Comp 388/424 - Client-Side Web Design

Spring Semester 2016 - Week 1

Dr Nick Hayward

Course Details

Lecturer

- Name: Dr Nick Hayward
- Office: 531 Lewis Towers (WTC)
- Office hours
  - *Monday afternoon by appointment (WTC)*
- Faculty Page

TA

- Name: Katharine Herringshaw
- Email: kherringshaw@luc.edu

Course Schedule

Important dates for this semester

- Monday @ 4.15pm to 6.45pm (6.30pm with no break)
    - *Corboy Law Center, Room 302, WTC*
- DEV week: 7th to 14th March 2016
    - *DEV week presentation due on 14th March 2016 @ 4.15pm*
- Spring break: 7th to 12th March 2016
    - *No class: 7th March 2016*
- Final class: 25th April 2016
    - *Final presentation due on 25th April 2016 @ 4.15pm*
- Exam week: 2nd May to 7th May 2016
    - *Final assessment due on 2nd May 2016 by 4.15pm*

Initial Course Plan - Part I

(up to ~ Spring Break)

- Build and publish a web app from scratch
  - *general setup and getting started*
  - *maintenance and publication*
  - *basic development and manipulation (HTML, JS...)*
  - *add some fun with Ajax, JSON, server-side...*
  - *useful data storage techniques and options*
  - *testing...*

Initial Course Plan - Part 2

(Up to the end of the semester)

- Augment and develop initial app
- Explore other options
  - *publication frameworks*
  - *further libraries and options*
  - *tools and workflows*
  - *visualisations, graphics...*
  - *publish (again...)*

Assignments and Coursework

Course will include

- weekly bibliography and reading (where applicable)
- weekly notes, examples, extras...

Coursework will include

- quizzes or group exercises at the end of each section (Total = 30%)
    - *based on course notes, reading, and examples*
- development and project assessment (Total = 70%)
    - *mid-semester assessment (Total = 30%)*
        - end of DEV week
        - demo due 14th March 2016 @ 4.15pm
    - *final assessment (Total = 40%)*
        - demo due 25th April 2016 @ 4.15pm
        - report due 2nd May 2016 @ 4.15pm

Quizzes, group exercises...

Course total = 30%

- at least one week notice before quiz
  - *average time ~30 minutes (can be extended...)*
  - *taken towards the end of class*
- group exercises
  - *help develop course project*
  - *test course knowledge at each stage*
  - *get feedback on project work*

Development and Project Assessment

Course total = 70% (Parts 1 and 2 combined total)

Initial overview

- combination project work
  - *part 1 = mid-semester **DEV Week** work (30%)*
  - *part 2 = final demo and report (40%)*
- group project (max 4 persons per group)
- design and develop a web app
  - *purpose, scope etc is group's choice*
  - ***no** blogs, to-do lists, note-taking...*
  - *chosen topic requires approval*
  - *must implement data from either self-hosted data, public API, or combination of both*

DEV Week Assessment

- web app developed from scratch
  - *examples, technology etc outlined during first part of semester*
- demo and project report
  - *due on 14th March 2016 @ 4.15pm*
- anonymous peer review
  - *similar to user comments and feedback*
  - *chance to respond to feedback before final project*

Final Assessment

- working final app
- presentation can be a live demo, slides, video...
  - *due on 25th April 2016 @ 4.15pm*
  - *show and explain implemented differences from DEV week project*
  - *where and why did you update the app?*
  - *benefits of updates?*
- how did you respond to peer review?
- final report
  - *due on 2nd May 2016 @ 4.15pm*

Goals of the course

A guide to developing and publishing interactive client-side web applications and publications.

Course will provide

- guide to developing client-side web applications from scratch
- guide to publishing web apps for public interaction and usage
- best practices and guidelines for development
- fundamentals of web application development
- intro to advanced options for client-side development
- ...

Course Resources

Website

Course website is available at https://csteach424.github.io

- timetable
- course overview
- course blog
- weekly assignments & coursework
- bibliography
- links & resources
- notes & material

GitHub

Course repositories available at https://github.com/csteach424

- weekly notes
- examples
- source code (where applicable)

Group projects

- add project details to Trello organisation
  - *weekly assignments part of each section's assessment*
- create channels on Slack for group communication
- start working on an idea for your project
- plan weekly development up to and including DEV Week
  - *7th to 14th March 2016*
  - *DEV week presentation due on 14th March 2016 @ 4.15pm*

Intro to Client-side web design

- allows us to design and develop online resources and publications for users
  - *both static and interactive*
- restrict publication to content
  - *text, images, video, audio...*
- develop and publish interactive resources and applications
- *client-side scripting* allows us to offer
  - *interactive content within our webpages and web apps*
- interaction is enabled via code that is downloaded and compiled, in effect, by the browser
- such interaction might include
  - *a simple mouse rollover or similar touch event*
  - *user moving mouse over a menu*
  - *simple but effective way of interacting*

Client-side and server-side - Part 1

Client-side

- scripts and processes are run on the user's machine, normally via a browser
    - *source code and app is transferred to the user's machine for processing*
- code is run directly in the browser
- predominant languages include HTML, CSS, and JavaScript (JS)
    - *HTML = HyperText Markup Language*
    - *CSS = Cascading Style Sheets*
    - *many compilers and transpilers now available to ease this development*
        - eg: *Go to JavaScript...*
- reacts to user input
- code is often visible to the user (source can be read in developer mode etc...)
- in general, cannot store data beyond a page refresh
- in general, cannot read files directly from a server (HTTP requests required)
- single page apps create rendered page for the user

Client-side and server-side - Part 2

Server-side

- code is run on a server
  - *languages such as PHP, Ruby, Python, Java, C#...*
  - *in effect, any code that can run and respond to HTTP requests can also run a server*
- enables storage of persistent data
  - *data such as user accounts, preferences...*
- code is not directly visible to the user
- responds to HTTP requests for a given URL
  - *not direct user input of any kind*
- can render the view for the user on the server side

and so on...

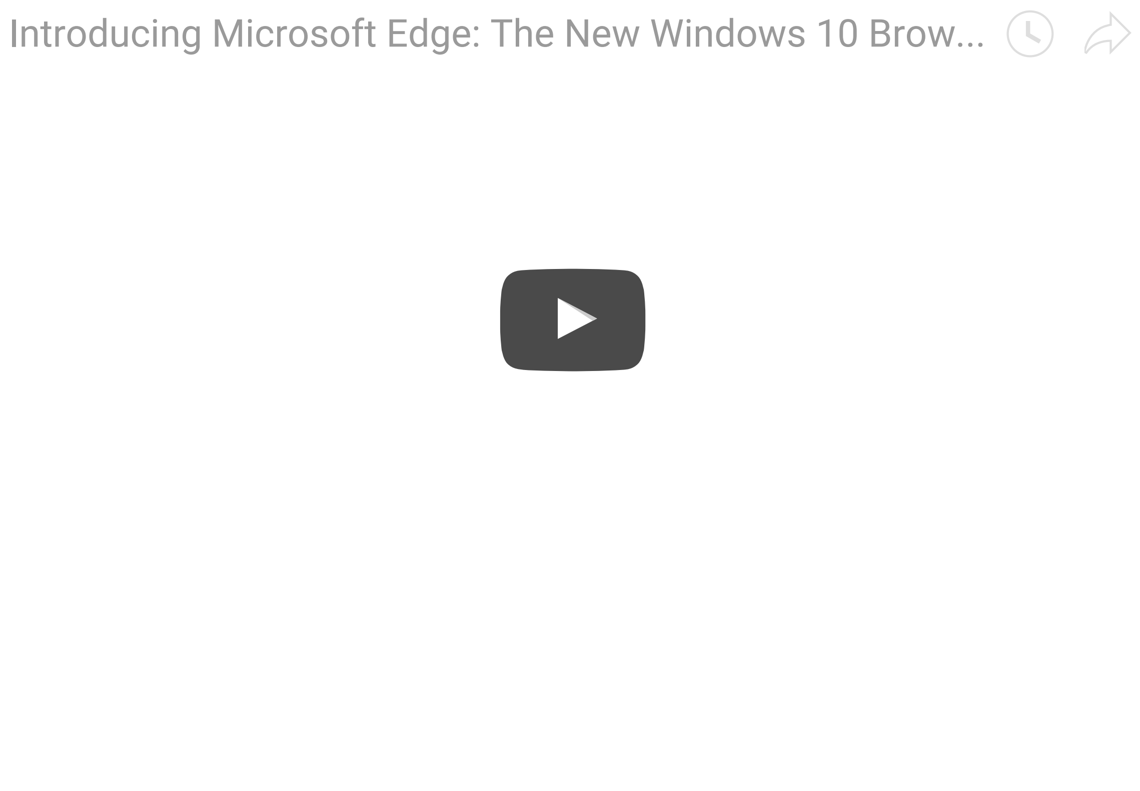
Getting started

- basic building blocks include HTML, CSS, and JS
- many tools available to work with these technologies
- three primary tools help with this type of development
- web browser
  - *such as Chrome, Edge (IE?), Firefox, Opera, Safari...*
- editor
  - *such as Atom, Sublime, Microsoft's Visual Studio Code...*
- version control
  - *Git, Mercurial, Subversion*
  - *GitHub, Bitbucket...*

Getting started - Web Browsers

- choose your favourite
  - *Chrome, Firefox, Safari, Edge...*
  - *not IE*
- developer specific tools
  - *Chrome etc view source, developer tools, JS console*
  - *Firefox also includes excellent developer tools*
  - *Firebug*
- cross-browser extension for web developers
  - *Web Developer*

Video - Microsoft Edge


Introducing Microsoft Edge: The New Windows 10 Brow...

Source - YouTube - Introducing Microsoft Edge

Getting started - Editors

Many different choices including

Linux, OS X, and Windows

- Atom
- Sublime
- Visual Studio Code
    - ***NB:*** *in preview, but interesting to test*

OS X specific

- BBEdit
    - *TextWrangler*

and so on.

Video - Atom 1.0

Source - YouTube - Introducing Atom 1.0

Browser technologies

- browser rendering engines
- web standards
  - *HTML*
  - *CSS*
  - *XML*
  - *XHTML*
- application foundations
- open web platform

Browser rendering engines

- Until 2013, *WebKit* was the default rendering engine for both Safari and Chrome
- Google switched to the open source alternative, *Blink*, whilst Safari continues to use *WebKit*
- Firefox continues to use the *Gecko* rendering engine
- Microsoft's new Edge browser uses a new proprietary engine called *EdgeHTML*
  - *fork of the Trident rendering engine*
  - *Microsoft notes that EdgeHTML will largely behave like Chrome and Safari*

Web standards

- many disparate web standards
  - *include the broader internet beyond www...*
  - *subset of particular interest to web developers*
- primary web standards
  - ***Recommendations** published by the W3C (World Wide Web Consortium)*
  - ***Unicode** standards published by the Unicode Consortium*
  - ***ECMA** standards now published by ECMA International*
  - *examine with React etc*

W3C Recommendations - part 1

Recommendations of the W3C of particular interest includes

- HTML (HyperText Markup Language)
  - *key building block of the web*
  - *stored as plain text*
  - *includes selection of tags*
  - *eg: headings, images, links, lists, paragraphs, tables...*
- CSS (Cascading Style Sheets)
  - *commonly used with HTML*
  - *controls rendering and stylistic characteristics of a web page*
  - *CSS concerned with presentation of the structure and data*

W3C Recommendations - part 2

Recommendations of the W3C of particular interest includes

- XML (Extensible Markup Language)
  - *often considered a meta-language*
  - *follow-on from SGML (Standard Generalised Markup Language)*
  - *used to describe data & not presentation, rendering of data*
  - *element tags not inherently pre-defined*
  - *foundation for many XML languages such as RSS, MathML, MusicML...*
- XHTML (Extensible HyperText Markup Language)
  - *attempt to update and rewrite HTML based on experience from XML*
  - *very similar to HTML with stricter rules*
  - *eg: HTML lapse in enforcing case sensitivity, closing tags...*
  - *strict rules structure inherited from XML style languages*

Video - W3C Web standards for the future

Web standards for the future

from **W3C** PLUS

01:50

CC

Source - Vimeo - W3C

Application foundations - Part 1

W3C, on the occasion of HTML5 achieving the status of W3C Recommendation, proposed

> *a set of technologies for developing distributed applications with the greatest interoperability in history. Application Foundations for the Open Web Platform*

- known as the OWP (Open Web Platform)
- driven by a blog post by Jeff Jaffe in October 2014
  - *suggested W3C's next priority should be Open Web Platform*
  - *OWP should be easier to use for developers*

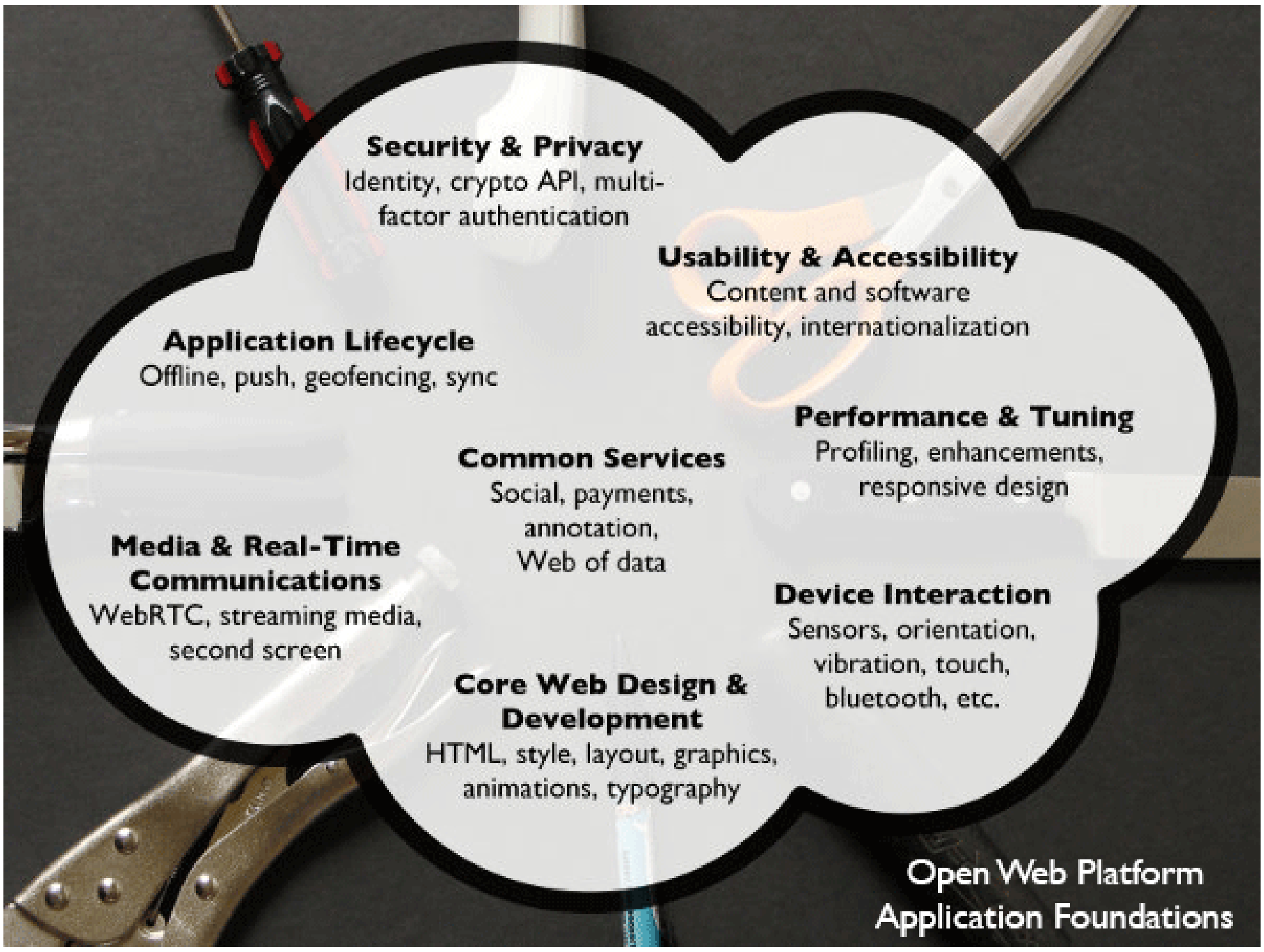
Application foundations - Part 2

Jaffe defined eight **Foundations** in that particular post, which include the following

- Security and Privacy
- Core Web Design and Development
- Device Interaction
- Application Lifecycle
- Media and Real-Time Communications
- Performance and Tuning
- Usability and Accessibility
- Services

Further information and updates can be found at the W3C's App Foundations website.

Image - Open Web Platform


Security & Privacy
Identity, crypto API, multi-factor authentication
Usability & Accessibility
Content and software accessibility, internationalization
Application Lifecycle
Offline, push, geofencing, sync
Performance & Tuning
Profiling, enhancements, responsive design
Common Services
Social, payments, annotation, Web of data
Media & Real-Time Communications
WebRTC, streaming media, second screen
Device Interaction
Sensors, orientation, vibration, touch, bluetooth, etc.
Core Web Design & Development
HTML, style, layout, graphics, animations, typography
Open Web Platform Application Foundations

Source - W3C

HTML - Intro

- acronym for *HyperText Markup Language*
- simple way to structure visual components of a website or web application
- HTML also uses keywords, or element tags
  - *follow a rigidly defined syntax*
- helps us to create web pages and web applications
  - *web browsers, such as Chrome or Firefox, may render for viewing*
- an error can stop a web page from rendering
  - *more likely it will simply cause incorrect page rendering*
- interested in understanding the core of web page designing
  - *understand at least the basics of using HTML*

HTML - Element syntax - part 1

Constructed using elements and attributes, which are embedded within an HTML document.

Elements should adhere to the following,

- start with an opening element tag, and close with a matching closing tag
    - *names may use characters in the range **0-9**, **a-z**, **A-Z***
- content is, effectively, everything between opening and closing element tags
- elements may contain empty content
- empty elements should be closed in the opening tag
- most elements permit attributes within the opening tag

HTML - Element syntax - part 2

An element's *start* tag adheres to a structured pattern, which may be as follows,

1. a **<** character
2. tag name
3. optional **attributes**, which are separated by a space character
4. optional space characters (one or more...)
5. optional **/** character, indicating a **void** element
6. a **>** character

For example,

```
<div>
```

HTML - Element syntax - part 3

An element's *end* tag also adheres to a pattern, again exactly as defined as following,

1. a **<** character
2. a **/** character
3. element's tag name (ie: name used in matching start tag)
4. optional space characters (one or more...)
5. a **>** character

For example,

```
</div>
```

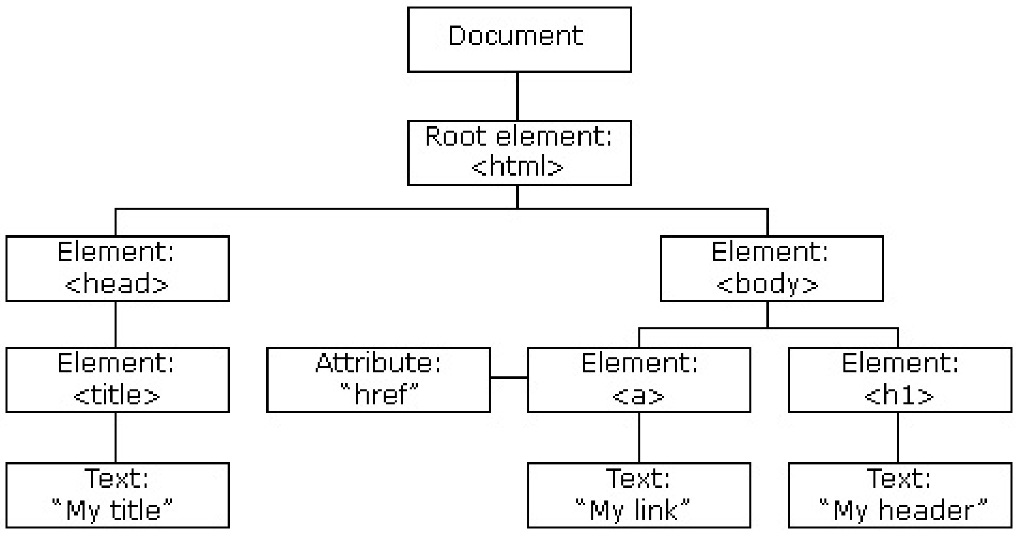
NB: void elements, such as `<br />` or `<img />`, must *not* specify end tags.

HTML - Element syntax - part 4

- HTML, XHTML, can be written to follow the patterns and layouts of XML
- HTML elements can also be nested with a parent, child, sibling...
  - *relationship within the overall tree data structure for the document*
- as the HTML page is loaded by a web browser
  - *the HTML DOM (document object model) is created*
- basically a tree of objects that constitutes the underlying structure
  - *the rendered HTML page*
- DOM gives us an API (application programming interface)
  - *a known way of accessing, manipulating the underlying elements, attributes, and content*
- DOM very useful for JavaScript manipulation

Image - HTML DOM Tree of Objects

W3C DOM Tree



Source - W3C

HTML - Attribute syntax - part 1

- HTML attributes follow the same design pattern as XML
- provide additional information to the parent element
- placed in the opening tag of the element
- follow the standard syntax of name and value pairs
- many different permitted legal attributes in HTML
- four common names that are permitted within most HTML elements
    - *`class`*, *`id`*, *`style`*, *`title`*

HTML - Attribute syntax - part 2

Four common names permitted within most HTML elements

- class
  - *specifies a classname for an element*
- id
  - *specifies a unique ID for an element*
- style
  - *specifies an inline style for an element*
- title
  - *specifies extra information about an element*
  - *can be displayed as a tooltip by default*

NB:

- cannot use same name for two or more attributes
- regardless of case
- on the same element start tag.

HTML - Attribute syntax - part 3

A few naming rules for attributes

- empty attribute syntax
  - `<input disable>`
- unquoted attribute-value syntax
  - `<input value=yes>`
  - *value followed by /, at least one space character after the value and before /*
- single quoted attribute-value syntax
  - `<input type='checkbox'>`
- double quoted attribute-value syntax
  - `<input title="hello">`

NB:

- further specific restrictions may apply for the above
- consult W3 Docs for further details
- above examples taken from W3 Docs - Syntax Attributes Single Quoted

HTML - Doctype - part 1

- `doctype` or `DOCTYPE` is a special instruction to the web browser
  - *concerning the required processing mode for rendering the document's HTML*
- `doctype` is a required part of the HTML document
- first part of our HTML document
- should always be included at the top of a HTML document, eg:

```
<!DOCTYPE html>
```

- doctype we add for HTML5 rendering
- not a HTML element, simply tells the browser required HTML version for rendering

HTML - Doctype - part 2

- HTML4 needs to specify the required *DTD* (document type definition)
  - *legacy of that version's origins in SGML*
- HTML4 can specify different types of documents
  - *helps the browsers render the page correctly, and as expected*
- different types include
  - *strict*
    - contains all HTML elements and attributes (excluding presentation & deprecated elements such as `font`, & no framesets)
  - *transitional*
    - contains all HTML elements and attributes (including presentational & deprecated elements such as `font`, & no framesets)
  - *frameset*
    - same as transition DTD, but allows the use of framesets
  - *XHTML 1.0 strict*
  - *XHTML 1.0 transitional*
  - *XHTML 1.0 frameset*
- more recent XHTML 1.1 DTD also available
  - *follows pattern of XHTML 1.0 strict*
  - *adds support for modules such as Ruby...*

HTML - Doctype - part 3

HTML4 Doctype examples include:

- strict

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

- transitional

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

- frameset

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
```

HTML - Doctype - part 4

XHTML Doctype examples include:

- XHTML 1.0 strict

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

- XHTML 1.0 transitional

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

- XHTML 1.0 frameset

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

- XHTML 1.1

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

HTML - Character encoding - part 1

- element text, and attribute values, must consist of defined **Unicode** characters
    - *The Unicode Consortium*
    - *Unicode Information*
        - Unicode examples - many, many examples...

- As with most things, there are some exceptions
    - *must not contain `U+0000` characters*
    - *must not contain permanently undefined Unicode characters*
    - *must not contain control characters other than space characters*
        - Space `U+0020`
        - Tab `U+0009`
        - Line feed `U+000A`
        - Form feed `U+000C`
        - Carriage return `U+000D`

HTML - Character encoding - part 2

Basically, we use the following definable types of text for content etc.

- normal character data
  - *this includes standard text and character references*
  - *cannot include non-escaped < characters*
- replaceable character data
  - *includes elements for `title` and `textarea`*
  - *allows text, including non-escaped < characters*
  - *character references*
    - a form of markup for representing single characters
    - eg: a dagger `&dagger;` or `&#8224;` or `&#x2020;`

XHTML vs HTML - part 1

- XHTML is often described as HTML redesigned as XML
- XHTML enforces correct markup of HTML
  - *follows same patterns of well-formed XML documents*
- primary differences between HTML and XHTML include
  - *XHTML DOCTYPE is* ***mandatory***
  - *`xmlns` attribute in `<html>` element is* ***mandatory***
  - ***mandatory*** *elements in XHTML include*
    - `<html>`, `<head>`, `<title>`, and `<body>`

XHTML vs HTML - part 2

Example XHTML 1.0 Strict template

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
   <head>
      <title>XHTML 1.0 Strict</title>
   </head>
   <body>
   </body>
</html>
```

XHTML vs HTML - part 3

XHTML elements adhere to the following rules,

- proper nesting
  - *elements must not overlap other elements*
    - breaks the underlying tree DOM for the page. eg:

```
<!-- incorrect overlapping -->
<div><p>some text...</div></p>
<!-- nesting -->
<div><p>some text...</p></div>
```

- must always be closed
  - *all elements must be closed with a matching closing tag eg:*

```
<!-- incorrect -->
<p>some text...
<!-- correct -->
<p>some text...</p>
```

- empty elements must also be closed correctly

```
<!-- incorrect -->
<br >
<!-- correct -->
<br />
```

- must be in lowercase
- must have a root element

XHTML vs HTML - part 4

XHTML attributes adhere to the following rules,

- must be lower in case
- must be *quoted*
    - *double quotes is standard for attribute values. eg:*

```
<!-- incorrect -->
<p class=content>
<!-- correct -->
<p class="content">
```

- minimisation is forbidden
    - *must include quoted value. eg:*

```
<!-- incorrect -->
<input checked>
<!-- correct -->
<input checked="true">
```

XHTML vs HTML - part 5

We can also update and convert legacy HTML code using the following options,

- every page needs to include an XHTML doctype declaration, eg:

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

- we can also add an `xmlns` attribute to the `html` element of every page, eg:

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

- update element names to ensure they are all **lowercase**
- ensure all elements are correctly closed
- update all attribute names to lowercase
- ensure all attribute values are correctly quoted

XHTML vs HTML - part 6

We can then double-check our XHTML using the W3C's validator,

- Markup Validation Service

References

- Jaffe, Jim., *Application Foundations For The Open Web Platform*. W3C. 10.14.2014. http://www.w3.org/blog/2014/10/application-foundations-for-the-open-web-platform/
- The Unicode Consortium
- Unicode Information
- Unicode examples
- W3 Docs for further details
- W3Schools - DOM Image