

XML

What is markup?

Markup is a type of annotation

- Markup has been used for centuries in publishing as a means of providing formatting instructions to printers (human and machine).
- “Instructions for the typesetter that are written on the copy (e.g. underlining words that are to be set in italics).” - Webster’s Online Dictionary

The importance of formatting

- Textual formatting has a lot to do with how we understand written text.
- Texts tend to have a familiar structure.
- We use non-alphanumeric characters to indicate meaning.
- We also use different *styles* to affect meaning.

an example

What is this?

Οἱ μὲν ἰππήων στρότον, οἱ δὲ πῆσδων,
οἱ δὲ νάων φαῖσ' ἐπὶ γᾶν μέλαιναν
ἔμμεναι κάλλιστον. ἔγω δὲ κῆν' ὄτ-

τω τις ἔραται.

πά]γχυ δ' εὐμαρες σύνετον πόησαι
πά]ντι τ[οῦ]τ'. ἄ γὰρ πόλυ περσκόπεισα
κά]λλος ἀνθρώπων Ἑλένα [τὸ]ν ἄνδρα
[κρίννεν ἄρ]ιστον,

Poetry, obviously

A bit easier to figure out than this:

ΟΙΜΕΝΙΠΠΗΩΝΣΤΡΟΤΟΝΟΙΔΕΠΕΣΔΩΝ
ΟΙΔΕΝΑΩΝΦΑΙΣΕΠΙΓΑΝΜΕΛΑΙΝΑΝΕΜΜ
ΕΝΑΙΚΑΛΛΙΣΤΟΝΕΓΩΔΕΚΗΝΟΤΤΩΤΙΣΕΡ
ΑΤΑΙ...

(which is how it might have been
written down originally)

Some thoughts:

- Markup is meta-textual information.
- That is, it provides information on how a text should be interpreted and re-rendered.
- Formatting influences semantics, *i.e.*, it implies meaning.
- So we see (and use) some types of markup all the time...

Examples of markup

- Highlighted sections of text.
- Corrections on a “marked” paper.
- Marginal comments.
- But we might also understand markup to trespass into the text itself.
 1. Punctuation.
 2. Spacing.

Markup Languages

- Markup languages provide standardized ways of annotating and structuring texts and data.
- These languages do things like:
 1. Provide formatting instructions to a rendering engine.
 2. Make semantic distinctions between portions of a text.
 3. Represent the underlying structure of a document.
 4. Add metadata to a document.

Let's get a bit less abstract

- HTML stands for HyperText Markup Language.
- So in what senses is this markup?
 1. HTML provides structure
 2. and formatting instructions
 3. and *some* semantics
 4. and allows the addition of metadata
 5. and provides linking mechanisms.

Let's get a bit more abstract again...

- HTML is a markup language, in the “proper” sense of the term.
- HTML is an *application* of SGML, Standard Generalized Markup Language.
- SGML, even though it has the words “markup language” in its name is really a syntax for creating markup languages, not a markup language itself.
- XML is a derivative of SGML. In fact, it *is* SGML, with more restrictions than standard SGML.

So XML is:

- A set of rules. If a document conforms to these rules, it is an XML document, or XML *instance*.
- An XML *application* is a grammar that specifies what tags can be used, and where.
- An XML *instance* is a document marked up in XML, **whether it uses a grammar or not.**

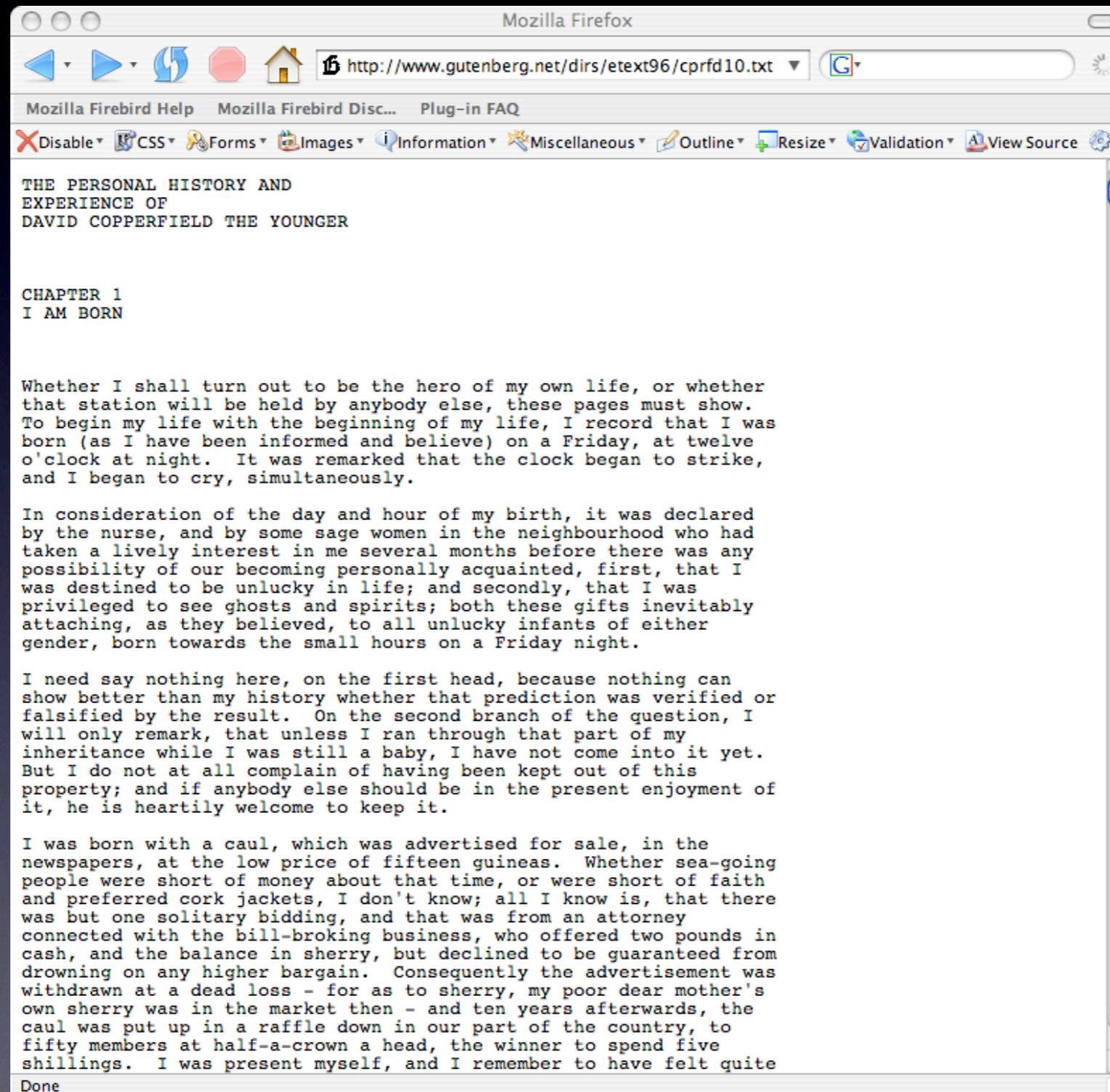
Two basic types of XML

- We've been assuming that XML is for texts. Actually it's commonly used for more rigidly structured kinds of data too.
- So you'll hear lots of talk about data-oriented vs. document-oriented XML.
- This is important because the two tend to pull in opposing directions.

Thought experiment

- Let's imagine a database with tables containing data. Maybe a directory with names, addresses, and phone numbers.
- Now, how would we store a book (say *David Copperfield*) in a structure like that?

The text



Tables

- We could have a "words" table:

id	word
1	Whether
2	I
3	shall
4	turn
5	out
6	to
7	be
8	the
9	hero
10	of
11	my
12	own
13	life
14	,
15	or
16	that

Tables

- And an "order" table:

id	order
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
1	16
16	17

- and so on. Unwieldy though, isn't it? What would we do about paragraphs? Chapters?

The moral of the story:

- Information that is loosely structured is hard to store in highly structured containers, such as relational databases.
- The converse is true also: highly structured information really benefits from the things that highly structured containers have to offer, like data typing, enforcement of relational integrity, constraints, etc.

data vs. document

- So these two kinds of information have fundamental differences...
- And XML handles both...

Break

XML Boot Camp

the nuts and bolts

XML is text

- That's actually a more involved assertion than it seems...
- Any program that reads text can open XML files.
- Any XML file can be opened and edited using any text editor.
 - Common text editors include: pico, vi, emacs, Notepad, TextEdit, UltraEdit, jEdit, and so on...

XML is structured

- A typical XML document has at least four components:
 - an XML declaration
 - Elements
 - The elements may have Attributes
 - Text inside elements & attributes

XML declaration

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

Elements

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<root>
```

```
  <another/>
```

```
</root>
```

The root element. There can be only one. The name doesn't matter.

Attributes

```
<?xml version="1.0" encoding="UTF-8"?>  
<root type="penguin">  
  <another type="fruit bat"/>  
</root>
```

Text

```
<?xml version="1.0" encoding="UTF-8"?>  
<root type="penguin">  
  <another type="fruit bat"/>  
  <textElt>Here is some text</textElt>  
  <div>  
    <p>Here is some more.</p>  
  </div>  
</root>
```

Other creatures of the forest

- Doctype declarations
- Entity References
- Character References
- Processing Instructions
- Comments
- CDATA
- Namespaces

DOCTYPE Declarations

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

```
<!DOCTYPE TEI.2 SYSTEM "http://docsouth.unc.edu/dtds/teixlite.dtd">
```

DOCTYPE

Declarations

- May show up in XML (and will show up in SGML) instance documents.
- Declare that the document conforms to a particular XML grammar.
- This grammar is specified by a DTD (Document Type Definition).

DTDs

- We'll go into more details later, but...
- DTDs can occur as part of an XML instance or can be linked to them (or both).
- They use a non-XML syntax.

Entities

- Entities are important and useful
- XML has five “built-in” entities
 - `<` = `<`
 - `>` = `>`
 - `"` = `"`
 - `'` = `'`
 - `&` = `&`
- and you can declare your own, too.

Character References

- These are basically entities.
- `Д` = Д (Cyrillic capital letter 'de')
- They provide a means of entering characters that may not be supported by your system.

What the heck was that again?

- We're getting into the technologies that underlie XML.
- XML is text. Let's think about what that means...
- To a computer, a character is just a number...everything is a number, actually.

Fun with binary

- If you've heard this before, or are just really bored by it, I apologize.
- Most modern computers organize data into bytes, which are 8 bits long.
- A bit is a 1 or 0, on or off, high or low voltage or radio frequency.

Fun with binary

- $11111111 = 255$, so between zero and the highest 8-digit binary number, you have 256 possibilities.
- The extended ASCII encoding uses these 256 codes for letters, punctuation, control codes, etc.
- Hexadecimal (base-16) is a convenient way of writing these things. $255 = FF$.

This one goes up to 1,114,111...

- 256 characters are clearly not enough. What if you want to type in Sanskrit, Chinese, or Ogham?
- Enter Unicode, an encoding that allows for up to 4-byte character codes.
- And that's why there's an encoding attribute on the XML declaration.

Comments

- Just like HTML:
 - `<!-- comment -->`
- Comments are a way of annotating an instance document with information that may be useful to someone reading it.

CDATA

- Provides another way of “escaping” text.
- You’d use CDATA sections when you had a lot of ‘<’s, ‘>’s, and ‘&’s that you wanted in your document.
- `<![CDATA[<whatever>]]>`

Processing Instructions

- `<?instruction type="doSomething"?>`
- PIs are what they sound like: messages to a particular program that may be doing something with the XML.
- They aren't used all that much, but you do see them sometimes.

Namespaces

These are
different

```
<?xml version="1.0" encoding="UTF-8"?>
<root type="penguin">
  <another type="fruit bat"/>
  <textElt>Here is some text</textElt>
  <ns:textElt xmlns:ns="http://some.url.nowhere">More text</ns:textElt>
  <div>
    <p>Here is some more.</p>
  </div>
</root>
```

Namespaces

- You'll actually see these a lot.
- They frequently use URLs as namespaces, which is confusing.
- There is absolutely no guarantee the URLs go anywhere.
- Namespaces are for distinguishing elements from each other. They prevent name

The Rules

- All XML documents have one, and only one root element.
- Elements can be empty, contain other elements, text, or a mixture.
- Elements must be properly nested. You can't do things like:
 - `<a>`

The Rules

- Any element that is opened must be closed.
 - Standalone elements can close themselves (`
` == `
</br>`).
- Text must be contained within elements or comments.
- Attribute values must be quoted.

The Rules

- Attributes must use plain quotes: " or ', not “smart quotes”.
- There can be **nothing** before the XML declaration (or the first element/ DOCTYPE).
- There can be absolutely no stray angle brackets or ampersands.

The Rules

- Extra “whitespace” doesn’t matter.
- There are absolutely no exceptions to the standard syntax.
- Break any of these and your document is not XML.
- That’s not everything, but those are the basics.

That's it!

- You now know enough to be dangerous.