ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Εισαγωγή στην Επιστήμη και Τεχνολογία των Υπηρεσιών

Ενότητα 2: Writing XML

Χρήστος Νικολάου
Τμήμα Επιστήμης Υπολογιστών

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Άδειες Χρήσης

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ
2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

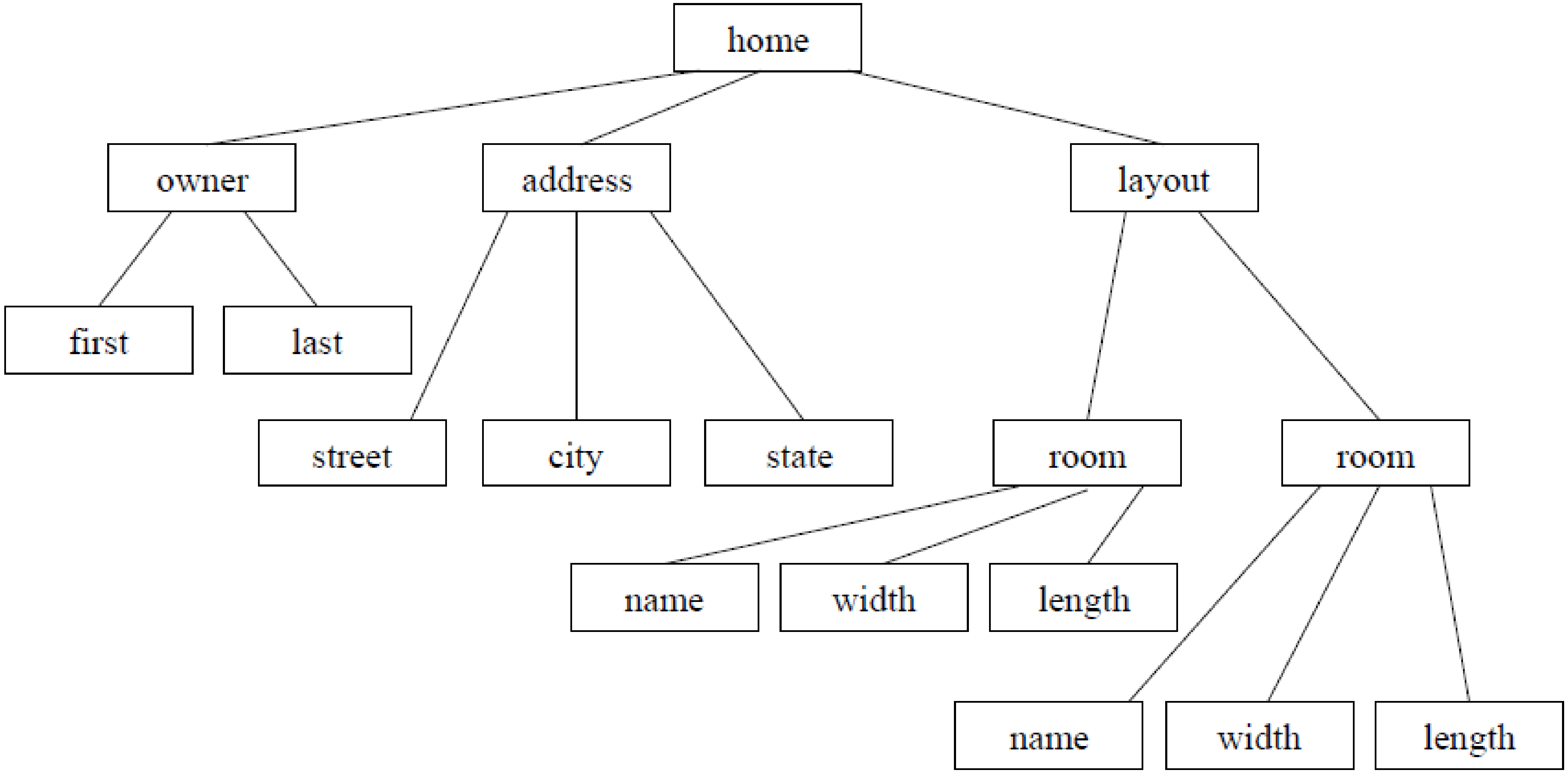
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

XML
Writing XML
605.44 / 635.444

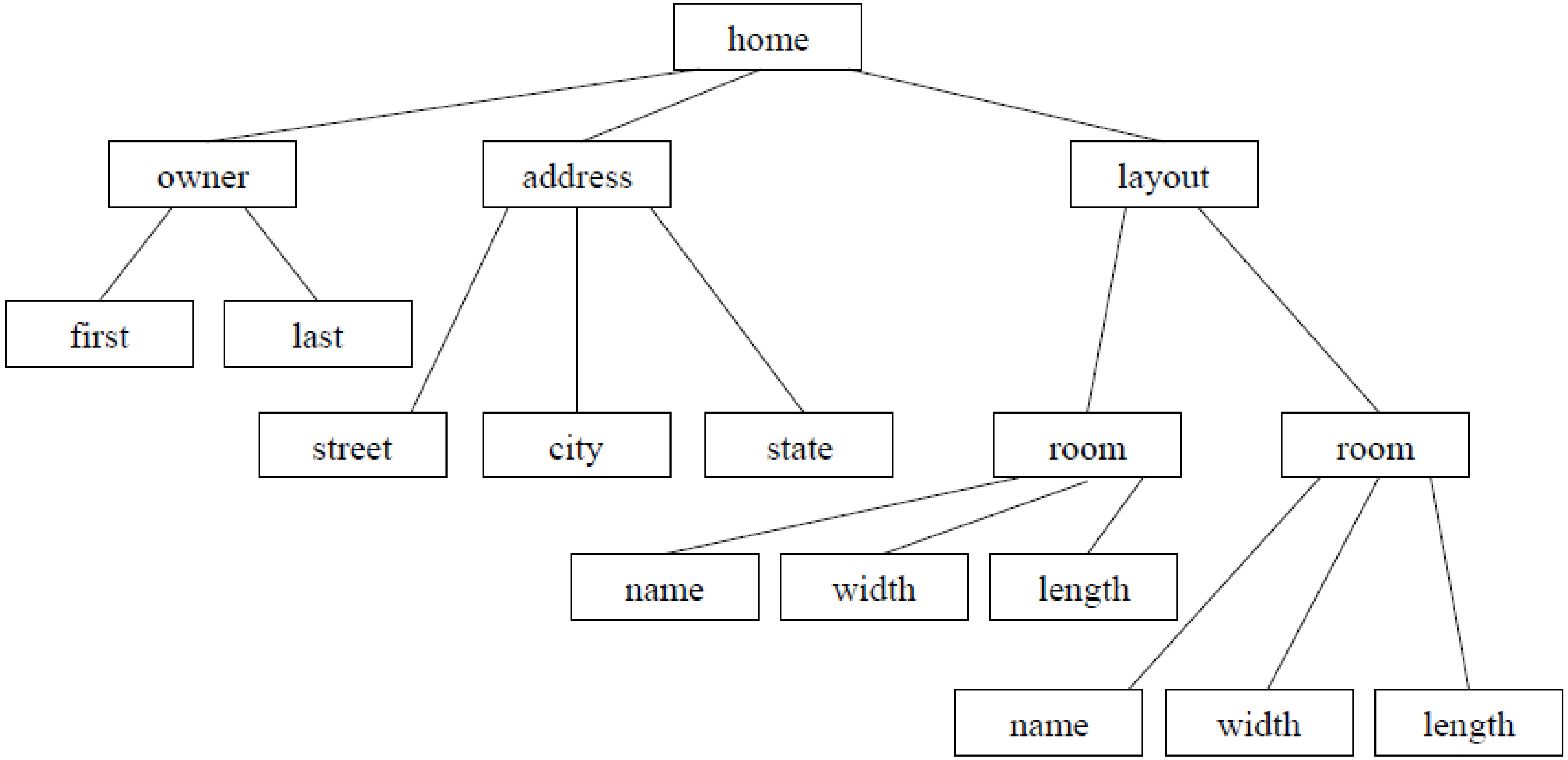
David Silberberg

Lecture 2

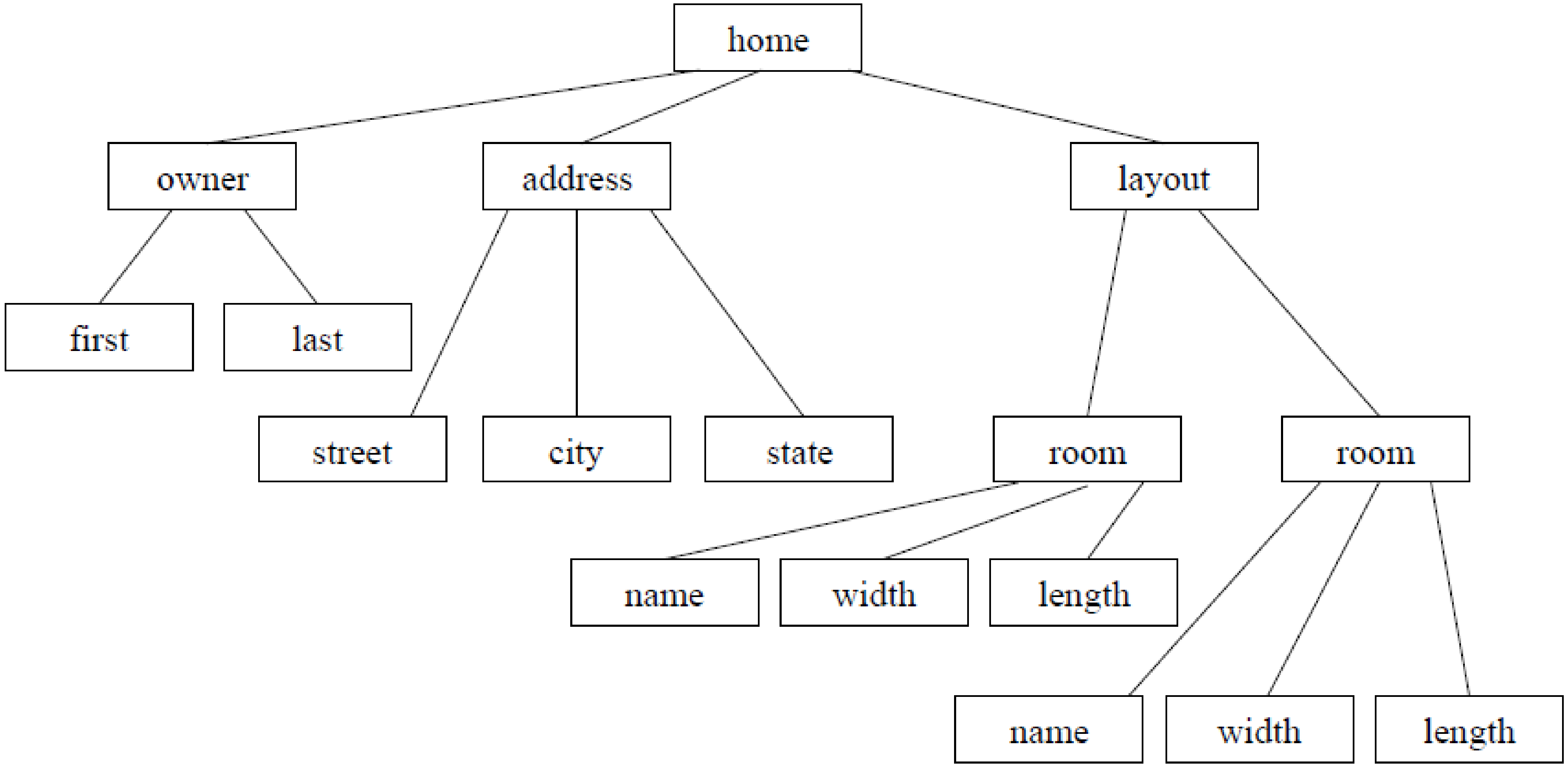
Let's Represent a House



Let’s Represent a House



Let’s Represent a House



XML Declaration (cont.)

- Encoding and Standalone attributes are optional
- First character in the file should be <
    - No breaks allowed
    - No spaces allowed
    - Some parsers are more flexible

XML Declaration - Encoding

- Definition
  - Character code is a one-to-one mapping between characters and their machine representations
  - Character encoding is the method used to represent the characters digitally
- ASCII
  - American Standard Code for Information Interchange
  - Most common English standard
  - Eight bit representations
    - "A" is 65
    - "a" is 97
  - Only represents 256 characters
  - Fine for English
  - Not fine for other character sets

Encoding (cont.)

- Unicode
  - UTF-16 (Unicode Transformation Format-16)
    - 16 bits = 2 bytes for every character
    - 2^{16} different characters can be represented
    - If you are representing ASCII text, you waste much space because every character can be represented by 1 byte
  - UTF-8
    - 8 bits = 1 byte for every character
    - Actually, the high bit encodes whether or not 7-bit ASCII is used
    - If the bit is set one way, it represents 7-bit ASCII characters in one byte
    - If the bit is set the other way, two bytes are used
    - This saves much space

Encoding (cont.)

- XML specification requires Unicode to be used internally
  - This is not usually the case for external files
  - Most are encoded with other standards
    - ISO-8859-1
    - windows-1252
    - EBCDIC
  - These are variants of ASCII, but not subsets of UTF-8 like ASCII
- The encoding attribute specified in the ?xml statement indicates the character encoding of the text to the parser
- Parser reads text and translates it to Unicode internally
- If no encoding is specified, UTF-8 or UTF-16 is assumed

Encoding (cont.)

- An XML document may ignore the encoding if there is a protocol specific encoding
  - For example, HTTP may have a protocol specific encoding
  - HTTP takes precedence over XML
- An XML document created in Notepad on Microsoft Windows
  - Some versions save characters in windows-1252 form by default
    - You need to create the corresponding ?xml statement:
      `<?xml encoding="windows-1252"?>`
  - If the parser does not understand this encoding, then use ISO-8859-1 or ASCII instead
  - In Notepad using Windows 2000 and NT
    - Saves characters in ANSI form by default
    - You can save a file in Unicode format

Standalone

- Standalone Document Declaration (SDD)
  - Either “yes” or “no” if used
  - “yes” indicates that the XML document does not depend on other documents (stands alone)
  - “no” indicates that the XML document depends on other files
  - Not required
- XML parser is not required do anything with the *standalone* attribute
  - Provides a hint to the parser
  - If standalone is "no," it lets the parser know that it should validate the document with an external DTD file
  - Standalone is not used with external XML Schema files

Sample Headers

- <?xml version="1.0"?>
  - Standard, no frills header
  - Unicode is assumed since no encoding is specified
  - It is assumed to be standalone="yes"

- <?xml version="1.0" encoding="ASCII" standalone="no"?>
  - ASCII encoding
  - Uses an external DTD file

- <?xml version="1.0" encoding= "UTF-8" standalone="yes"?>
  - UTF-8 encoding
  - Does not use an external DTD file

Sample Headers

- <?xml version="1.0"?>
  - Standard, no frills header
  - Unicode is assumed since no encoding is specified
  - It is assumed to be standalone="yes"

- <?xml version="1.0" encoding="ASCII" standalone="no"?>
  - ASCII encoding
  - Uses an external DTD file

- <?xml version="1.0" encoding= "UTF-8" standalone="yes"?>
  - UTF-8 encoding
  - Does not use an external DTD file

PIs Continued

- There can be many PIs in one XML file
- Convention
  - PIs that start with xml-[something] refer to XML-related technology
  - Other PIs are not XML-related technology
  - Example
    - Stylesheets are XML technology
    - <?xml-stylesheet href="House.xsl" type="text/xsl"?>
    - We will cover stylesheets later
- <?xml …?> is not really a PI
  - Cannot get the 'xml' declarations from most parsers
  - <?xml …?> can only be placed at the beginning of the document

Document Type Declarations

- Specifies a DTD for an XML document

  <!DOCTYPE home SYSTEM “C:\home.dtd”>
- Must start with <!DOCTYPE
- Must end with >
- The first parameter is the root element
  - This is the outermost tag in the content of an XML document
  - XML permits only one root in a document
  - If your XML document includes another XML document, then the other document’s root element is enclosed in your root element
  - If the root element does not match, the parser returns an error

DTDs (continuted)

- Second element
  - SYSTEM
    - Corresponding DTD is available on your system or another system
    - Need to specify the location of the DTD using a
      - Uniform Resource Locator (URL) or
      - More generally, Uniform Resource Identifier (URI)
        - » URI can be a URL or Universal Resource Name (URN)
        - » We'll cover URNs later when we speak about Namespaces
    - Examples on personal PC system
      - <!DOCTYPE home SYSTEM "C:\home.dtd">
      - <!DOCTYPE home SYSTEM "file:///DTD/home.dtd">
    - Example on other systems
      - <!DOCTYPE home SYSTEM "http://www.apl.jhu.edu/home.dtd">
      - <!DOCTYPE home SYSTEM "urn:HomeStandard:home-design">

DTDs (continuted)

- PUBLIC
  - Publicly available resources
  - Well known standards
  - Not in traditional URI format
  - Allows second URI in case the first is unavailable
  - Examples
    - <!DOCTYPE home PUBLIC
      “-//W3C//DTD XHJTML 1.0 Transitional//EN”
      “http://www.apl.jhu.edu/home.dtd”>
    - <!DOCTYPE home PUBLIC
      “HomeDesign/Home Template/”
      “http://www.apl.jhu.edu/home.dtd”>
  - May go into more detail in future lectures

Comments

- Standard comment syntax
- Starts with <!--
- Ends with -->
- Can span multiple lines
- Anything can go in between (almost anything)
- Cannot have nested comments
  - <!-- comment <!-- subcomment --> -->
- Cannot exist inside a tag
  - <home <!-- comment --> >
- Cannot contain --
  - <!-- comment -- not valid -->

The Document Root Element

- Highest level tag in document
- In well-formed documents, the document root element must be the first opening tag and the last closing tag.

```
<home>
      …
</home>
```

- Otherwise, it is just like any other tag
- There is only one document root element per document
- Allows document inclusion to work seamlessly

XML Data Elements - Names

- Can start with letters or “_” or “:”.
- After the first character, numbers, “-”, and “.” are allowed
- Names are case sensitive
  - <HOME> is different than <home>
- Names cannot:
  - start with numbers or punctuation symbols
  - contain spaces
  - (Should not) contain “:” unless you are using Namespaces
  - start with XML in upper, lower, or mixed case
- Names should be readable and reasonable
  - Not too long, too short, or too cryptic

Names (cont.)

- Names should be meaningful
- Need to have beginning and end
  - End starts with /
  - <length>14</length>
  - <width>12</width>
- If there is no text between begin and end tag
  - <painted></painted> is a bit tedious
  - <painted/> is just as good
  - <owner ssn="S-111-11-1111"/>
- Tags cannot overlap
  - <length>12<width></length>12</width>

Attributes

- What is the difference between elements and attributes?
  - ssn element

```
<owner>
        <ssn>S-111-11-1111</ssn>
</owner>
```

  - ssn attribute

```
<owner ssn="S-111-11-1111"></owner>
<owner ssn="S-111-11-1111"/>
```

- It is often a matter of preference
- There are no differences in usage or meaning. (However, IDs and IDREFs can only be defined in attributes.)

Attributes (continued)

- Usually, element data is information that is of interest to a user or someone looking at the document
- Usually, attribute data is information that is of interest to a program for some reason
  - Can be identification number that is not of concern to average user
  - Can be information used for indexing or searching
- Must have values
  - Invalid: <name paid>…</name>
  - Valid: <name paid="true">…</name>
- Values must be surrounded by "" or ''
  - Single quotes can be contained in double quotes or visa versa

White Space

- HTML
  - Strips white space that is considered insignificant

    ```
    <P>You can write a paragraph.      This one has two
    sentences.</P>
    ```

  - HTML prints it as follows

    ```
    You can write a paragraph. This one has two sentences.
    ```

  - To preserve the space in HTML, special “non-breaking spaces” (&nbsp;) are used

    ```
    <P>You can write a paragraph.&nbsp;&nbsp;&nbsp;&nbsp;
    &nbsp;&nbsp;This one has two sentences.</P>
    ```

  - HTML prints it as follows

    ```
    You can write a paragraph.      This one has two sentences.
    ```

White Space (cont.)

- XML
  - Does not strip white space

    ```
    <P>You can write a paragraph.      This one has two
      sentences.</P>
    ```
  - Data is as follows

    ```
    You can write a paragraph.      This one has two
      sentences.
    ```
  - Carriage returns, line feeds, and new lines
    - All different on different operating systems
    - XML strips them out and replaces with a single line feed character
    - Data exchange is simplified

White Space (cont.)

- XML
  - White space between tags

```
<owner ssn="S-111-11-1111">
      <first>David</first>
</owner>
```

  - Should there be a line feed after <owner> and before <first>?
  - If so, this is *extraneous white space*
  - Cannot tell from this document alone
  - DTD or Schema files answer this question

Illegal Characters

- Cannot use “<“ and “&” characters
- Example:
  - <fact>8 < 80 & 80 > 8</fact>
  - Parser expects no space after <
  - Parser expects no space after &
  - Need escaping characters

Escaping Characters

- Escaping characters

&amp;	- the &amp; character
&lt;	- the &lt; character
&gt;	- the &gt; character
&apos;	- the ‘ character
&quot;	- the “ character
&amp;nnn;	- the Unicode character
&amp;#xnnn;	- the hexadecimal number (&amp;169; or &#xA9; is ©)

- Example
  - &lt;fact&gt;8 &lt; 80 &amp; 80 &gt; 8&lt;/fact&gt;

Constants

- Can represent constant values with escaping characters
- Anything after an & and before ; is considered an *entity reference*
- XML handles the escaped characters on the previous page in a special way
- In general, the parser will use DTD and Schema files or other means to deal with constants
- Examples
  - <room>&MyFavoriteRoom;</room>
  - Parser or program will deal with it

Unparsed Data (CDATA)

- CDATA stands for Character Data
- Useful for passing large chunks of literal text without the parser touching it
- Example:

```
<![CDATA[
    <Room 1>Dining Room
    <Room 2>Living Room
]]>
```

Parsers

- Apache Xerces: http://xerces.apache.org/
- DataChannel XJ Parser: http://xdev.datachannel.com/directory/xml_parser.html
- IBM XML4J: http://alphaworks.ibm.com/tech/xml4j
- James Clark's XP:
  - http://www.jclark.com/xml/xp
  - http://www.jclark.com/xml/expat.html
- OpenXML: http://www.openxml.org
- Oracle XML Parser: http://technet.oracle.com/tech/xml
- Sun Microsystems Project X: http://java.sun.com/products/xml
- Tim Bray's Lark and Larval: http://www.textuality.com/Lark
- Vivid Creations ActiveDOM: http://www.vivid-creations.com

XML Parser Errors

- Two types of errors in XML
- Errors
  - Violation of the rules in the specification where results are undefined
  - Parser is allowed to recover from the error and continue
- Fatal Errors
  - XML document is not well formed
  - Parser may only continue to identify more errors
  - Parser will not try to recover

Design Considerations

- Should one model a given data item as a **<u>subelement</u>** or as an **<u>attribute</u>** of an existing element?
- Example, you could model the title of a slide either as:

```
<slide>
  <title>This is the title</title>
</slide>
```

- or as:

```
<slide title="This is the title">
...
</slide>
```

Forced Choices

- Sometimes, the choice between an attribute and an element is forced on you by the nature of attributes and elements
- The data contains substructures
  - Must be modeled as an *element*
  - Attributes take only simple strings.
  - So if the title can contain emphasized text like this:

    `The <em>Best</em> Choice, then the title must be an element.`
- The data contains multiple lines
  - Here, it also makes sense to use an element
  - Attributes need to be simple, short strings or else they become unreadable, if not unusable.

Forced Choices (2)

- The data changes frequently
  - When the data will be frequently modified, especially by the end user, then it makes sense to model it as an element
  - XML-aware editors tend to make it easy to find and modify element data
  - Attributes can be somewhat harder to get to, and therefore somewhat more difficult to modify.
- The data is a small, simple string that rarely if ever changes
  - This is data that can be modeled as an attribute
  - However, just because you can does not mean that you should

Forced Choices (3)

- The data is confined to a small number of fixed choices
  - Here is one time when it makes sense to use an attribute
  - Using DTD or Schema specifications, attributes can be prevented from taking on values that are not in the pre-approved lists
  - An XML-aware editor can even provide those choices in a drop-down list
  - Note: the gain in validity restriction comes at a cost in extensibility
    - Author of the XML document cannot use any value that is not part of the DTD
    - If another value becomes useful in the future, the DTD or Schema will have to be modified before the document author can make use of it

Forced Choices (3)

- The data is confined to a small number of fixed choices
  - Here is one time when it makes sense to use an attribute
  - Using DTD or Schema specifications, attributes can be prevented from taking on values that are not in the pre-approved lists
  - An XML-aware editor can even provide those choices in a drop-down list
  - Note: the gain in validity restriction comes at a cost in extensibility
    - Author of the XML document cannot use any value that is not part of the DTD
    - If another value becomes useful in the future, the DTD or Schema will have to be modified before the document author can make use of it

Stylistic Choices (2)

- Visibility
  - If data is intended to be shown to end users, then it is reasonable to model them as elements
  - If the data guides XML processing but are never displayed, then it may be better to model them as attributes
  - Example
    - *manufacturer name* can be modeled as an element
    - *manufacturer's code number* can be modeled as an attribute

Stylistic Choices (3)

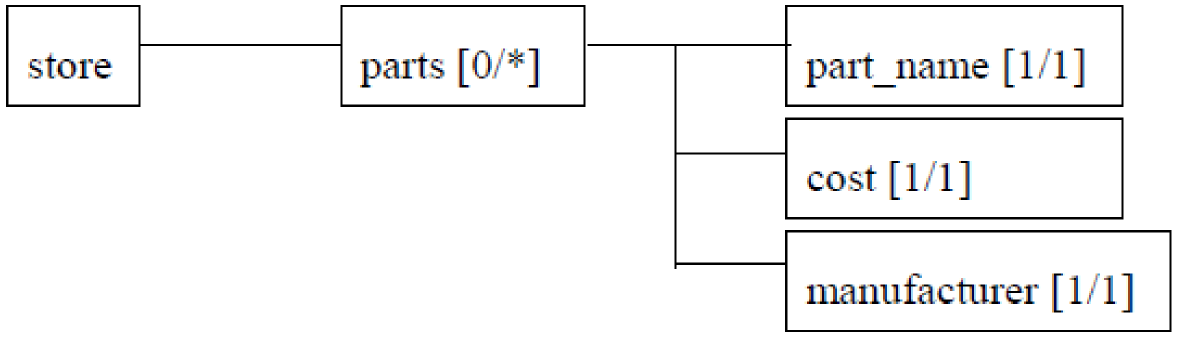
- Consumer / Provider
  - Determine who is the consumer and/or provider of the information
  - Human enters *manufacturer name* so it is modeled as an element
  - Software supplies *manufacturer's code number* so it is modeled as an attribute
- Container vs. Contents
  - Another way of thinking about elements and attributes is to think of an element as a container
  - The contents of the container (water or milk) correspond to XML data modeled as elements
  - On the other hand, the characteristics of the container (blue or white, pitcher or can) correspond to XML data modeled as attributes

Modeling XML

- Pictures are always useful when conveying a design of a data model
- I will present just one of many possible XML data structure models
- This will be modified slightly later in the semester when representing DOM structure
- The main point is to model the hierarchy correctly

Modeling a Simple XML Structure

- Represent each element/attribute using a rectangle
  - Do not be concerned about whether you will ultimately represent data as an element or an attribute
  - The hierarchy is the most important aspect right now
  - Lines between elements/attributes represent direct hierarchical structure
- Example hierarchy:

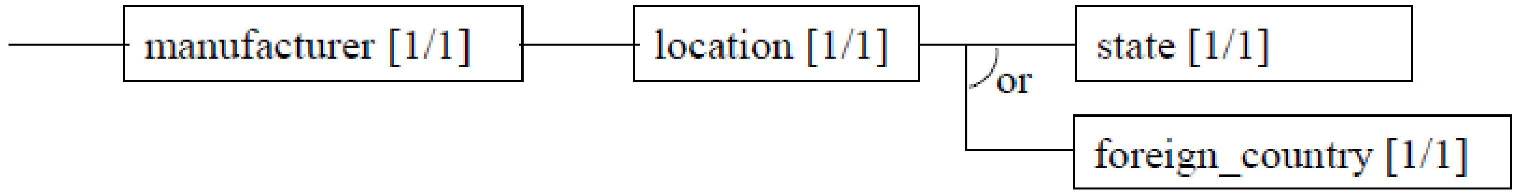


Hierarchy Explanation

- Document root element is *store*
  - Does not need a cardinality
  - There can only be one of these in an XML document
- *store* has one type of child which is *parts*
  - At a minimum, there can be no *parts*
  - At a maximum, there can be an infinite number of *parts*
- *parts* must have 3 children – *part_name, cost, manufacturer*
  - At a minimum, there can be one of each of these children per each *parts*
  - At a maximum, there can be one of each of these children per each *parts*

Modeling Choice

- Suppose the location of a *manufacturer* is either represented as a U.S. state or a foreign country, but not both.
  - An **or** is used to represent this choice
  - The **or** is placed in the hierarchy as:



Modeling Known Values

- In general, the range of data values of elements and attributes are many
  - It does not make sense to list all values
- In some cases, the range of data values are few
  - It is useful to list all the values
  - This is how it can be represented in the model:

```
state [1/1] – range: AK, AL, …
```

```
title [1/1] – range: Mr., Ms., Mrs., Miss, Dr.
```

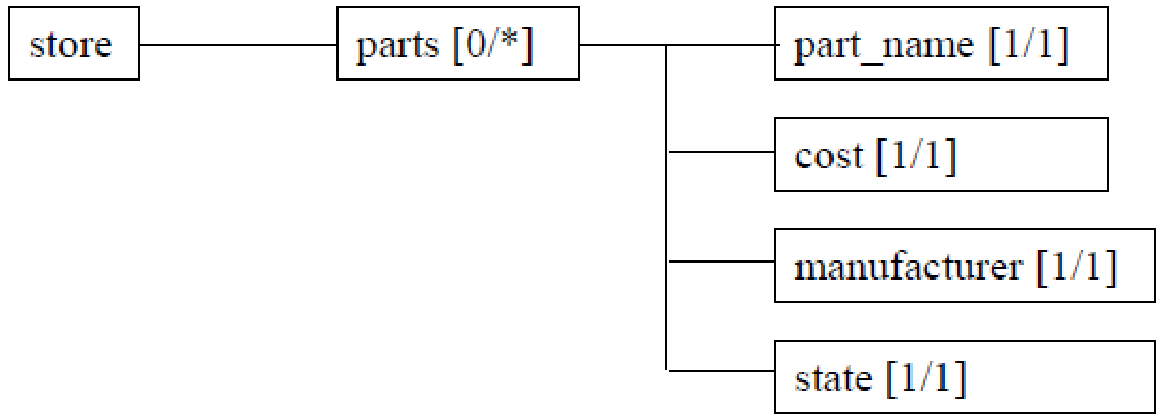
Normalization

- The process of eliminating redundancies is known as normalization
- Defining DTD entities or Schema types is one good mechanism to help guide data normalization
- The considerations for defining an entity reference are similar to those you would apply to modularize database tables or program code

Normalization (2)

- Whenever you find yourself writing the same thing more than once, create one element substructure
  - Use IDREFs to reference the element substructure
  - IDs and IDREFs can only be defined in attributes
  - Lets you write it one place and reference it multiple places.
- If the information is likely to change and is used in more than one place, define it in one place
- Normalization produces modular XML that is smaller as well as easier to update and maintain
- The normalization process can make the resulting document somewhat more difficult to visualize
- However, once you understand it, it makes sense

Problems of Un-normalized Data



part_name	cost	manufacturer	state
widgit	$3	Acme Inc.	MD
thing-a-ma-bob	$5	Acme Inc.	MD
doodad	$4	XYZ Ent.	NJ

Un-normalized XML

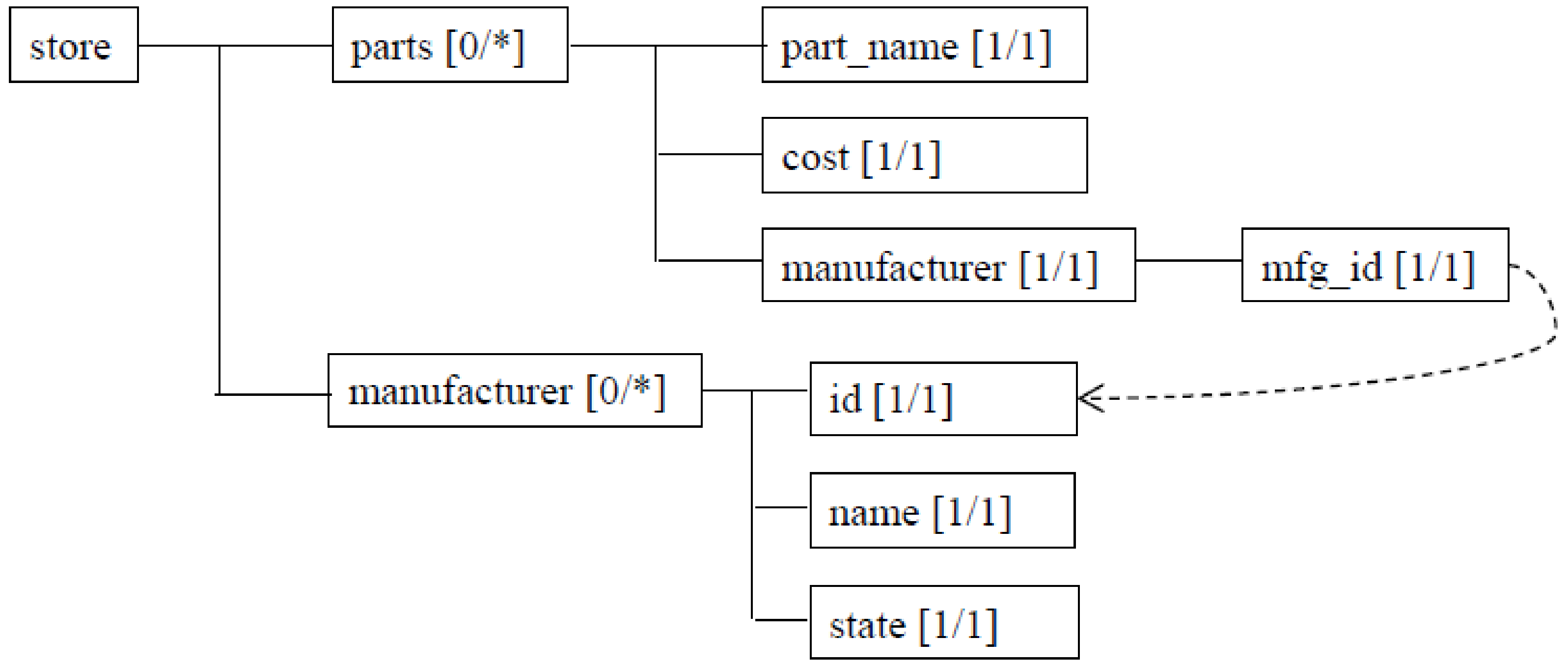
```
<store>
        <parts>
             <part_name> widget </part_name>
             <cost> 3 </cost>
             <manufacturer> Acme Inc. </manufacturer>
             <state> MD </state>
        </parts>
        <parts>
             <part_name> thing-a-ma-bob </part_name>
             <cost> 5 </cost>
             <manufacturer> Acme Inc. </manufacturer>
             <state> MD </state>
        </parts>
        <parts>
             <part_name> doodad </part_name>
             <cost> 4 </cost>
             <manufacturer> XYZ Ent. </manufacturer>
             <state> NJ </state>
        </parts>
</store>
```

Problems of Un-normalized Data (2)

- Repetition anomaly - state repeated
- Update anomaly - state updated twice
- Insertion anomaly
  - Cannot add new manufacturer until we have part
  - Visa versa
- Deletion anomaly - deleting 'doodad' deletes XYZ from DB

Modeling References

- To normalize
  - The data must be aggregated into logical groups
  - The groups must reference through identifier references

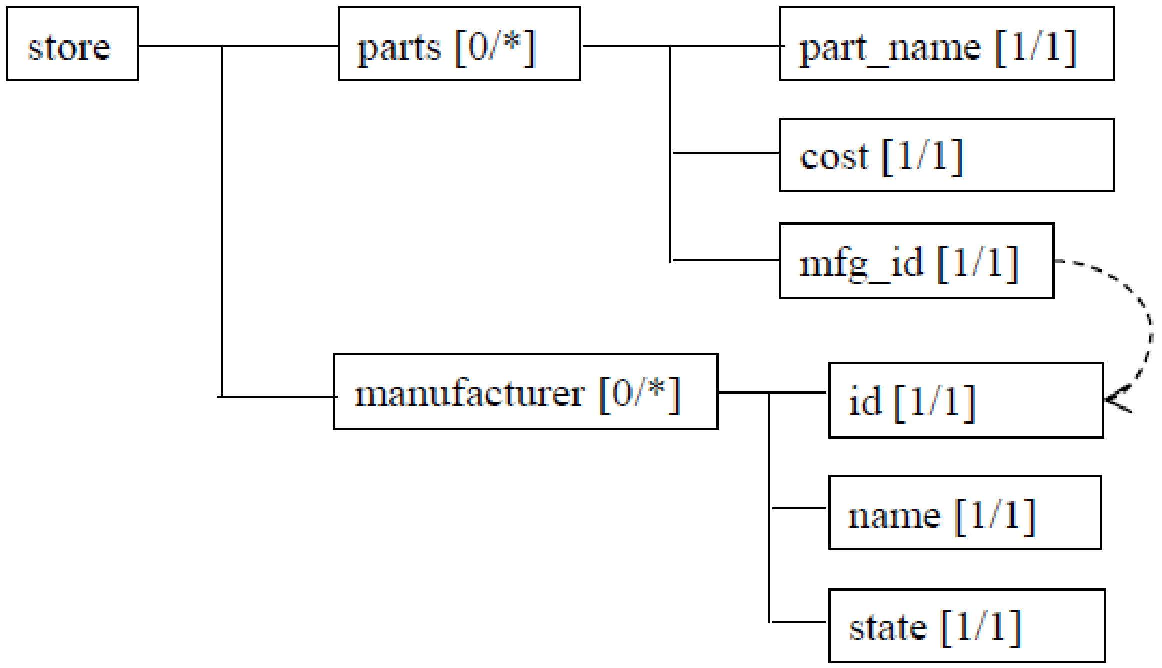


Normalized XML

```
<store>
    <parts>
        <part_name> widget </part_name>
        <cost> 3 </cost>
        <manufacturer mfg_id="m1"/>
    </parts>
    <parts>
        <part_name> thing-a-ma-bob
        </part_name>
        <cost> 5 </cost>
        <manufacturer mfg_id="m1"/>
    </parts>
    <parts>
        <part_name> doodad </part_name>
        <cost> 4 </cost>
        <manufacturer mfg_id="m2"/>
    </parts>
```

```
    <manufacturer id="m1">
        <name>Acme Inc. </name>
        <state> MD </state>
    </manufacturer>
    <manufacturer id="m2">
        <name>XYZ Ent. </name>
        <state> NJ </state>
    </manufacturer>
</store>
```

Another Possible Model



Normalized Version of Other Model

```
<store>
        <parts mfg_id="m1">
             <part_name> widget </part_name>
             <cost> 3 </cost>
        </parts>
        <parts mfg_id="m1">
             <part_name> thing-a-ma-bob
             </part_name>
             <cost> 5 </cost>
        </parts>
        <parts mfg_id="m2">
             <part_name> doodad </part_name>
             <cost> 4 </cost>
        </parts>
        <manufacturer id="m1">
             <name>Acme Inc. </name>
             <state> MD </state>
        </manufacturer>
        <manufacturer id="m2">
             <name>XYZ Ent. </name>
             <state> NJ </state>
        </manufacturer>
</store>
```

Τέλος Ενότητας