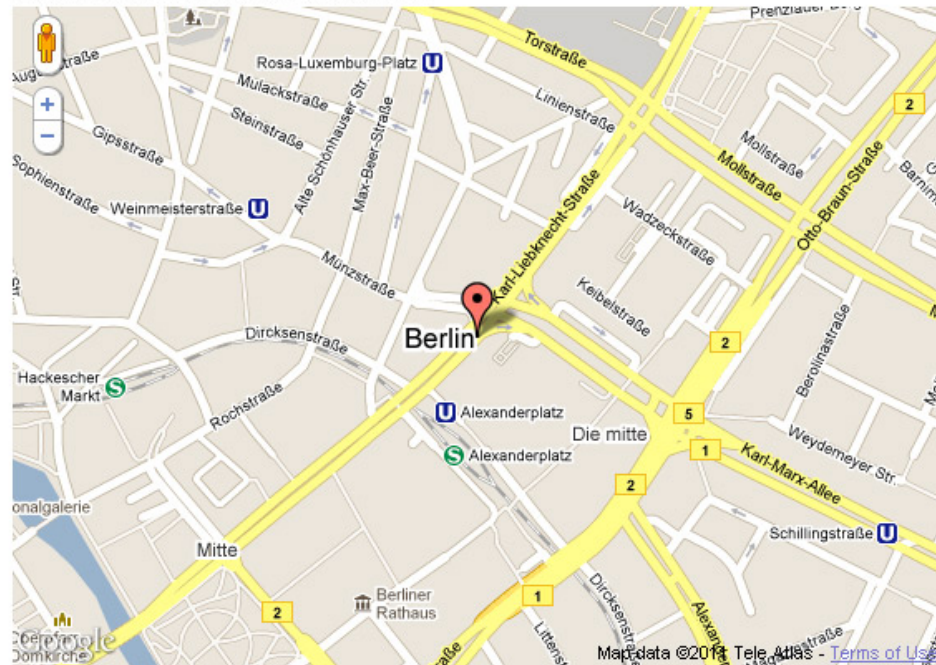
    var latlng = new google.maps.LatLng(
        position.coords.latitude,
        position.coords.longitude
    );
```

```
// ...
}
```

- ✓ Weitere Informationen zur **Google-Maps API** erhalten Sie bei Google:
<http://code.google.com/intl/de-DE/apis/maps/>

Beispiel:

Suche nach Ihrer Position: Gefunden!



Hier das ausformulierte Beispiel. In den `span#status` wird die Erfolgs- bzw. Fehlermeldung ausgegeben. Im Erfolgsfall wird ein Div-Container generiert und mit einer Google-Map gefüllt.

- ✓ Hierfür ist es erforderlich, das entsprechende **Maps-Script** von `google.com` einzubinden.

```
<!-- Google-Script: -->
<script type="text/javascript"
  src="http://maps.google.com/maps/api/js?sensor=false">
</script>

<!-- Container für Ausgabe: -->
<article>
  <p>Suche nach Ihrer Position:

```

```
<span id="status">...noch beim suchen...</span></p>
<!-- hier wird das Map-Div eingefügt -->
</article>

<script>

// Callback bei erfolgreicher Positionsbestimmung:
function success(position) {

    var s = document.querySelector('#status');

    if (s.className == 'success') {
        // Funktion bereits gelaufen:
        return;
    }

    s.innerHTML = "Gefunden!";
    s.className = 'success';

    var dieKarte = document.createElement('div');
    dieKarte.id = 'dieKarte';
    dieKarte.style.height = '400px';
    dieKarte.style.width = '560px';

    document.querySelector('article')
        .appendChild(dieKarte);

    // Konstruktor erfordert Google-Script (s.o.)!!
    var latlng = new google.maps.LatLng(
        position.coords.latitude,
        position.coords.longitude
    );

    var myOptions = {
        zoom: 15,
        center: latlng,
        mapTypeControl: false,
        navigationControlOptions: {
            style: google.maps.NavigationControlStyle.SMALL
        },
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };

    // Konstruktor erfordert Google-Script (s.o.)!!
    var karte = new google.maps.Map(
```

```
        document.getElementById("dieKarte"),
        myOptions
    );

    // Konstruktor erfordert Google-Script (s.o.)!!
    var marker = new google.maps.Marker({
        position: latlng,
        map: karte,
        title:"Sie sind hier! Ungefähr jedenfalls..."
    });
}

// Callback im Fehlerfall:
function error(msg) {
    var s = document.querySelector('#status');
    s.innerHTML = (typeof msg == 'string') ?
        msg : "... ist fehlgeschlagen";
    s.className = 'fail';
}

// Aufruf der Geolocation:
if(navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(
        success,
        error
    );
}
else {
    error('not supported');
}

</script>
```

8.1. Properties des Positionsobjekt-Objekts

Die Methode `getCurrentPosition()` des Geolocation-Objekts `navigator.geolocation` übergibt an sein Success-Callback ein Positionsobjekt, das eine Eigenschaft `coords` besitzt. In diesem sind eine Reihe von Eigenschaften definiert. Je nach Applikation können einzelne Eigenschaften mit dem *null*-Wert belegt sein, wenn sie nicht zur Verfügung

gestellt werden können. Allgemein sind `latitude` und `longitude` stets verfügbar.

```
navigator.geolocation.getCurrentPosition(  
    // Erfolg; Positionsobjekt wird übergeben:  
    function(position){  
        var lat = position.coords.latitude,  
        var long = position.coords.longitude;  
        // evtl. weitere Eigenschaften...  
    },  
    // Misserfolg  
    function() {  
        // Fehlermeldung  
    }  
));
```

- ✓ Für eine im realen Leben nützliche Anwendung sollte stets das `accuracy`-Property einbezogen werden, da Positionsangaben (je nach Methode der Ermittlung) grob ungenau sein können.

Property	Wert	Beschreibung
<code>latitude</code>	Grad (Dezimalangabe)	Geografische Breite
<code>longitude</code>	Grad (Dezimalangabe)	Geografische Länge
<code>altitude</code>	m	Höhe
<code>accuracy</code>	m	Präzision von Länge und Breite
<code>altitudeAccuracy</code>	m	Präzision der Höhenmessung
<code>heading</code>	Grad	Richtung
<code>speed</code>	m/s	Geschwindigkeit

8.1.1. Modernizr und Geolocation

CSS-Klassen: `.geolocation` / `.no-geolocation`

JavaScript-Property: `Modernizr.geolocation`

9. Drag and Drop API

An HTML5 angegliedert ist eine API, mit der sich Drag 'n Drop-Effekte erzielen lassen. Im Prinzip ist es einfach, ein Element draggable zu machen – es genügt (zumindest laut Spezifikation), ihm ein das Attribut `draggable="true"` zuzuweisen und einen Eventlistener für das `dragstart`-Ereignis zuzuweisen.

Dieser ist dafür zuständig ein Datenobjekt zu bilden, das die beim Drag erfassten Daten „verkapselt“. Der angesprochene Handler verstaut die Daten im sogenannten `DataTransfer`-Object und legt fest, auf welche Art die Daten an das Target weitergegeben werden (*copy*, *move*, *link*). Dieser Transfermodus wird als „*Drageffect*“ bezeichnet.

Das `DataTransfer`-Objekt steht dann über das **Eventobjekt** `e` dem Drophandler des Targets zur Verfügung (als `e.dataTransfer`).

Ganz ohne beteiligtes JavaScript kommt man hierbei nicht aus. Hier ein Beispiel, wie Droppables und Dropzone deklariert werden:

Ziehen Sie die Dropitems über das Droptarget:



Mit folgendem HTML:

```
<p>Ziehen Sie die Dropitems über das Droptarget:</p>
<div id="droptarget"></div>
<ul id="dragsource">
  <li><a href="#" id="Eins">Eins</a></li>
  <li><a href="#" id="Zwei">Zwei</a></li>
  <li><a href="#" id="Drei">Drei</a></li>
```

```
<li><a href="#" id="Vier">Vier</a></li>
<li><a href="#" id="Fünf">Fünf</a></li>
</ul>
```

Und folgendem JavaScript:

```
// Collection aus allen Links (Dropitems) bilden:
var links = document.querySelectorAll('li > a'),
    el = null;

// Schleife über alle Links bilden:
for (var i = 0; i < links.length; i++) {
    // den aktuellen Link nehmen
    el = links[i];
    // ... diesem Attribut draggable="true" hinzufügen:
    el.setAttribute('draggable', 'true');

    // normalisierte Eventlistener (siehe unten):
    addEvent(el, 'dragstart', function (e) {
        // effectAllowed='copy' macht Element droppable
        e.dataTransfer.effectAllowed = 'copy';
        // Daten, die beim Drop übergeben werden:
        e.dataTransfer.setData('meinId', this.id);
    });
}

// wir holen uns das Droptarget:
var droptarget = document.querySelector('#droptarget');

// Utility-Funktion aus h5utils.js:
addEvent(droptarget, 'dragover', function (e) {
    // sonst kann nicht gedroppt werden
    if (e.preventDefault) e.preventDefault();
    this.className = 'over';
    // erlaubt den Datentransfer zum Target
    e.dataTransfer.dropEffect = 'copy';
    return false;
});

// Highlighten des Targets:
addEvent(droptarget, 'dragenter', function (e) {
    this.className = 'over';
    return false;
});
```

```
// und das Highlighten entfernen:
addEvent(droptarget, 'dragleave', function () {
    this.className = '';
});

// drop-EventListener für das Target:
addEvent(droptarget, 'drop', function (e) {

    // verhindern, dass Drop-Event weitergegeben wird
    if (e.stopPropagation) e.stopPropagation();

    // hier holen wir uns den ID (siehe oben):
    var el = document.getElementById(
        e.dataTransfer.getData('meinId')
    );

    // ein Element erzeugen...
    var p = document.createElement('p');
    p.innerHTML = el.id;

    // ... und in das Target schreiben:
    droptarget.appendChild(p);

    // das verschobene Element löschen
    el.parentNode.removeChild(el);

    // Highlighten entfernen
    droptarget.className = '';

    return false;
});
```

- ✓ Die hier eingesetzte **vereinheitlichte Eventbindung** mit der Methode `addEvent()` stammt aus dem Hilfsskript **h5utils.js** von *Remy Sharp*. Dadurch werden auf einfachste Weise ältere IE vor IE9 unterstützt.

9.1. Drag & Drop-Events

dragstart:

Wird an das „Draggable“ gebunden, um das *DataTransfer*-Objekt zu definieren. Feuert zu *Beginn* des Drag-Vorgangs.

```
myDraggable.ondragstart = function(e){  
    e.dataTransfer.setData('meinText', 'meineDaten');  
    e.dataTransfer.effectAllowed = 'move';  
};
```

drag:

Feuert kontinuierlich während des Drag-Vorgangs.

dragend:

Feuert beim Ende des Drag-Vorgangs.

dragenter:

Feuert sobald ein Element in die Drop-Zone gezogen wird.

dragleave:

Cancel bei `dragenter` eingeleite Events beim Verlassen der Drop-Zone.

dragover:

Blockiert den Drop und muss daher gewöhnlich **gecanceled** werden:

```
myDropzone.ondragover = function(e){  
    e.preventDefault();  
};
```

drop:

Feuert beim Dropvorgang und beendet die Drag & Drop Operation.

9.1.1. Modernizr und Drag & Drop

- *CSS-Klassen:* `.draganddrop` / `.no-draganddrop`
- *JavaScript-Property:* `Modernizr.draganddrop`

In Modernizr 1.5 geprüfte Drag & Drop Events:

drag	dragstart	dragenter	dragover
dragleave	dragend	drop	

Randbemerkung:

Die in **h5utils.js** definierte `addEvent`-Funktion soll hier kurz gezeigt werden:

```
var addEvent = (function () {

    // Eventbindung nach Standard:
    if (document.addEventListener) {
        return function (el, type, fn) {
            if (el && el.nodeName || el === window) {
                el.addEventListener(type, fn, false);
            } else if (el && el.length) {
                for (var i = 0; i < el.length; i++) {
                    addEvent(el[i], type, fn);
                }
            }
        };
    }

    else {

        // Eventbindung im IE bis IE8:
        return function (el, type, fn) {
            if (el && el.nodeName || el === window) {
                el.attachEvent('on' + type, function () {
                    return fn.call(el, window.event);
                });
            } else if (el && el.length) {
                for (var i = 0; i < el.length; i++) {
                    addEvent(el[i], type, fn);
                }
            }
        };
    }
})();
```

Es wird ein **Funktionsobjekt** zurückgegeben, dass die Eventbindung an den jeweiligen Browser angepasst vornimmt und Bindungen nicht nur an Einzelelemente, sondern auch an Collections vornehmen kann (hierzu dient der jeweils zweite Teil mit `else if (el && el.length)`).

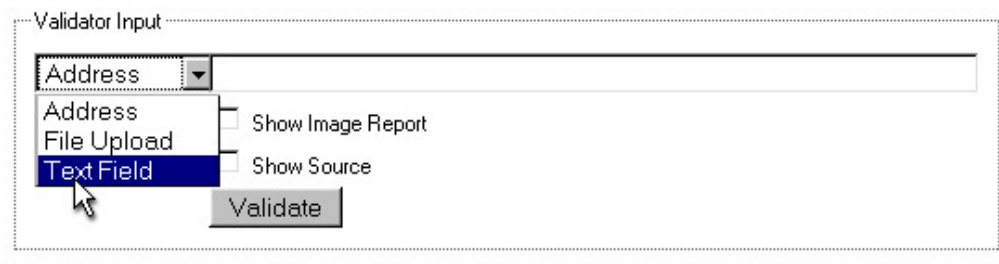
10. Validieren von HTML5-Dokumenten

HTML5-Dokumente können Sie online validieren lassen. Für diesen Zweck existiert ein speziell auf HTML5 ausgerichteter **Validator**, der sich jedoch laut Eigenaussage noch im Stadium „highly experimental“ befindet. Eine Alternative hierzu bietet das W3C an.

10.1. Validator.nu

Der in Java geschriebene **Validator.nu** akzeptiert Webadressen und Fileuploads und stellt auch ein Textfeld zur direkten Übergabe von zu validierendem Code bereit:

Validator.nu (X)HTML5 Validator (Highly Experimental)

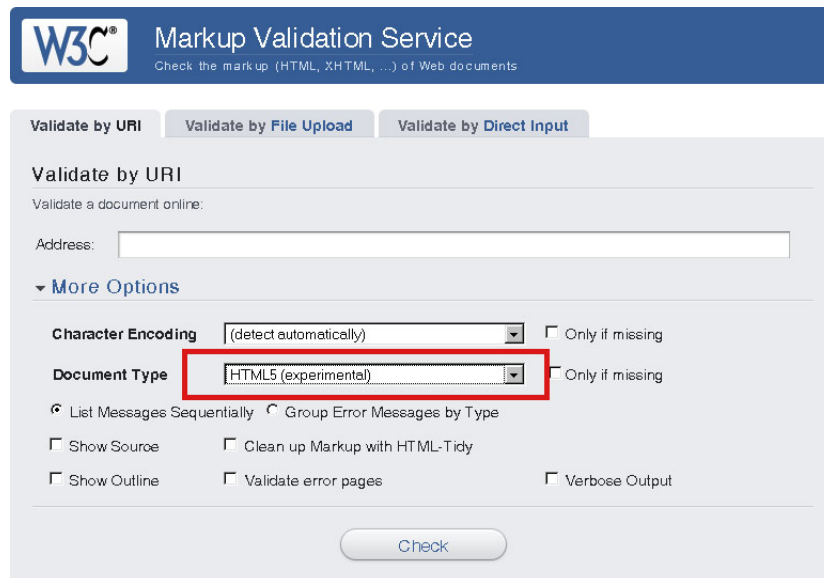


Sie finden den HTML5-Validator unter folgender Adresse:

- <http://html5.validator.nu>

10.2. W3C-Validator

Auch das W3C stellt für seinen HTML-Validator eine HTML5-Option zur Verfügung, die aber ebenfalls noch als „experimentell“ eingestuft ist. Sie können Ihr zu prüfendes Dokument entweder, unter „More options“, direkt über das Pulldownmenü als HTML5 zu erkennen geben, oder sich auf die Doctype-Declaration verlassen.



Sie finden den W3C-Validator unter folgender Adresse:

http://validator.w3.org/#validate_by_uri+with_options

11. Tools und Hilfsmittel

11.1. Zen-Coding

Das **Zen-Coding**-Plugin ermöglicht ein beschleunigtes Coding von Hand, indem es eine, an die CSS-Syntax angelehnte Kurzschreibweise expandiert.

✓ **Zen-Coding steht als Plugin zur Verfügung für:** SlickEdit, Sublime, TextMate, TopStyle, UltraEdit, Coda, Dreamweaver, Espresso, Notepad++, PSPad, SciTE, Aptana, Komodo Edit und andere.

Im einfachsten Fall wird ein **Elementname** zum **Elementcontainer** expandiert:

p

wird zu

<p></p>

Hierbei kann ein ID festgelegt werden:

p#beispiel

wird zu

```
<p id="beispiel"></p>
```

Dasselbe funktioniert für CSS-Klassen:

p.eineKlasse

wird zu

```
<p class="eineKlasse"></p>
```

Zwei aufeinanderfolgende Elemente werden durch + abgekürzt:

h1+p

wird zu

```
<h1></h1>
<p></p>
```

Auch Hierarchien können erstellt werden:

ul>li

wird zu

```
<ul>
  <li></li>
</ul>
```

Zusätzlich stehen Multiplikatoren zur Verfügung:

ul>li*3>a

wird zu

```
<ul>
  <li><a href=""></a></li>
  <li><a href=""></a></li>
  <li><a href=""></a></li>
</ul>
```

Mit vordefinierten Kombinatoren wie `html:4t`, `html:4s`, `html:xt`, `html:xs` oder `html:5` können komplette Dokumentrümpfe geschrieben werden:

html:5

wird zu

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>

</body>
</html>
```

Mit Kombinatoren (den runden Klammern) kann zusätzlich gearbeitet werden. Das etwas komplexe Gebilde

```
html:5>div#wrapper>(div#header>p)+(div#navi>ul>li*5>a)+
(div#content>(h1+p))+(div#footer>p)
```

expandiert sich zu:

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <div id="wrapper">
    <div id="header">
      <p></p>
    </div>
    <div id="navi">
      <ul>
        <li><a href=""></a></li>
        <li><a href=""></a></li>
        <li><a href=""></a></li>
        <li><a href=""></a></li>
        <li><a href=""></a></li>
      </ul>
    </div>
    <div id="content">
      <h1></h1>
      <p></p>
```

```
        </div>
        <div id="footer">
            <p></p>
        </div>
    </div>
</body>
</html>
```

Download: <http://code.google.com/p/zen-coding/>

Online-Demo: <http://zen-coding.ru/demo/>

11.2. Remy Sharps HTML5-Shiv

„Ältere“ Versionen des Internet Explorers – und hiermit sind Versionen *vor* IE9 gemeint – sperren sich darin, CSS-Styles auf ihnen unbekannte Elemente anzuwenden, wozu alle in HTML5 neu eingeführten Elemente gehören.

Die semantischen Blockelemente einfach mit einer `display:block`-Anweisung zu versehen, genügt daher nicht:

```
<style type="text/css">

    article {
        display: block;
        border: 1px dotted #aaa
    }

</style>
```

Erscheint nun ein `<article>`-Element im Dokument, wird dies entgegen der Erwartungen weder als Block dargestellt, noch mit einer Border versehen:

```
<article>
    <h1>Ein HTML5 Artikel</h1>
    <p>Bleibt in <em>IE8 und älter</em>
        leider ungestylt.</p>
</article>
```

Eine Lösung für dieses Problem wurde von dem britischen Programmierer Remy Sharp (www.remysharp.com) in ein kleines Script gepackt und zur freien Verwendung zugelassen.

Das Prinzip beruht darauf, dass eine im Vorfeld per JavaScript erstellte Elementinstanz auch eines unbekannten Typs den Internet Explorer (bis zurück zu dessen Version 6!) veranlasst, auch *nachfolgend* im Dokument erscheinende Instanzen dieses Typs zu erkennen und ihnen deklarierte CSS-Styles zuzuordnen.

Die erstellten Elementknoten stören nicht, da sie nicht wirklich in das DOM eingehängt werden (dies ist tatsächlich nicht erforderlich).

Hier eine Ansichtsversion des Scripts., Es steht in Conditional Comments, damit es nur von den Zielbrowsern geladen wird. Der Code sollte frühzeitig ins Dokument eingebunden werden, am besten im Head-Container:

```
<!--[if lt IE 9]>
<script>

// alle zu erstellenden Elementtypen als String:
var e = ("abbr,article,aside,audio,canvas," +
        "datalist,details,figure,footer,header," +
        "hgroup,mark,menu,meter,nav,output," +
        "progress,section,time,video").split(',');

// eine Schleife erstellt je eine Instanz:
for (var i = 0; i < e.length; i++) {
    document.createElement(e[i]);
}

</script>
<![endif]-->
```

Eine stets aktualisierte Version des Scriptes wird bei Google gehostet. Sie können es auf folgendem Weg einbinden (das Script ist komprimiert):

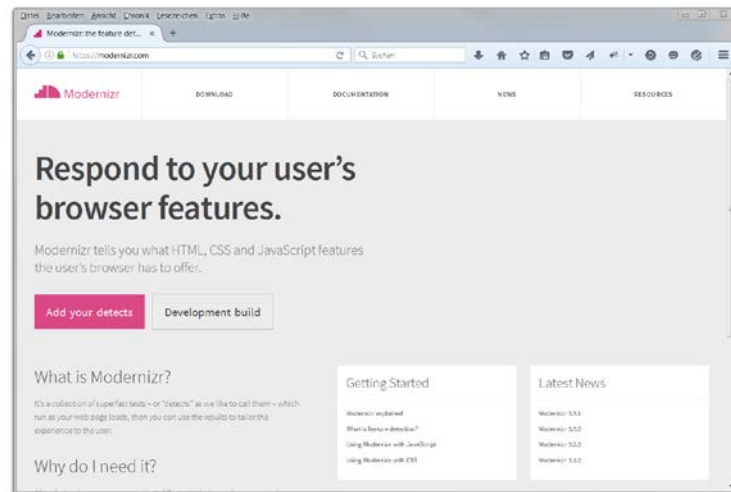
- <http://html5shim.googlecode.com/svn/trunk/html5.js>

Die Einbindung erfolgt wieder in Conditional Comments:

```
<!--[if lt IE 9]>
<script
  src="http://html5shim.googlecode.com/svn/trunk/html5.js">
</script>
<![endif]-->
```

11.3. Modernizr

Über „HTML5 shiv“ hinaus geht das Script **Modernizr** von *Paul Irish* und *Faruk Ate_*. Es beinhaltet sämtliche Funktionen von „HTML shiv“ (um exakt zu sein, beinhaltet Modernizr in der aktuellen Version *tatsächlich* das Script von *Remy Sharp*), prüft aber zusätzlich eine beinahe umfassende Reihe von Funktionalitäten aus HTML5 und CSS3.



Download: www.modernizr.com

- ✓ Nicht vergessen darf man bei der Anwendung von **Modernizr**, dass die Bibliothek *keinerlei Funktionalitäten den Browsern hinzufügt*, sondern ausschließlich die native Unterstützung von Features prüft (also eine **Featuredetection** vornimmt). Das Script ist weder dazu gedacht noch imstande, etwaige funktionale Defizite eines Browsers zu beheben!

11.3.1. Download und Konfiguration

Auf der Downloadseite von Modernizr besteht eine Konfigurationsmöglichkeit, indem die zu prüfenden Features an- oder abgewählt werden können. Dies hat einerseits Auswirkungen auf die Größe der geladenen Datei (eher zu vernachlässigen), andererseits (wichtiger) auf die Verarbeitungsgeschwindigkeit und die Menge der im Lauf des Ladens vergebenen CSS-Klassen.

Desweiteren kann zwischen einer minimierten Version (automatisch) und einer nicht minimierten Developerversion gewählt werden – für letztere muss die **Minify-Option** im linken Menü abgewählt werden.

- ✓ Beachten Sie, dass die **html5shiv-Funktionalität** zur Unterstützung alter, nicht HTML5-fähiger Browser in der aktuellen Modernizr-Version explizit gewählt werden muss!

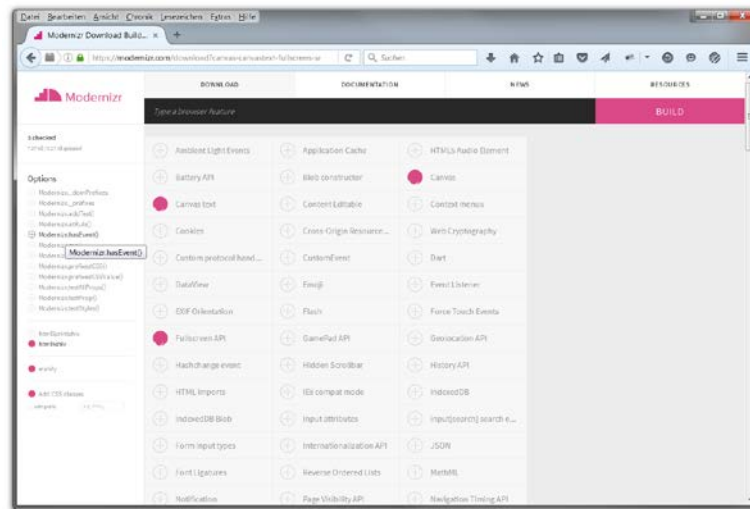


Abb.: Konfiguration von Modernizr (u.a. Canvas u. FullScreen gewählt)

11.3.2. Einbinden von Modernizr

Wie jede Bibliothek muss Modernizr in das Dokument, das durch dieses Feature aufgewertet werden soll, eingebettet werden. Hier der entsprechende Code in vereinfachter HTML5-Schreibweise (davon ausgehend, dass die derzeit **aktuelle Version 3.3.1** von Modernizr verwendet wird – die Versionsnummer kann sich daher ändern):

```
<script src="/js/modernizr.3.3.1.min.js"></script>
```

Als weiterer Schritt ist dem Root-Element des Dokuments von Hand eine **CSS-Klasse „no-js“** hinzuzufügen:

```
<html class="no-js">
```

Diese Klasse dient dem Abfangen des Umstandes, dass JavaScript deaktiviert sein könnte, was die folgende Prüfung aushebeln würde.

- ✓ Ist JavaScript aktiv, so *entfernt* Modernizr in einem ersten Schritt diese Klasse und ersetzt sie durch eine Klasse „js“.

Existiert die Klasse zu einem späteren Zeitpunkt noch, so muss davon ausgegangen werden, dass JavaScript nicht zur Verfügung steht.

Natürlich funktioniert damit auch keine eventuell eingebundene Bibliothek, aber im CSS kann eine Fallbacklösung für diesen Fall vorgesehen sein:

```
/* Styles bei fehlender JS-Unterstützung */

.no-js canvas {
  /* Canvas-Elemente verstecken */
  display:none;
}

...
```

11.3.3. Auswertung der Featureprüfung über CSS-Klassen

Ist JavaScript aktiv, fügt **Modernizr** dem Wurzelement HTML für *jedes* geprüfte Feature eine Klasse hinzu. Diese wechselt je nach Erfolg oder Misserfolg der Prüfung. Das kann wie folgt aussehen:

```
<html class="js canvas canvastext no-geolocation ...">
```

Soeben wurde gefunden, dass der Browser die Canvas API sowie das Erstellen von Texten in Canvas unterstützt, *nicht aber* über Geolocation verfügt. Die weiteren Prüfungen fügen ebenfalls Klassen hinzu, die hier aus Platzgründen weggelassen wurden.

- ✓ Im Grunde sollten **nur benötigte Features geprüft** werden, da alles andere zu unnötig vergebenen, brachliegenden CSS-Klassen führt. *Optimieren* Sie Modernizr beim **Download** entsprechend.

Im CSS können diese Klassen nun eingesetzt werden (dieses Beispiel ist nur illustrativ):

```
/* Canvas wird nicht unterstützt: */
.no-canvas canvas {
  /* Canvas-Elemente verstecken */
  display:none;
}

/* Canvas wird unterstützt: */
.canvas canvas {
  /* Canvas-Elemente zeigen und umranden: */
}
```

```

display:block;
border: 1px dotted #aaa;
}

```

11.3.4. Auswertung der Featureprüfung über JavaScript

Auch auf JavaScript-Ebene ist Modernizr nutzbar. Die Anwendung erzeugt ein **JavaScript-Objekt** namens `Modernizr`, das die geprüften Funktionalitäten in seinen Properties widerspiegelt:

- ✓ Das Canvas-Element wird z. B. getestet, indem **1.)** eine Instanz des Elementtyps erzeugt wird (das geht immer) und dann **2.)** geprüft wird, ob diese über die Methode `getContext()` verfügt (dies ist nur der Fall, wenn die **Canvas API** *tatsächlich* unterstützt wird).

Ist der Test positiv, so enthält `Modernizr.canvas` den Wert `true`.

```

<!-- HTML: -->
<div id="canvasCont">
    <!--Ersatzbild, falls kein Canvas möglich: -->
    
</div>

/* Im CSS: */
.no-canvas div#canvasCont img {
    /* Platzhalterbild zeigen */
    display: block;
}

.canvas div#canvasCont img {
    /* Platzhalterbild verstecken */
    display: none;
}

// Im JavaScript:
if (Modernizr.canvas) {
    var myCanvas = document.createElement('canvas');
    var context = myCanvas.getContext('2d');
    // Grafik erzeugen ...
    // ... und einhängen:
    document.getElementById('canvasCont')
        .appendChild(myCanvas);
}

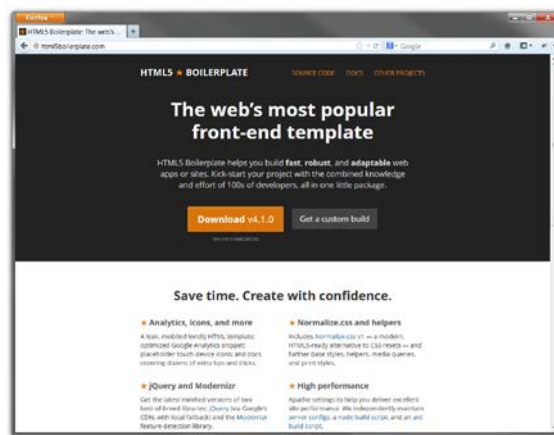
```

}

11.4. HTML5 Boilerplate

Das Templatepaket „HTML5 Boilerplate“ von *Nicolas Gallagher*, *Paul Irish*, *Mathias Bynens*, *Divya Manian* und *Hans Christian Reinl* stellt eine Sammlung aktueller “Best Practices” dar, die es gut geeignet macht, als Ausgangspunkt eines zeitgenössischen Webprojektes zu dienen.

✓ **HTML5 Boilerplate:** <http://html5boilerplate.com/>



Die Website *html5boilerplate.com*

Das **Downloadpaket** enthält neben dem HTML-Template unter anderem auch JavaScript- und CSS-Ressourcen. Aktuell ist die Version 5.3.0.

11.4.1. HTML-Template

Im HTML-Rootelement ist eine „no-js“-Klasse ist bereits gesetzt:

```
<html class="no-js">
```

Ein Meta-Tag stellt sicher, dass IE stets im *vorwärtskompatiblen* Modus mit der jeweils **aktuellen Rendering-Engine** („edge“) betrieben wird:

```
<meta http-equiv="X-UA-Compatible"
      content="IE=edge">
```

Ein weiterer Meta-Tag setzt Randbedingungen für Mobile Browser:

```
<meta name="viewport"
      content="width=device-width, initial-scale=1">
```

Automatisch eingebunden sind zwei CSS-Dateien, die einen allgemeinen Reset bewirken und ein Grundstyling enthalten:

```
<link rel="stylesheet" href="css/normalize.css">
<link rel="stylesheet" href="css/main.css">
```

Als letztes Element im Head ist Modernizr eingebunden – die Einbindung erfolgt im *Head*, damit die CSS-Klassen vor Rendering des Body zur Verfügung stehen:

```
<script
  src="js/vendor/modernizr-2.8.3.min.js"></script>
```

Erst *am Ende* des Bodyinhalts wird **jQuery 1.12.2** eingebunden:

- ✓ Dies geht auf eine Optimierungsprämisse von *Steve Souders* zurück (Prinzip: „DOM ist bereits komplett, Script verzögert das Laden nicht...“):

```
<script src="//ajax.googleapis.com/ajax/libs/
      jquery/1.12.2/jquery.min.js">
</script>
<script>
  window.jQuery ||
  document.write('<script src="js/vendor/jquery-
                1.12.2.min.js"></script>')
</script>
```

Erst wird versucht jQuery aus dem **Google-Repository** zu laden. Schlägt dies fehl (Offline-Betrieb), so wird das lokale jQuery aus dem *vondor*-Ordner geladen.

```
<script src="js/plugins.js"></script>
<script src="js/main.js"></script>
```

Als nächstes werden die (leeren) Stub-Dateien *main.js* und *plugins.js* geladen, die User-Programme aufnehmen sollen.

11.4.2. CSS-Dateien

Die beiden Dateien *normalize.css* und *main.css* befinden sich im Verzeichnis *css/* des Pakets.

- ✓ `normalize.css`:
Nullt Defaultstyles des Browsers und definiert HTML5-Elemente als Blockelemente, sofern diese „unbekannt“ sein könnten.
- ✓ `main.css`:
Setzt verschiedene Basistile konsistent neu.

11.4.3. JavaScript-Dateien

Im Boilerplate-Paket enthalten sind im Verzeichnis `js/` auch verschiedene JavaScript-Dateien.

- ✓ `main.js`:
(leer) Nimmt User-Programme auf
- ✓ `plugins.js`:
Leeres jQuery-Plugin, das ergänzt werden kann.

Im Verzeichnis `vendor/` befinden sich die eingesetzten Frameworks:

- ✓ `jquery-1.12.2.min.js`
Aktuelle Version von jQuery als Backup für die Version aus dem Google-Repository, falls deren Laden fehlschlägt
- ✓ `modernizr-2.8.3.min.js`
Aktuelle Version von Modernizr; wird im Head eingebunden

11.4.4. Weitere Ressourcen

Im Paket enthalten ist desweiteren ein **buntes Sammelurium** an weiteren Dateien, wie Platzhalter für Favicons, Apple-Touch-Icons, 404-Seiten, `robots.txt` etc.

11.5. Initializr

Das **Initializr**-Projekt von *Jonathan Vereccia* ist ein Onlinekonfigurator für HTML5Boilerplate. Sie können hier Ihre gewünschte Variante des Boilerplates konfigurieren und downloaden.

✓ **Initializr:** <http://www.initializr.com/>



Sie haben die Möglichkeit, verschiedene Features von Boilerplate an- oder abzuwählen und auch zwischen unterschiedlichen Templates und CSS-Ressourcen zu wählen:

- ✓ „Klassisches“ Boilerplate
Verwendet das „herkömmliche“ HTML5-Boilerplate-Template
- ✓ Responsive Template
Legt eine Datei zugrunde, die ein „responsive Layout“ ermöglicht
- ✓ Bootstrap Template
Verwendet ein Template, das auf Twitters **Bootstrap** basiert

12. Überblick über die Spezifikationen

12.1. Spezifikationen für Anwendungsprogrammierer

Die folgenden zwei Spezifikationen sind extrem umfangreich, da sie nicht nur die Informationen enthalten, die zur Anwendung von HTML5 erforderlich sind, sondern auch die zur Implementierung in Endgeräte und Anwendungen. Es wurde auf exakte Regelung aller Eventualitäten geachtet, was das Ergebnis leider nahezu unlesbar macht.

HTML5 — A vocabulary and associated APIs for HTML and XHTML

Von:	W3C
URL:	http://dev.w3.org/html5/spec/
	Beschreibt die Markup-Sprache HTML5 und enthält einige der Unterspezifikationen. Ausgelagert und deshalb hier nicht enthalten sind die Spezifikationen zu Microdata, Canvas 2D Context und Web Messaging.

HTML — Living Standard

Von:	WHATWG
URL:	http://www.whatwg.org/specs/web-apps/current-work/multipage/
	Beschreibt übergreifend nicht nur die Markup-Sprache HTML5, sondern enthält auch Spezifikationen, die beim W3C ausgelagert wurden sowie Randthemen wie Web Video Text. Nicht enthalten sind Web Storage, Web Sockets und Server-Events (die allerdings allesamt ohnehin eigentlich nicht Teil von HTML5 sind).

12.2. Spezifikationen für Autoren

Die nachfolgend beschriebenen Dokumente sind Auszüge der offiziellen Spezifikationen, die sich auf Aspekte beschränken, die für Anwender und Autoren von Interesse sind. Aspekte, welche die Implementierung in Anwendungen wie Browsern betreffen, bleiben außen vor.

HTML5: The Markup Language Reference

Von:	Michael[tm] Smith
URL:	http://dev.w3.org/html5/markup/

	Eine Kurzversion der offiziellen W3C-Spezifikation mit Betonung auf „Knappheit und Lesbarkeit“
--	--

HTML5 (Edition for Web Authors)

Von:	Ian Hickson, Google, Inc.
URL:	http://dev.w3.org/html5/spec-author-view/
	Ausführlicher als die vorgenannte Kurzfassung, orientiert sich diese stärker an der W3C-Version, lässt allerdings alle Implementierungsaspekte weg.

HTML5 for Web Developers

Von:	Ben Schwarz
URL:	http://developers.whatwg.org/
	Vergleichbar mit der Kurzfassung von Hickson nimmt Schwarz in seine Version auch die Beschreibung von Microdata auf.

12.3. Weitere Spezifikationen

Neben den Spezifikationen, die die eigentliche Markup-Sprache betreffen, existieren weitere Spezifikationen, die zwar nicht zum Sprachkern gehören, aber in Zusammenhang mit HTML5 von Bedeutung sind. Die folgenden Spezifikationen können somit dem „inneren Kreis“ hinzugerechnet werden:

Geolocation API Specification:

www.w3.org/TR/geolocation-API/

Web Storage Specification:

<http://dev.w3.org/html5/webstorage/>

The WebSocket API:

<http://dev.w3.org/html5/websockets/>

Web SQL Database:

<http://www.w3.org/TR/webdatabase/>

Web Workers:

<http://dev.w3.org/html5/workers/>

HTML5 Vocabulary:

<http://dev.w3.org/html5/vocabulary/>

HTML5 Offline Web Applications:

<http://www.w3.org/TR/offline-webapps/>

HTML5 Server-Sent Events:

<http://dev.w3.org/html5/eventsource/>

HTML Design Principles:

<http://www.w3.org/TR/html-design-principles/>

HTML Microdata:

<http://dev.w3.org/html5/md/>

HTML5 Web Messaging:

<http://dev.w3.org/html5/postmsg/>

HTML+RDFa 1.1:

<http://dev.w3.org/html5/rdfa/>

13. Literatur

HTML5 und CSS3 – Das umfassende Handbuch

Jürgen Wolf

(Rheinwerk Verlag, 2015)

HTML5 Handbuch

Clemens Gull und Stefan Münz

(Franzis Verlag, 2014)

HTML5. Webseiten innovativ und zukunftssicher, 2. Aufl.

Peter Kröner

(Open Source Press, 2011)