ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Εισαγωγή στην Επιστήμη και Τεχνολογία των Υπηρεσιών

Ενότητα 4: Document Type Definitions (DTDs) - 1

Χρήστος Νικολάου
Τμήμα Επιστήμης Υπολογιστών

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται στην άδεια χρήσης **Creative Commons** και ειδικότερα

Αναφορά – Μη εμπορική Χρήση – Όχι Παράγωγο Έργο v. 3.0

(Attribution – Non Commercial – Non-derivatives)

- Εξαιρείται από την ως άνω άδεια υλικό που περιλαμβάνεται στις διαφάνειες του μαθήματος, και υπόκειται σε άλλου τύπου άδεια χρήσης. Η άδεια χρήσης στην οποία υπόκειται το υλικό αυτό αναφέρεται ρητώς.

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
Document Type Definitions (DTDs)
605.444 / 635.444

David Silberberg

Lecture 3

What are Document Type Definitions (DTDs)?

- A mechanism for describing the rules of forming a set of documents
  - Description of the syntactic (and some semantic) rules of an XML document
  - Akin to structure definitions in programming languages
  - Description of the configuration of elements and attributes for a document set
- Enable validating XML documents for syntactic correctness
  - Useful for document exchange
  - Useful for business to business applications
  - Useful for ensuring correctness of single document
- Provides a mechanism for documenting XML files

Why Create DTDs?

- Portability
  - All applications know the correct structure of documents
  - Reduce the need for “paper” documentation
  - Can be parsed by a program
  - Structure definition independent of platform
- Reduce code size
  - Constraints of data not coded in application
  - Can rely on tools to determine validity of document structure
- Reduce maintenance
  - Changes in structure require only corresponding changes in a single DTD
  - No need to update validation software of multiple applications

Why DTDs? (cont.)

- What is wrong with just ensuring that a document is well formed?
  - Well-formed documents are syntactically correct
  - However, their structures may not make sense from a semantic perspective
  - A mechanism is needed to ensure that documents make sense

An Example Document that is Well-Formed

```
<?xml version="1.0"?>
<course_list>
    <course internal_id="I-02237">
        <name>XML: Technology and Application</name>
        <number>635.781</number>
        <number>605.741</number>
        <instructor>Silberberg</instructor>
    </course>
    <course internal_id="I-20087">
        <name>User Interfaces</name>
        <number>635.721</number>
        <number>605.766</number>
        <instructor>Gersh</instructor>
    </course>
</course_list>
```

Another Well-Formed Document

```
<?xml version="1.0"?>
<name>
    <instructor internal_id="I-02237">
        <course>XML: Technology and Application</course>
        <number>635.781</number>
        <number>605.741</number>
        <course_list>Silberberg</course_list>
    </instructor>
    <instructor internal_id="I-20087">
        <course>User Interfaces</course>
        <number>635.721</number>
        <number>605.766</number>
        <course_list>Gersh</ course_list>
    </instructor>
</name>
```

Well-formed, but does not make sense (semantically incorrect).

DTD Benefits

- Describes document structure
- Shares document structure definition with other applications
- Enables software to verify and validate document structure correctness
  - Are the right elements (and attributes) there?
  - Are the wrong elements (and attributes) not there?
- Ensures data types are correct
- Provides default values

- DTDs enable automated software to check these

More on DTDs

- A DTD describes an *XML vocabulary*
  - A cooperating set of documents
  - A cooperating set of applications that share the documents
- A group of XML documents that share the same vocabulary is known as a *document type*
- An individual XML document that conforms to a document type is a *document instance*
- A Document Type Definition (DTD) is a formal description of the syntax (and some of the semantics) of an XML document

DTD Locations

- DTDs can be defined in
  - A single document that is external to XML files.
  - Internally, within an XML document
  - A combination of both
  - Internal definitions override external definitions
- The DTD specification provides a standard syntax for describing document types
- Validating parsers can examine DTDs to determine the validity of XML documents

Common Scenario

- Multiple applications that share XML documents
  - Some applications create documents
  - Some read documents
  - Some modify documents
- Applications need to verify that the documents are structured according to a predefined semantics
  - Content verification
    - Ensure all required elements are present
    - Ensure disallowed elements are not present
  - Semantic interpretation - structure & data
  - Provide default values
  - Error handling

Alternative Approaches to XML Validation

- Special software to perform application-specific checking
  - Could streamline performance
  - However, each application would require a customer parser to perform checks
    - Error-prone
    - Difficult to maintain - especially across organizations
    - Code duplication
- External custom syntax checker
  - Eliminates problem with code duplication
  - Reduces, but does not eliminate, errors
  - Easier to maintain
  - However, a new syntax checker must be created and maintained for each document type

The DTD Approach

- Standard specification and syntax for defining document types
- Standard validating parsers can be built in accordance with the specification
- Semantics are declarative, rather than procedural
  - Changes are introduced by changing file specifications
  - Software never needs modification for verification
  - Application software
    - Needs to change in either case
    - However, with DTDs, changes are only required for operations on XML documents that are specific to the applications themselves

Validating Parsers

- Simple XML parsers to ensure that XML documents are well formed
  - Validates the syntax
  - Internet Explorer and Netscape
- Validating parsers check the XML documents for adherence to DTD descriptions
  - Validates the semantics, as well
  - Ensures that XML documents conform to the rules of the XML vocabulary
  - Compares XML document to associated DTD

Example

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE store SYSTEM "file:/C:/My Documents/APL/XML/DTD/store.dtd">
<store>
    <cust_list>
        <customer cid="C-1" >
            <name title="Mr.">
                <first>Sam</first>
                <middle>Ta</middle>
                <middle>Chuan</middle>
                <last>Chu</last>
            </name>
            <address>
                <street>123-A Kensington Circle</street>
                <city>London</city><country>England</country>
            </address>
            <purchase date="2002-01-12" items="I-34 I-62 I-15" qty="3 1 1" />
            <purchase date="2002-01-13" items="I-15" qty="2" />
        </customer>
```

Example (2)

```
        <customer cid="C-2" ctype="bad">
            <name title="Dr.">
                <first>Darsana</first>
                <last>Sudarsen</last>
            </name>
            <address>
                <street>11100 Johns Hopkins</street>
                <city>Baltimore</city><state>MD</state>
                <zip>21207</zip>
            </address>
            <purchase date="2002-02-02" items="I-62 I-5" qty="2 1" />
        </customer>
    </cust_list>
    <inventory_list>
        <item item_no="I-5" supplier_id="S-1">
            <description>Ivory Soap</description>
            <in_stock>50</in_stock>
            <price>1.09</price>
            <cost>.28</cost>
        </item>
```

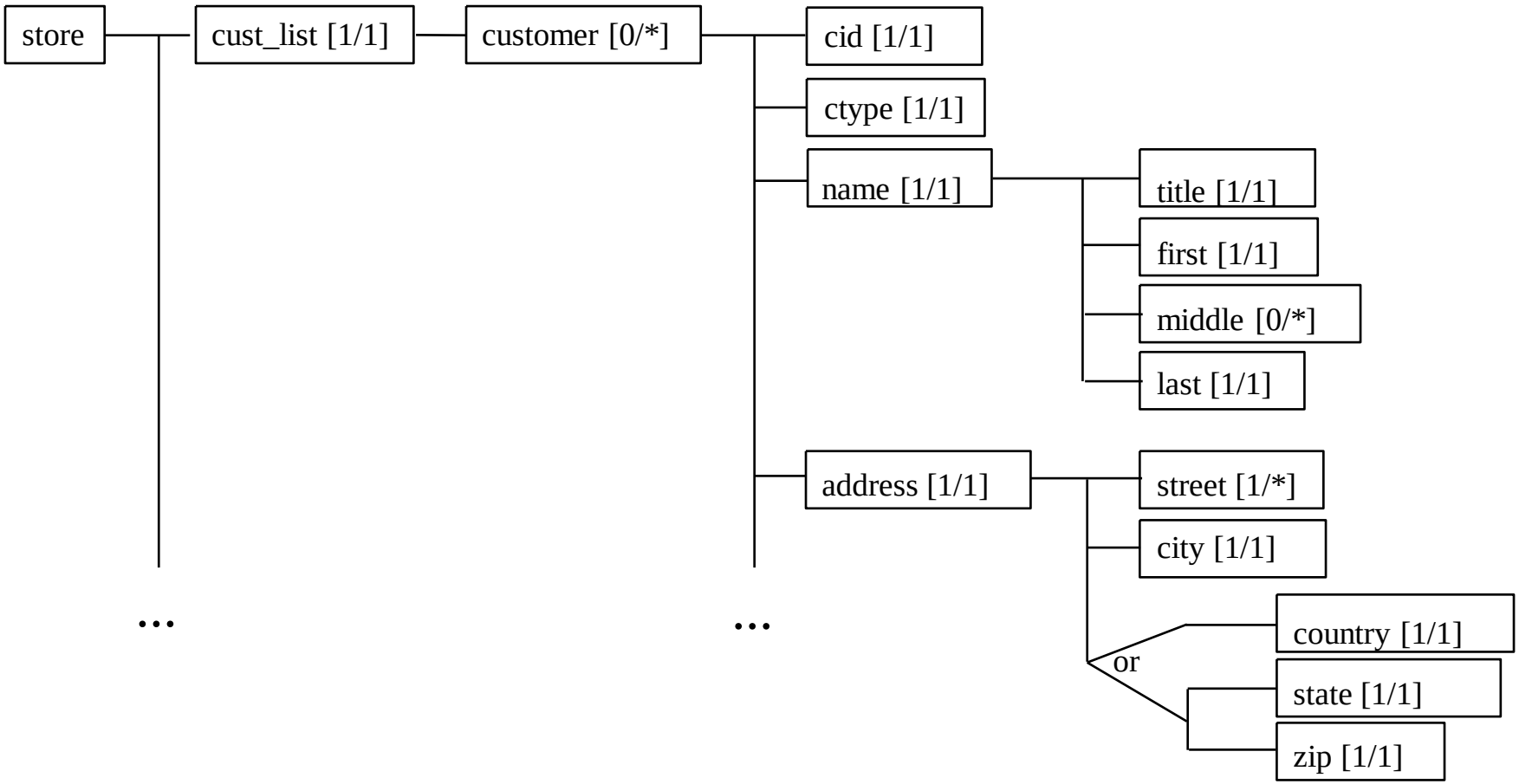
Example (3)

```
<item item_no="I-15" supplier_id="S-1">
    <description>Cheer Detergent</description>
    <in_stock>37</in_stock>
    <price>5.98</price>
    <cost>2.10</cost>
</item>
<item item_no="I-34" supplier_id="S-1">
    <description>Bounty Paper Towels</description>
    <in_stock>112</in_stock>
    <price>1.48</price>
    <cost>.57</cost>
</item>
<item item_no="I-62" supplier_id="S-2">
    <description>Sunshine Tissues</description>
    <in_stock>320</in_stock>
    <price>1.65</price>
    <cost>.98</cost>
</item>
</inventory_list>
```

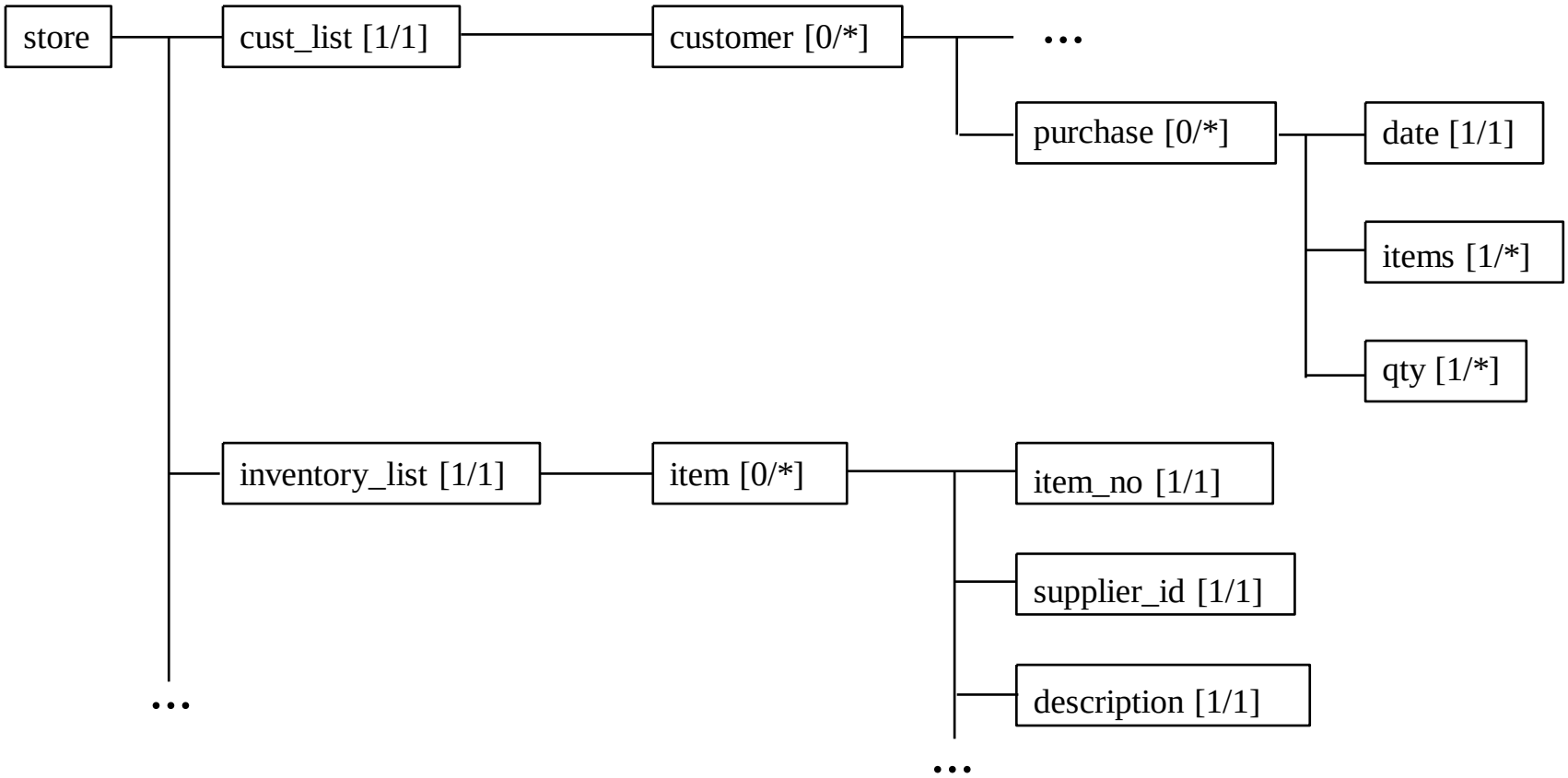
Example (4)

```
        <supplier_list>
                <supplier sid =“S-1”>
                        <company>Proctor and Gamble</company>
                        <telephone>1-800-PAMPERS</telephone>
                </supplier>
                <supplier sid =“S-2”>
                        <company>Sunshine Products</company>
                        <telephone>1-888-344-9881</telephone>
                </supplier>
        </supplier_list>
</store>
```

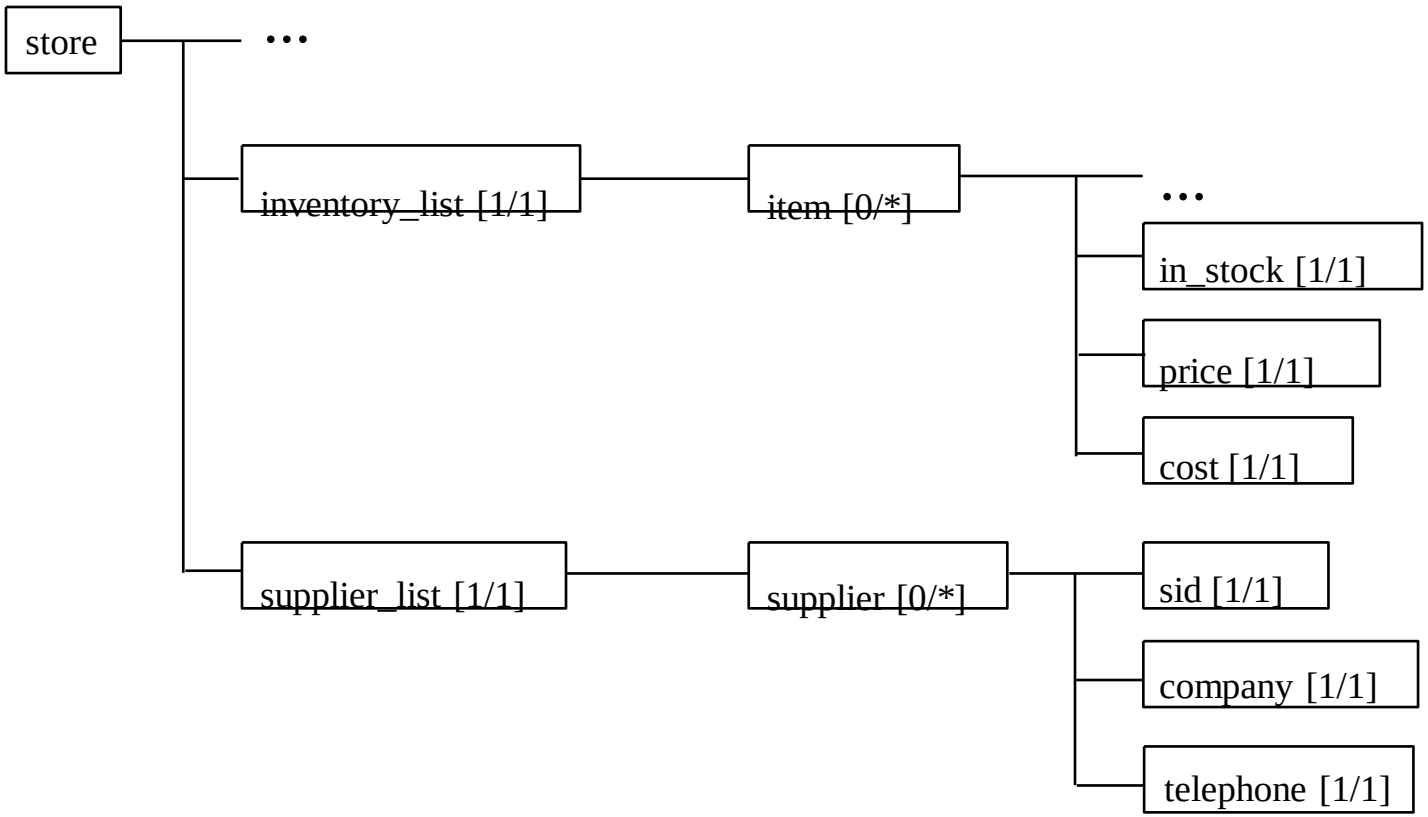
Data Model


store
cust_list [1/1]
customer [0/*]
cid [1/1]
ctype [1/1]
name [1/1]
title [1/1]
first [1/1]
middle [0/*]
last [1/1]
address [1/1]
street [1/*]
city [1/1]
country [1/1]
or
state [1/1]
zip [1/1]
...
...

Data Model (2)


store
cust_list [1/1]
customer [0/*]
...
purchase [0/*]
date [1/1]
items [1/*]
qty [1/*]
inventory_list [1/1]
item [0/*]
item_no [1/1]
supplier_id [1/1]
description [1/1]
...
...

Data Model (3)


store
...
inventory_list [1/1]
item [0/*]
...
in_stock [1/1]
price [1/1]
cost [1/1]
supplier_list [1/1]
supplier [0/*]
sid [1/1]
company [1/1]
telephone [1/1]

Internal vs. External DTDs

- DTDs can be divided into two parts
  - Internal subset (included in the document)
  - External subset (external to the document)
- Can combine internal and external subsets within a single document
- Internal declarations override the external declarations
- Use of DTDs
  - Generally, one should use external DTDs!
  - Changes are made to one document
  - Maintenance nightmare if DTDs are in document instances

Associating DTDs with an XML Document

- Need a Document Type Declaration (DOCTYPE - not DTD!)
- Specifies the document(s) to search for the associated DTD
  - At most, two places will be searched
  - Only one of the two documents is used for the DTD
    - If the first matches, it will be used
    - If there is no document at the first location, the second will be used
    - More on this later
- The use of only one DTD is very limiting
  - Cannot build up a larger vocabularies from smaller subsets
  - Cannot merge vocabularies

XML Declaration

- Standard declarations
  - <?xml version="1.0" encoding="…" standalone="yes"?>
  - <?xml version="1.0" encoding="…" standalone="no"?>
- Standalone attribute
  - yes - document is self-contained
    - No need for external DTD
    - If there is a DTD, it is internal
  - no - document refers to external DTD

DOCTYPE Declaration

<!DOCTYPE document_root_element source location1 location2 [internal subset of DTD] >

- Must follow the <?xml …?> declaration
- Must precede all element and character data
  - Comments can precede a DOCTYPE statement
  - Processing Instructions (PIs) can precede a DOCTYPE statement
- XML documents do not need to be validated
- If one wants a document to be validated, one must use a DOCTYPE command
- Only one DOCTYPE per XML document
  - Can be divided into internal and external definitions

DOCTYPE syntax

Example: <!DOCTYPE store SYSTEM
“file:/C:/My Documents/APL/XML/DTD/store.dtd”>

Example: <!DOCTYPE store SYSTEM “store.dtd”>

- < > delimiters
- **!** - special XML declaration
- **DOCTYPE** keyword
- **document_element** - name of the top-level (root) element name
  - **store** - example for this document
  - Must be a single set of <store> ... </store> tags at the top level of an associated document

DOCTYPE syntax (2)

- DTD source
  - Only two options - **SYSTEM** or **PUBLIC**
  - Must declare at least **location1**
  - If **PUBLIC** is used, **location2** may be declared
- **SYSTEM**
  - **location1** must be a Uniform Resource Locator (URL) or more general Uniform Resource Identifier (URI)
  - Parser attempts to retrieve DTD from URI
  - If not found, an error is reported
  - URIs with URL fragment identifiers (#) should not be used
    - "file:/C:/My Documents/APL/XML/DTD/store.dtd#section_1"

DOCTYPE syntax (3)

- **PUBLIC**
  - Useful when applications are spread out
  - Each application requires quick access to the DTDs
  - Rather than having the DTD at one location, make copies and place them local to the applications
  - If local copy not available use master copy
    - **SYSTEM** keyword is implied

Example: <!DOCTYPE store PUBLIC

“store.dtd” “http://www.store.com/store.dtd”>

DTD Markup

- DTDs define the elements of a document and their structure
- Syntax
  - <!keyword parameter1 parameter2 … paramenterN>
  - <!keyword
    parameter1
    parameter2
    …
    parameterN>
  - Whitespace and carriage returns are permitted

Keywords

- ELEMENT
  - Declares XML element name
  - Declares permitted children of the element
  - <!ELEMENT …>
- ATTRIBUTE
  - Declares XML attribute names
  - Declares permitted attribute values
  - <!ATTRIBUTE …>

Keywords (cont.)

- ENTITY
  - Declares special character references, text macros, and other content from external sources
  - <!ENTITY …>
- NOTATION
  - Declares external non-XML content (e.g., images, etc.)
  - Declares applications that handle them
  - <!NOTATION …>

ELEMENT Declarations

- All XML documents use at least one element (the document root element)
- Almost all XML documents contain sets of nested elements
- Two types of ELEMENT declarations
  - <!ELEMENT name category >
  - <!ELEMENT name (content_model) >
- Name is the actual element name
- Category and content model parameters describe the content that ELEMENTs may contain

ELEMENT Names

- ELEMENT name rules
  - Must use XML NameChar characters
  - Unicode letters
  - Numbers
  - “_”, “:”, “-”, “.”
  - Must start with Unicode character, “_” or “:”
  - Colons are not recommended because they can be confused with namespace delimiters

Content Categories - ANY

- ANY
  - Element type can contain any well-formed XML data
  - Character data
  - Other elements
  - Comments
  - Processing Instructions (PIs)
  - Character Data (CDATA) sections
  - <!ELEMENT someEltName ANY>
- Disadvantage
  - Since anything can go in this element, no real checking is done
  - Parsers just ensure that the element and its sub-hierarchy is well-formed

Content Categories - EMPTY

- EMPTY
  - Can only contain ATTRIBUTES
  - DTD specifications
    - <!ELEMENT purchase EMPTY>
    - <!ELEMENT BR EMPTY>
  - XML document example
    - <purchase date="02-02-2002" items="I-62 I-5" qty="2 1"/>
    -
 or
 </BR>

Content Models

- <!ELEMENT element_name (content_model)>
- Content_model can be:
  - **PCDATA**
    - Parsed character data
  - **element**
    - One ore more child elements
  - **mixed**
    - Combination of PCDATA and elements

PCDATA Content

- Parsed character data
- DTD specifications
  - <!ELEMENT city (#PCDATA)>
  - <!ELEMENT description (#PCDATA)>
- XML examples
  - <city>Baltimore</city>
  - <description>Ivory Soap</description>

Τέλος Ενότητας