SENG1050

Web Technologies

Week 03 XML and DTD

2

Lecture Plan

Weekly program (lectures)

- ✓ Week 1 – The Internet, Protocols, TCP/IP, Email, HTTP
- ✓ Week 2 – HTML basics
- ❑ **Week 3 – XML and DTD**
- ❑ Week 4 – CSS
- ❑ Week 5 – More HTML with CSS
- ❑ Week 6 – Revision and Midterm
- ❑ Week 7 – XSLT
- ❑ Week 8 – JavaScript
- ❑ Week 9 – More JavaScript and User Interface
- ❑ Week 10 – Encoding, Compression and Information Retrieval
- ❑ Week 11 – Security and Encryption
- ❑ Week 12 – Ethics and Course review

3

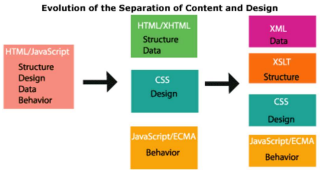
Week 03 Lecture Outline

XML and DTD

- ❑ What is XML?
- ❑ Structure and elements of XML
- ❑ XML Parsing: CDATA and PCDATA
- ❑ XML Tree
- ❑ Document Type Definition (DTD)
- ❑ DTD elements, attributes, entities
- ❑ Linking DTD and XML files

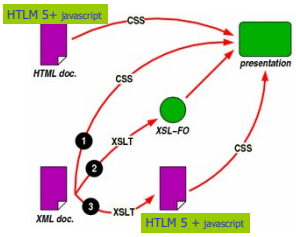
4

- Week 3: XML
- Week 4: CSS
- Week 5: HTML and CSS
- Week 6: Mid Term Exam
- Week 7: XSLT
- Week 8: JavaScript



5

- Week 3: XML
- Week 4: CSS
- Week 5: HTML and CSS
- Week 6: Mid Term Exam
- Week 7: XSLT
- Week 8: JavaScript



6

Dealing with data

- An example of Computer vs Human interpretation
  - Computer
    - Time 10:30pm
    - Place Sydney
  - Human
    - I need to be on a plane that leaves Sydney at 10:30pm otherwise I won't get to Los Angeles
- Humans need to know the extra *context* as well as the data

Dealing with data

7

- Problem: How can we represent data so that **computers** and **humans** can both read it in the way that they understand?

- Answer: XML

What is XML?

8

- XML…
  - stands for eXtensible Markup Language
  - A standard **set of design rules** to make data readable by humans and computers
  - A document that follows these rules is an XML Document or an XML Application

What is eXtensible?

9

- An important thing to know is the meaning of eXtensible…
  - It means that the user has full control over the data, context, and patterns used in a XML document
  - Therefore, an XML document can contain *any* data the user wants
  - XML can thus be used for many different purposes.

Think of XML like a template or framework.

The XML Family

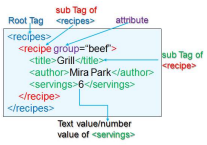
10

- XML = eXtensible Markup Language
  - A syntax for arbitrary semantic tags
- DTD = Document Type Definition
  - Defines the structure (pattern) of a valid XML document
- XSLT = eXtensible Stylesheet Language Transformations
  - An XML document that describes how to transform another XML document into a different format.
- There are some other members such as *XML Schema* and *XSL-FO* that perform similar tasks to DTD and XSLT, but these will not be covered in this course.

Basic XML Principles

11

- XML is extensible (meaning that the user is able to adjust the structure of the XML application to suit their needs)
- XML can be used to exchange data with other users and programs in a platform independent way
- XML is self-describing and can be used to express meaningful semantics of documents (it provides context for data)



How XML is different from HTML

12

- XML and HTML were designed with different goals:
  - XML was designed to transport and store data, with focus on what data is
  - HTML was designed to display data, with focus on how data looks
- XML is not a replacement for HTML
  - It is a complement to HTML

Writing XML - Declaration

13

- Documents begin with XML declaration

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

- It must be situated at the **first** position of the **first line** in the **XML** document.
- Tells the computer that the file should be read as XML.
- Also defines the encoding that the document is written in.

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

<departures airport="Newcastle">
    <flight id="001">
            <destination code="LAX">Los Angeles</
            <departureTime>22:30</departureTime>
            <aircraft seats="450">Boeing 747</aircr
    </flight>

    <flight id="002">
            <destination code="SYD">Sydney</dest
            <departureTime>23:05</departureTime>
            <aircraft seats="800">Airbus A380</airc
    </flight>
```

Writing XML - Tags

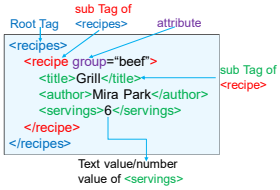
14

- The rest of the document is written using *tags*
  - An *opening and closing tag* (or a content tag)
    `<tag>` **contents** `</tag>`
  - A *single tag*
    `<tag />` (= `<tag>` … `</tag>` : with no content)
  - Nesting tags

```
<outerTag>
  <innerTag>Now for some data</innerTag>
</outerTag>
```

XML Elements

- An XML element is everything from (including) the element's start tag to (including) the element's end tag.
- An element can contain:
  - Root Tag
  - Sub Teg
  - Attributes
  - Element contents (=Data)



Writing XML - Tags

16

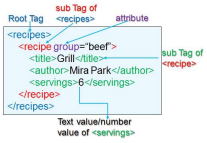
- Tags let us assign a context to each piece of data
  - Example: Write the time 10:30pm in XML…
    - `<time>10:30pm</time>`
    - `<pm>10:30</pm>`
    - `<clock>`
      `<time>10:30</time>`
      `<amPm>pm</amPm>`
      `</clock>`
- Remember, XML is e*X*tensible – all these answers are correct. The answer you choose is **up to you.**
- Can you think of some other answers?

Writing XML - Attributes

17

- All tags can optionally contain attributes – small single-word pieces of information that describe a property of the tag
- Tags can have as many *attributes* as you like. All attributes must have an associated ***value***, which is given inside quotes

```
<tag attribute="value"> … </tag>

<tag attribute="value" />
```



Writing XML - Attributes

18

- You can have as many attributes as you like

  Each attribute *must* have a value

  Examples:

```
<clock time="10:30pm" />
<clock amPm="pm">10:30</clock>
<clock amPm="pm">
      <time hours="10" minutes="30" />
</clock>
```

- Once again, the attributes you use are totally **up to you**

A sample XML document

19

- Consider the following scenario:
  - I want to know…
    - All flights leaving Newcastle airport
    - What time they leave
    - Their destination
    - The type of aircraft
    - How many seats are on the aircraft
  - Here is an example XML document for this data…

airline.xml

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

<departures airport="Newcastle">
    <flight id="001">
            <destination code="LAX">Los Angeles</destination>
            <departureTime>22:30</departureTime>
            <aircraft seats="450">Boeing 747</aircraft>
    </flight>

    <flight id="002">
            <destination code="SYD">Sydney</destination>
            <departureTime>23:05</departureTime>
            <aircraft seats="800">Airbus A380</aircraft>
    </flight>
</departures>
```

- Is there another way you can represent the same data? Yes…

airline.xml

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

<flights airport="Newcaslte">
  <departure id="001" time="22:30">
      <destination code="LAX">
            <name>Los Angeles</name>
      </destination>
      <aircraft seats="450">
            <type>Boeing 747</type>
      </aircraft>
  </departure>

  <departure id="002" time="23:05">
      <destination code="SYD">
            <name>Sydney</name>
      </destination>
      <aircraft seats="800">
            <type>Airbus A380</type>
      </aircraft>
  </departure>
</flights>
```

20

airline.xml

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

<departures airport="Newcastle">
    <flight id="001">
            <destination code="LAX">Los Angeles</destination>
            <departureTime>22:30</departureTime>
            <aircraft seats="450">Boeing 747</aircraft>
    </flight>

    <flight id="002">
            <destination code="SYD">Sydney</destination>
            <departureTime>23:05</departureTime>
            <aircraft seats="800">Airbus A380</aircraft>
    </flight>
</departures>
```


The Rules and Principles

21

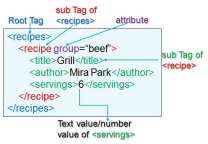
- Tag names can be anything you like, provided they follow the following rules…
  - Start with a **letter** or the "_" character
  - Contains only `letters`, `digits`, "_", "–" or "."
- Tag names are case sensitive !!!
  - `<name>` and `<Name>` are NOT the same
  - `<Message>` This is incorrect `</message>`
  - `<message>` This is correct `</message>`

The Rules and Principles

22

- **Root tag**
  - An XML document must have a single container tag that has all other tags and data inside it (except for the XML declaration). This is called the root tag.

```
<?xml version="1.0" encoding="utf-8" ?>
<rootTag>
    <!-- all other tags go here -->
</rootTag>
```



The Rules and Principles

23

- Tags must be correctly nested
  - `<tag>` must have matching `</tag>`
  - Tags must be opened and closed in the correct order (a FILO – First In Last Out - stack)
  - Examples:
    - Correct ☺

```
<foo>
  <bar>…
  </bar>
</foo>
```

    - Incorrect ☹

```
<foo>
  <bar>
</foo>
  </bar>
```

The Rules and Principles

24

- All XML elements must have a closing tag

```
<sample>
    …
</sample>
```

- **Empty** elements must be written as
  - `<sample></sample>` OR
  - `<sample />`

Note: XML declaration did not have a closing tag. This is not an error. The declaration is not a part of the XML document itself, and it has no closing tag.

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

The Rules and Principles

25

- XML applications can contain comments

  `<!-- ... -->`

  - Are ignored by the application software
  - Possibly thrown away by the XML parser
  - Examples:

```
<!-- This is my first XML, author: Mira Park -->
<?xml version="1.0" encoding="utf-8"?>

<departures airport="Newcastle">
    <!- first flight ->
    <flight id="001">
            <destination code="LAX">Los Angeles</destination>
            <departureTime>22:30</departureTime>
            <aircraft seats="450">Boeing 747</aircraft>
    </flight>
    <!- second flight ->
    <flight id="002">
            <destination code="SYD">Sydney</destination>
            <departureTime>23:05</departureTime>
            <aircraft seats="800">Airbus A380</aircraft>
    </flight>
</departures>
```

```
<!-- remember to fix up this section of XML -->

<!-- what does foobar really mean? -->
```

It must be situated at the **first** position of the **first line** in the **XML** document.

The Rules and Principles

26

- XML applications can contain comments

  `<!-- ... -->`

  - Are ignored by the application software
  - Possibly thrown away by the XML parser
  - Examples:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- This is my first XML, author: Mira Park -->

<departures airport="Newcastle">
    <!- first flight ->
    <flight id="001">
            <destination code="LAX">Los Angeles</destination>
            <departureTime>22:30</departureTime>
            <aircraft seats="450">Boeing 747</aircraft>
    </flight>
    <!- second flight ->
    <flight id="002">
            <destination code="SYD">Sydney</destination>
            <departureTime>23:05</departureTime>
            <aircraft seats="800">Airbus A380</aircraft>
    </flight>
</departures>
```

```
<!-- remember to fix up this section of XML -->

<!-- what does foobar really mean? -->
```

It must be situated at the **first** position of the **first line** in the **XML** document.

XML Parser

27

- In computing, **a parser** is a program (or a piece of code) which analyses files to identify the component parts.
  - Parsed data
  - Unparsed data

Parsing an XML Document

28

- Most of browsers include a built-in XML parser.
- Don't expect XML files to be displayed as HTML pages.
- If there are no syntax errors, the XML document will be displayed. A plus (+) or minus sign (-) to the left of the elements can be clicked to expand or collapse the element structure.

```
<!-- Edited by XMLSpy® -->
- <note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
```

```
<!-- Edited by XMLSpy® -->
+ <note></note>
```

Parsed Character Data

29

- **Parsed Character DATA**, or **PCDATA** consists of all those characters that XML treats as parts of the code of XML document
  - The XML declaration
  - The opening and closing tags of an element
  - Empty element tags
  - Character or entity references
  - Comments
  - Processing instructions

Encoding – Unicode

30

- All XML parsers are required to understand the Unicode encodings ***UTF-8***
- Unicode aims to cover all characters in all past or present written languages
- Code points 0-127 coincide with ACSII characters
- The syntax ***&#N;*** denotes Unicode code point ***N***

  http://www.unicode.org/

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	⌂

31

- ASCII stands for the "American Standard Code for Information Interchange".
- ASCII is a 7-bit character set containing 128 characters.

Encoding – Unicode

32

Symbol	Character Code	Entity Name	Description
>	>	>	Greater than
<	<	<	Less than
'	&#27;	'	Apostrophe (single quote)
"	&#22;	"	Double quote
&	&	&	Ampersand
©	©		Copyright
®	®		Registered trademark
™	™		Trademark
°	°		Degree
½	½		One-half
¼	¼		One-fourth
£	£		Pound sign
€	€		Euro sign
¥	¥		Yen sign

UTF-8

33

- One bytes : the first 128 characters (US-ASCII) need one byte.
- Two bytes: the next 1,920 characters need two bytes to encode. This includes Latin letters with diacritics and characters from the Greek, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac and Tāna alphabets.
- Three bytes are needed for the rest of the Basic Multilingual Plane (which contains virtually all characters in common use).
- Four bytes are needed for characters in the other planes of Unicode, which include less common CJK characters and various historic scripts.

Encoding – Unicode

34

```
<ingredients>
   <ingredient>&#189; cup soy sauce</ingredient>
   <ingredient>&#189; cup plum sauce</ingredient>
   <ingredient>&#189; cup pineapple juice</ingredient>
   <ingredient>&#189; cup ketchup</ingredient>
   <ingredient>&#188; cup scallions, finely sliced</ingredient>
   <ingredient>3 tbsps cilantro, chopped</ingredient>
   <ingredient>2 tbsps fresh ginger, minced</ingredient>
   <ingredient>2 tbsps fresh garlic, minced</ingredient>
   <ingredient>2 flank steaks, about 1&#188; lbs each</ingredient>
</ingredients>

<directions>
   <step>Combine the marinade ingredients in a medium bowl.</step>
   <step>Trim the flank steaks of surface fat.</step>
   <step>Place the steaks and marinade in a sealed plastic bag and
         refrigerate for 3 to 6 hours.</step>
   <step>Remove the steaks from the bag and boil the marinade in
         a small saucepan for 1 full minute.</step>
   <step>Let steaks stand at room temperature for 20 to
         30 minutes.</step>
   <step>Grill steaks over direct high heat (500&#176;) for 8 to
         10 minutes, turning once halfway through.</step>
   <step>Remove steaks from the grill and let rest for 3 to 5
         minutes.</step>
   <step>Cut steaks across the grain into thin strips and serve with
         the marinade sauce.</step>
</directions>
```

Parsing an XML Document

35

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <!--
     Recipes from stanleyblinck.com

     Filename: recipes.dtd
     Author:   Dana Anderson
     Date:     3/1/2012

  -->
- <recipes>
  - <recipe group="beef" rationality="asian">
      <title>Grilled Flank Steak with Asian Sauce</title>
    - <description>
      - <![CDATA[
          A flavorful marinade adds a touch of zest to this classic
          steak recipe. The dish is best served with green beans
          a side. Be sure to remember tenderizing the marinade spices with
          a dash also of a fine dill.
        ]]>
      </description>
      <author>Tonya Evans</author>
      <servings>4</servings>
    - <ingredients>
        <ingredient>½ cup soy sauce</ingredient>
        <ingredient>½ cup plum sauce</ingredient>
        <ingredient>½ cup pineapple juice</ingredient>
        <ingredient>½ cup ketchup</ingredient>
        <ingredient>½ cup scallions, finely sliced</ingredient>
        <ingredient>3 tbsps cilantro, chopped</ingredient>
        <ingredient>2 tbsps fresh ginger, minced</ingredient>
        <ingredient>2 tbsps fresh garlic, minced</ingredient>
        <ingredient>2 flank steaks, about 1½ lbs each</ingredient>
      </ingredients>
    - <directions>
        <step>Combine the marinade ingredients in a medium bowl.</step>
        <step>Trim the flank steaks of surface fat.</step>
        <step>Place the steaks and marinade in a sealed plastic bag and refrigerate for 3 to 6 hours.</step>
        <step>Remove the steaks from the bag and boil the marinade in a small saucepan for 1 full minute.</step>
        <step>Let steaks stand at room temperature for 20 to 30 minutes.</step>
        <step>Grill steaks over direct high heat (500°) for 8 to 10 minutes, turning once halfway through.</step>
        <step>Remove steaks from the grill and let rest for 3 to 5 minutes.</step>
        <step>Cut steaks across the grain into thin strips and serve with the marinade sauce.</step>
      </directions>
    </recipe>
  </recipes>
```

the CDATA displayed as it is entered into the XML file

the ½ character reference appears as ½

the ¼ character reference appears as ¼

the ° character reference appears as °

Unparsed data

36

- Imagine an XML page with large chunk of C code.
- XML parser will try to parse everything including characters like *&*, *<* and will syntax errors
- CDATA - (Unparsed) Character Data
- The term CDATA is used about text data that should not be parsed by the **XML parser**
- Everything inside a CDATA section is ignored by the parser

Unparsed Data

37

- You can enter raw character data using
  `<![CDATA[raw text]]>`
  - The *raw text* is parsed through XML parser without processing
  - Example:
    `<![CDATA[foo<bar & bar>foobar]]>`

In parsed data, equivalent to (foo<bar & bar>foobar) as Unicode entities.

CDATA Example

38

```
<recipes>
  <recipe group="beef" nationality="asian">
    <title>Grilled Flank Steak with Asian Sauce</title>
    <description>
      <![CDATA[
        A flavorful marinade adds a touch of zest to this classic
        steak recipe. The dish is best served with green beans
        & rice. We recommend complementing the marinade spices with
        a pale ale or a Pinot Noir.
      ]]>
    </description>
    <author>Tanya Evans</author>
    <servings>6</servings>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <!--
    Recipes from thekeyphrase.com

    Filename: recipes.xml
    Author:   Tess Anderson
    Date:     3/2/2012

  -->
- <recipes>
  - <recipe group="beef" nationality="asian">
      <title>Grilled Flank Steak with Asian Sauce</title>
    - <description>
      - <![CDATA[
          A flavorful marinade adds a touch of zest to this classic
          steak recipe. The dish is best served with green beans
          & rice. We recommend complementing the marinade spices with
          a pale ale or a Pinot Noir.

        ]]>
      </description>
      <author>Tanya Evans</author>
      <servings>6</servings>
    - <ingredients>
      - <ingredient>½ cup soy sauce</ingredient>
        <ingredient>¼ cup plum sauce</ingredient>
        <ingredient>¼ cup pineapple juice</ingredient>
        <ingredient>¼ cup ketchup</ingredient>
        <ingredient>¼ cup scallions, finely sliced</ingredient>
        <ingredient>2 tbsp cilantro, chopped</ingredient>
        <ingredient>2 tbsp fresh ginger, minced</ingredient>
        <ingredient>4 cloves garlic, minced</ingredient>
        <ingredient>2 flank steaks, about 1½ lbs each</ingredient>
      </ingredients>
    - <directions>
      - <step>Combine the marinade ingredients in a medium bowl.</step>
        <step>Trim the flank steaks of surface fat.</step>
        <step>Place the steaks and marinade in a sealed plastic bag and refrigerate for 3 to 6 hours.</step>
        <step>Remove the steaks from the bag and boil the marinade in a small saucepan for 1 full minute.</step>
        <step>Grill steaks stand at room temperature for 10 to 30 minutes.</step>
        <step>Grill steaks over direct high heat (500°F) for 5 to 10 minutes, turning once halfway through.</step>
        <step>Remove steaks from the grill and let stand for 5 minutes.</step>
        <step>Cut steaks across the grain into thin strips and serve with the marinade sauce.</step>
      </directions>
    </recipe>
  </recipes>
```

the CDATA displayed as it is entered into the XML file

the ½ character reference appears as ½

the ¼ character reference appears as ¼

the ° character reference appears as °

XML Trees

39

- Conceptually, an XML document is a hierarchical structure called an XML tree, consisting of nodes of various kinds arranged as a tree
- XML Trees - Another way to represent XML that may make it easier to understand
- Lets us visualise the structure

XML Trees

40

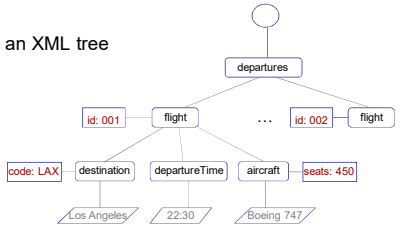
Remember:

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

<departures>
  <flight id="001">
      <destination code="LAX">Los Angeles</destination>
      <departureTime>22:30</departureTime>
      <aircraft seats="450">Boeing 747</aircraft>
  </flight>

  <flight id="002">
      <destination code="SYD">Sydney</destination>
      <departureTime>23:05</departureTime>
      <aircraft seats="800">Airbus A380</aircraft>
  </flight>
</departures>
```

XML Trees

41

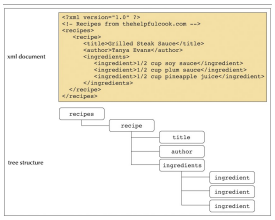
- As an XML tree



XML Trees –The Element Hierarchy

42

- Recall: All elements must be nested within a single **document** or **root tag**. There can be only **one** root tag.



How to define our own tags?

43

- DTD = Document Type Definition
  - Defines the structure (pattern) of a valid XML document and with a list of legal elements and attributes.
  - The purpose of a DTD is to define the legal building blocks of an XML document.
- XML Schema
  - XML Schema is an XML-based alternative to DTDs - (Not covered in this course)

Document Type Definition (DTD)

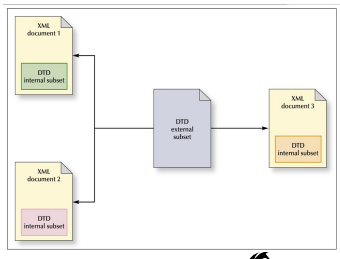
44

- A DTD describes
  - Which tags are valid in an XML Document
  - Which tags can be nested inside which tags
  - Which attributes are valid for which tags

DTD

45

- It can be divided into two parts: an internal subset and an external subset.
  - An **internal subset** is declarations placed in the same file as the document content.
  - An **external subset** is located in a separate file.



Linking XML and DTD

46

- Internal DTD

```
<!DOCTYPE root_tag [
   DTD_rules
]>
```

- External DTD

```
<!DOCTYPE root_tag SYSTEM "file">
```

DTD - Elements

47

```
<!ELEMENT tag (children)>
```

- *children* is a list of tags, commas (,) separate "sequences"

```
<!ELEMENT family (mother,father,child)>
<!ELEMENT body (shape)>
```

- No children? A special child #PCDATA for XML character data

```
<!ELEMENT departureTime (#PCDATA)>
```

- Vertical bars (|) separate "choices"

```
<!ELEMENT foo (bar|foobar|baz)>
```

DTD – Elements

48

- Each child has an optional modifier
  - *child* = one and only one such child allowed
  - *child*+ = one or more such children allowed
  - *child** = zero or more such children allowed
  - *child*? = zero or one such children allowed
- Mixed content
  - ((#PCDATA| *child2* | … | *childn*)*)

DTD - Elements

49

- Examples:

```
<!ELEMENT body (shape)>

<!ELEMENT departureTime (#PCDATA)>

<!ELEMENT family (mother?,father?,child*)>
<!ELEMENT mother (firstName,surname,DOB)>
…
<!ELEMENT foo (bar+|foobar*|baz?)>
<!ELEMENT anything (#PCDATA|name|note)*>
```

DTD - Attributes

50

```
<!ATTLIST tag attribute type default>
```

- example

```
<!ELEMENT circle EMPTY>
      <!ATTLIST circle radius CDATA "0">
```

DTD - Attributes

51

- **Attributes**

```
<!ATTLIST tag attribute type default>
```

  - *type* = **CDATA** – character data
  - *type* = (*en1* | *en2* | …) – one of the listed values

```
<?xml version="1.0"?>
  <!DOCTYPE image [
        <!ELEMENT image EMPTY>
                <!ATTLIST image height CDATA #REQUIRED>
                <!ATTLIST image width CDATA #REQUIRED>
]>

<image height="32" width="32"/>
```

DTD - Attributes

52

- **Attributes**

  `<!ATTLIST tag attribute type default>`
  - *type* = CDATA – character data
  - *type* = (*en1* | *en2* | …) – one of the listed values

```
<!ELEMENT payment (#PCDATA)>
<!ATTLIST payment method (cheque|cash) "cash">
```

Valid XML:

```
<payment method="cheque" />
```
or
```
<payment method="cash" />
```

Invalid XML:

```
<payment method="credit" />
```

DTD - Attributes

53

- **Attributes**

  `<!ATTLIST tag attribute type default>`
  - *default* = *value*
  - *default* = #FIXED *value* – value cannot be changed by the tag
  - *default* = #REQUIRED – must be supplied by the tag
  - *default* = #IMPLIED – no default, but doesn't have to be supplied by the tag

DTD - Attributes

54

`<!ATTLIST tag attribute type default>`

- *default* = *value*
  - DTD:

```
<!ELEMENT circle EMPTY>
<!ATTLIST circle radius CDATA "0">
```

  - Valid XML:

```
<circle radius="100" />
```

  - Valid XML:

```
<circle />
```

If no *radius* is specified, it has a default value of 0.

DTD - Attributes

55

<!ATTLIST tag attribute type **default**>:

- *default* = **#FIXED** *value*
  - DTD:
    ```
    <!ATTLIST sender company CDATA #FIXED "Microsoft">
    ```
  - Valid XML:
    ```
    <sender company="Microsoft" />
    ```
  - Invalid XML:
    ```
    <sender company="W3Schools" />
    ```
  - Use the #FIXED keyword when you want an attribute to have a fixed value. If another value is given, the parser will return an error.

DTD - Attributes

56

<!ATTLIST tag attribute type **default**>:

- *default* = **#REQUIRED**
  - DTD:
    ```
    <!ATTLIST person number CDATA #REQUIRED>
    ```
  - Valid XML:
    ```
    <person number="5677" />
    ```
  - Invalid XML:
    ```
    <person />
    ```
  - It forces that the attribute must always be included

DTD - Attributes

57

<!ATTLIST tag attribute type **default**>:

- *default* = **#IMPLIED**
  - DTD:
    ```
    <!ATTLIST contact fax CDATA #IMPLIED>
    ```
  - Valid XML:
    ```
    <contact fax="555-667788" />
    ```
  - Valid XML:
    ```
    <contact />
    ```
  - Use the #IMPLIED keyword if you don't want to force the inclusion of an attribute

Example

58

```
<!DOCTYPE customers
[
   <!ELEMENT customers (customer+)>

   <!ELEMENT customer (name, address, phone, email?, orders)>
   <!ATTLIST customer custID ID #REQUIRED>
   <!ATTLIST customer custType (school | home | business) #IMPLIED>

   <!ELEMENT name (#PCDATA)>
   <!ATTLIST name title (Mr. | Mrs. | Ms.) #IMPLIED>

   <!ELEMENT address (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
   <!ELEMENT email (#PCDATA)>
   <!ELEMENT orders (order+)>

   <!ELEMENT order (orderDate, items)>
   <!ATTLIST order orderID ID #REQUIRED>
   <!ATTLIST order orderby IDREF #REQUIRED>

   <!ELEMENT orderDate (#PCDATA)>
   <!ELEMENT items (item)>

   <!ELEMENT item (#PCDATA)>
   <!ATTLIST item itemPrice CDATA #REQUIRED>
   <!ATTLIST item itemQty CDATA "1">
]>
```

#IMPLIED indicates that the attribute is optional and may be omitted from the element

#REQUIRED indicates that the attribute must always be included with the element

DTD - Entities

59

- **Entities**
  - Entities are variables used to define **shortcuts** to standard text or special characters.
  - Entity references are references to entities
- As well as XML predefined entities such as: `>`, `<`, `&`, `"`, `'`
  Other Entities can be defined in the DTD

Creating Parsed Entities

60

`<!ENTITY name "value">`

- Defines a new character entity `&name;`

- For example, an entity named "MBL25" can be created to store a product description:

```
<!ENTITY MBL25 "Monarch Butterfly, 6-12 larvae">
```

- After an entity is declared, it can be referenced anywhere within the document.

```
<item>&MBL25;</item>
```

- This is interpreted as

```
<item> Monarch Butterfly, 6-12 larvae </item>
```

should not be there

Example – XML + DTD

61

```
<?xml version="1.0" ?>
<!DOCTYPE greeting [
   <!ELEMENT greeting (#PCDATA)>
   <!ATTLIST greeting style (big|small) "small">
   <!ENTITY hi "Hello" >
]>
```

Example – XML + DTD

62

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE note [
      <!ELEMENT note (to, from, heading, body)>
      <!ELEMENT to (#PCDATA)>
      <!ELEMENT from (#PCDATA)>
      <!ELEMENT heading (#PCDATA)>
      <!ELEMENT body (#PCDATA)>
]>
```

order

Example – XML + DTD

63

Tutorials.dtd

```
<!ELEMENT tutorials (tutorial)+>
<!ELEMENT tutorial (name,url)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ATTLIST tutorials type CDATA #REQUIRED>
```

XML Validation against DTD

64

- A document is ***well-formed*** XML if-and-only-if:
  - Its syntax conforms to the XML specification
  - Its tags form a hierarchical tree, with a single root node
- A document is ***valid*** XML if-and-only-if:
  - It is well-formed, and …
  - It conforms to an associated DTD
  - `http://validator.w3.org`

validation Important

Summary

65

- The rules and principles of XML
- XML Parsing
  - CDATA and PCDATA
- XML Trees – the hierarchical structure of the XML document
- How to define your own XML tags
- Internal and External DTD
- How to declare DTD elements
- How to set the attributes for DTD elements
- What is a DTD entity

References

66

- **Programming the World Wide Web**
  By Robert W. Sebesta
  - *Chapter 7*
- http://www.w3schools.com/xml/default.asp
- http://www.w3resource.com/xml/xml.php

Credits

- http://www.w3.org/
- http://www.w3.org/XML/
- http://www.w3.org/TR/xmlschema-0/
- http://www.w3.org/TR/xslt
- http://www.w3.org/TR/xpath
- http://www.w3schools.com/xml/default.asp