

XML and Databases

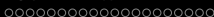
Chapter 1: XML Syntax

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

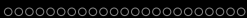
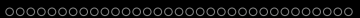
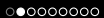
Winter 2023/24

<http://www.informatik.uni-halle.de/~brass/xml23/>



Contents

- 1 Introduction
- 2 XML Documents
- 3 DTDs
- 4 DOCTYPE Decl.



Introduction (1)

- XML (“eXtensible Markup Language”) is basically a simplification (subset) of SGML (“Standard Generalized Markup Language”).

SGML is an ISO-Standard since 1986. XML was developed mainly 1996, and became an W3C Recommendation on February 10, 1998.

- HTML was too restricted for exchanging semantic data over the internet (not only text documents): User-defined tags are needed.

E.g., there is no tag “<price>” in HTML.

- The browser vendors complained that SGML was too complex.



XHTML Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!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">
  <head>
    <title>My first XHTML document</title>
  </head>
  <body>
    <h1>Greeting</h1>
    <p>Hello, world!</p>
  </body>
</html>
```


Contents

- ① Introduction
- ② XML Documents**
- ③ DTDs
- ④ DOCTYPE Decl.

Elements (1)

- An XML/SGML document is a text, in which words, phrases, or sections are marked with “tags”, e.g.

```
<title>My first XHTML document</title>
```

- “<title>” is an example for a start-tag.
- “</title>” is an example for an end-tag.
- Specialized editors also use other symbols on the screen, e.g.

```
title➤ My first XHTML document ◀title
```


Elements (3)

- Quite often, “tag” is used when “element” would be formally right:

- A tag is the string from “<” to “>” (inclusive).
- “The title tag” is not quite correct.

Unless you refer to a specific occurrence in the text, but there are always two title tags, the start tag and the end tag.

- Better say “The title element of the document” or something similar.

No points will be taken off because this confusion is so common.

Elements (5)

- The contents of an element is the text between start-tag and end-tag.

- E.g. consider again

```
<title>My first XHTML document</title>
```

- The contents of this element is:

```
My first XHTML document
```

- For each element type, one can define in the DTD what exactly is allowed as contents of these elements (“Content Model”).
- E.g. elements of the type `title` can contain only pure text in XHTML (one cannot nest any other elements inside).

Elements (6)

- The element type “ul” (unordered list) contains a sequence of elements of the type “li” (list item):

```
<ul><li>First</li><li>Second</li></ul>
```

- Since elements can contain themselves elements, one can understand an SGML document as a tree:
 - Inner nodes are labelled with elements.
 - Leaf nodes are labelled with text or with elements (which have empty contents in this case).

Elements (9)

- Elements cannot overlap only partially.

For each two elements *A* and *B*, either *A* is completely contained in *B*, or *B* completely in *A*, or the two do not overlap at all.

- This means that opening and closing tags must be nested correctly: E.g. the following is legal:

```
<h1><code>...</code></h1>
```

However, this is a syntax error:

```
<h1><code>...</h1></code>
```

- Begin and end tags work like parentheses of different types: ([]) is legal, but [()] is not.

Elements (10)

- Four kinds of element types can be distinguished:
 - Element types that can only contain text.
 - Element types that can only contain other element types.

Of course, these other elements might contain text. The DTD defines which element types are exactly valid inside the given element type and in which sequence they must appear.

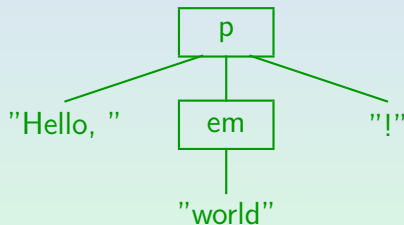
- Element types that can contain a mixture of text and other elements (“mixed content model”):

```
<p>Hello, <em>world</em>!</p>
```

- Element types that always have empty contents.

Elements (11)

- The tree representation of an element with mixed content looks as follows:



- Elements with empty contents work as markers. E.g. “**br**” (break) does a line break in XHTML.

Empty Elements

- In XML, for every opening tag there must be a corresponding closing tag.

In contrast, SGML has “tag minimization” rules that permit to leave out tags that can be uniquely reconstructed by the SGML parser.

- Since `
</br>` does not look very nice, and the closing tag contains no additional information, empty element tags were introduced in XML: “`
`” is equivalent to “`
</br>`”.

This was one of the few points where XML was not a subset of SGML at the beginning. Of course, SGML was/will be extended to permit this syntax, too.

Line Ends

- In SGML, line ends (record boundaries) directly after a start tag or directly before an end tag are ignored (i.e. at the start or end of the content).

- In XML, line ends or empty space is not ignored.

The parser passes it to the application, which can of course ignore it.

E.g. a validating parser, which knows that an element contains pure element content (not mixed content) will ignore whitespace between the elements.

- In XML, line ends are normalized to a line feed.

Even on a Windows system (which uses CR, LF for line ends), the XML application receives LF (ASCII 10) from the parser.

Attributes (1)

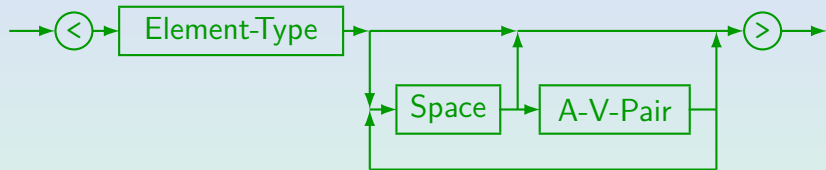
- In the start tag, attribute-value pairs can be optionally specified.
- E.g. in XHTML, links to other documents are marked with the element `a` (“anchor”):

```
XML was developed by the  
<a href="http://www.w3.org">W3C</a>.
```

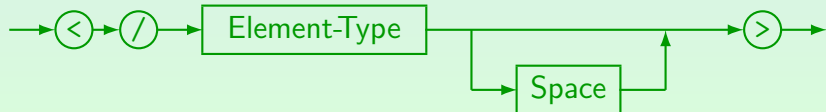
- The text of the reference is given in the element content, the URI of the referenced web page is specified in the attribute “`href`”.

Attributes (2)

Start-Tag:

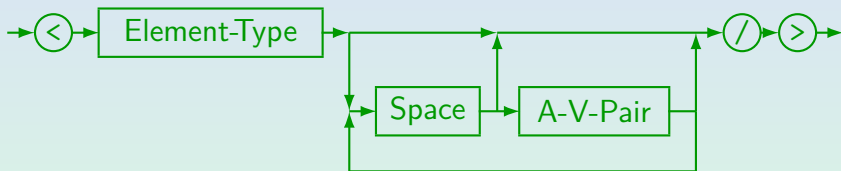


End-Tag:



Attributes (3)

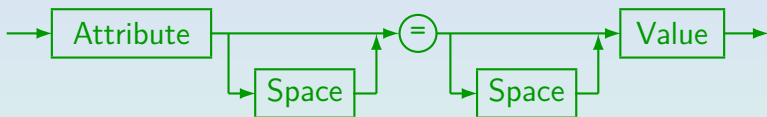
Empty Element Tag:



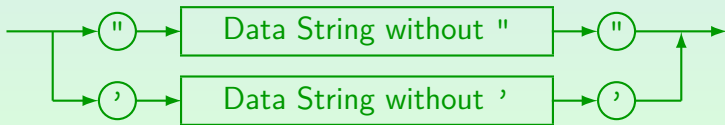
- “Space” (white space) consists of one or more space characters, carriage returns, line feeds, and tabs (ASCII 32, 13, 10, 9).

Attributes (4)

A-V-Pair:



Value:



Attributes (5)

- Attribute values can be enclosed in " or ' .
The other sign can appear inside the string.

If one needs both quotation marks, one must use an entity or character reference (see below).

- Attribute values cannot contain elements.
- The character "<" is forbidden in attribute values.

If necessary, one can include it with a character reference or an entity reference. Excluding "<" in attribute values helps to detect errors earlier (such as a missing quote). To make this clear: "<" is not forbidden in the internal value of an attribute (which an XML parser can pass to the application), it is only forbidden in the external representation. But it never creates elements in attribute values.

Attributes (6)

- The character “&” is treated special in attribute values (character/entity reference, see below).
- Attribute values can extend over multiple lines. The parser replaces tabs and line ends in the attribute value by a space.

Depending on the type of the attribute, white space may be normalized: It is then removed at the beginning and at the end of the attribute value, and several consecutive spaces are merged into one. However, this does not happen for normal “CDATA” attributes.

- The sequence in which several attribute-value-pairs are listed in a tag is not important.

Character References (1)

- One must distinguish between
 - the repertoire of characters used internally (e.g. data passed from XML parser to application)
 - the encoding of these characters in bytes for exchanging documents (external representation).
- Internally, XML uses the Unicode character set.
- For exchanging documents, one can e.g. use the ISO 8859-1 (ISO Latin 1) character codes, which contains only a subset of all Unicode characters.

Other encodings contain e.g. cyrillic or japanese characters.

Character References (2)

- The XML declaration at the beginning of the XML file defines the encoding (see below).

The encoding can also be specified in the HTTP protocol.

- The characters in the ISO Latin 1 character set are also contained in the Unicode character set and have the same numeric codes in both character sets.

I.e. Unicode is upward compatible to ISO Latin 1. However, the encoding as sequence of bytes is different. At the beginning, Unicode character numbers had 16 Bit, now there are 17 planes of 16 Bit each. With the UTF-8 encoding of Unicode, at least the 7-bit ASCII characters have the same encoding in ASCII, ISO Latin 1, and Unicode. However, for German national characters (ä, ö, ü, etc.) this is no longer true: UTF-8 uses two bytes for them. See Slide 36.

Character References (3)

- Characters that cannot be directly entered, can be written as a “character reference” using their numeric code:

`ä`

is an “ä”. Hexadecimal notation can also be used:

`ä`

- The numbers refer to the repertoire (i.e. Unicode), not to the encoding for exchange.

ASCII and ISO Latin 1 codes can be used since Unicode is upward compatible.

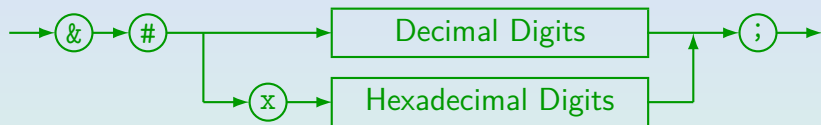
- Character references can also be used to “escape” characters that otherwise would have special meaning in SGML/XML.

The result of a character reference is always treated as data.

E.g. if a double quote (ASCII 34) needs to be included in an attribute value that is enclosed in double quotes, one can write it as “"”.

Character References (4)

Character Reference:



- In DTDs, abbreviations/macros (“entities”) can be defined (see appendix).
- In this way, one does not have to remember character codes.

E.g. in HTML, one would write “`ä`” for an “ä” (if one wants to stick to pure ASCII). In XML, this is not predefined.

Short Digression: UTF-8 (1)

- The character “ä” has the number (“code point”) `U+00E4` (228) in Unicode.
- It has the same number in ISO Latin 1, and this encoding uses one byte per character, therefore it is represented by the byte `0xE4`.
- UTF-8 uses one byte per character only for characters with codes up to `U+007F` (127).

I.e. when the first bit of a byte is 0, this byte encodes a character by itself.
This ensures that UTF-8 is upwards compatible to ASCII.

Short Digression: UTF-8 (2)

- For characters with code point above **U+007F** (including **ä**), multi-byte sequences are needed in UTF-8:
 - For n -byte sequences, the first byte starts with n **1**-bits followed by a **0**-bit.

E.g. the prefix 110 in the first byte means that this byte sequence consists of two bytes.
 - All other bytes start with **10**.

If one jumps somewhere into an UTF-8 byte stream, one can detect the start of character encodings (bytes starting not with 10).
- Unicode reserves 17 planes of 16 bit each (up to **U+10FFFF**), which requires max. 4 bytes in UTF-8.

Short Digression: UTF-8 (3)

- A two byte sequence consists of bytes **110xxxxx** and **10yyyyyy**, representing the code point **xxxxxyyyyyy** and is used for the range **U+0080** to **U+7FF**.
- The character “ä” with code **U+00E4** (**11100100**) is encoded as **11000011** (**0xC3**) and **10100100** (**0xA4**).
- If an editor assumes that the text is encoded in ISO Latin 1, it will display the two characters **Ã ¤**.

The second is a “general currency symbol”: U+00A4 (= 164).
- Each code point must be encoded in the shortest byte sequence (e.g. ASCII characters as one byte).

Comments (1)

- Comments can be used to enter notes or explanations for a reader of the SGML/XML source file into the document.
- Comments are ignored by programs that process an SGML/XML file. E.g. they might not appear in the formatted output.

The XML standard permits that an XML parser passes comments to the application program, but it does not require this.

- A comment in SGML/XML has the form

```
<!-- This is a comment -->
```

Comments (2)

- Comments can extend over several lines.

I.e. they do not have to be closed on the same line.

- Within a comment, it is forbidden to write two consecutive hyphens “--”.

In SGML, the comment actually extends from “--” to “--”. However, it can only be used in a markup declaration, which starts with “<!” and ends with “>”.

- Tags within a comment are permitted, but confuse many browsers.

Browsers try to correct syntax errors. When they see a tag, they might assume that the author forgot the “end of comment” mark.

Comments (3)

- Comments can be used anywhere in the document outside other markup.
- They cannot be used within tags.
- In SGML (but not in XML), comments “`-- ... --`” can appear in markup declarations at places permitted by the grammar.

In modern programming languages, whitespace including comments is allowed between tokens. SGML/XML are different: maybe because they are languages for writing documents, not programs, maybe they are a bit outdated in this aspect.

- XML supports only “`<!-- ... -->`”.

Exercise

Please find syntax errors:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<GradesDB3>
  <student sid='101' first='Ann' last="Smith"
    <result cat='H' eno = 1 points=10>
    <result cat='H' eno ='1' points='8'>
    <result cat='M" eno ="1" points='12'
      >
  </ student >
  <!------- Exercises ----->
  <ex cat='H' eno='1' note='<em>difficult</em>''
    Rel&#97 tional Algebra</ex>
</Grades-DB>
```

Software (1)

- Although browsers are very generous with syntax errors in HTML documents, they show all errors in XML documents.

E.g. Internet Explorer, Firefox.

- They check only the syntax of well-formed XML, they do not validate documents against a DTD.
- If no style sheet is given, the document tree is displayed (child nodes are indented under the parent).

It is possible to collapse/expand subtrees by clicking on the `-/+` in front of the elements.

Software (2)

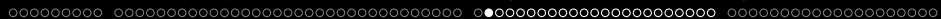
- Xerces from the Apache Software Foundation is an example for a validating parser for XML (supporting DTDs and XML Schema).

See [<http://xerces.apache.org/>]. It has a DOM and a SAX interface for accessing the parsed data. It comes with a test program domprint, which can be used for checking the syntax (it is an unparser, i.e. it outputs the result of parsing again as XML, but probably differently formatted). There is a C++ and a Java version, and a Perl interface to the C++ version.

- There are also validation services on the web, e.g.
 - [<http://www.xmlvalidation.com/>]
 - [https://www.w3schools.com/xml/xml_validator.asp]

Contents

- 1 Introduction
- 2 XML Documents
- 3 DTDs
- 4 DOCTYPE Decl.



Example

Simple DTD for a HTML-Subset:

```

<!ELEMENT html (head, body)>
<!ELEMENT head (title)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT body ((#PCDATA|p|em|ul)*)>
<!ELEMENT p ((#PCDATA|em|br)*)>
<!ELEMENT em (#PCDATA)>
<!ELEMENT br EMPTY>
<!ELEMENT ul (li+)>
<!ELEMENT li ((#PCDATA|p|em|ul|br)*)>

```

Element-Type Declarations (1)

An Element-Type Declaration consists of:

- “<!” followed by the keyword “ELEMENT”.

In SGML, one could define a different string instead of “<!”. This is the parameter MDO (“Markup Deklaration Open Delimiter”) in the SGML declaration. Correspondingly, “>” is called MDC (“Markup Declaration Close Delimiter”). XML is based on a fixed SGML declaration, so one cannot change these delimiters.

- Name of the element type to be declared.

Such names are officially called “Generic Identifiers”.

- Then one specifies what is permitted as content of this type of elements (“content model”).
- “>”.

Element-Type Declarations (2)

Element-Type Declaration:



White space is required between “<!ELEMENT” and the name, and between the name and the content specification. It is permitted but not required between content specification and the “>”.

Names in XML must start with a letter, an underscore “_” or a colon “:”, and can otherwise contain letters, digits, periods “.”, hyphens “-”, underscores “_”, colons “:”, or certain special Unicode characters. Names starting with “xml” in any capitalization are reserved, the colon is treated specially by the XML namespace standard.

The element type declaration in SGML is more complicated: There, also specifications for markup minimization are required (if the markup minimization parameter OMITTAG is set), “exclusions” and “inclusions” are possible, several element types can be declared together, etc.

Content Specifications (1)

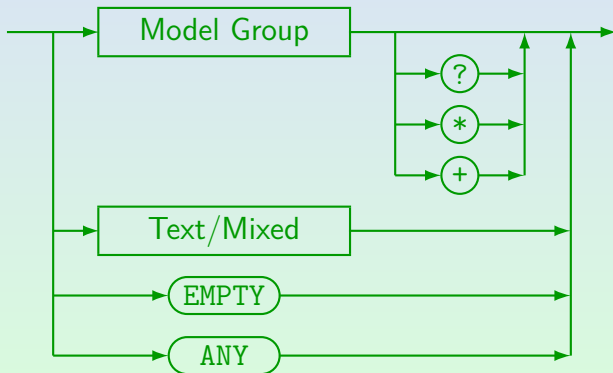
- The building blocks of content specifications are
 - Names X of element types: This pattern matches exactly one element of type X , i.e. basically `<X>...</X>`.
 - The keyword `#PCDATA`: Pure textual data without tags (but possibly character/entity references).
 - #PCDATA stands for “Parsed Character Data” (it is checked that there are no nested tags). SGML (not XML) has also CDATA (treats `<` as text).
- One can specify the optionality/multiplicity of elements and groups by attaching occurrence indicators:
 - `A?`: Optional, non repeatable (0 or 1 time).
 - `A*`: Optional, repeatable (0 or more times).
 - `A+`: Required, repeatable (1 or more times).

Content Specifications (2)

- Content specifications can be connected with
 - $(A_1 | \dots | A_n)$: “Alternative”/“Choice” (one of the A_i).
E.g. $(A | B)$: “A or B”. The content must match A or match B.
 - (A_1, \dots, A_n) : “Sequence” (all A_i in the given sequence).
E.g. (A, B) : “First A, then B” (“A followed by B”). A prefix of the content must match A, the rest B.
SGML (not XML) has also $(A \& B)$: “A and B”. A and B must both appear, but in arbitrary sequence. This is equivalent to $((A, B) | (B, A))$.
- The A_i are
 - An element type (possibly with $?$ / $*$ / $+$).
 - #PCDATA (in XML with restrictions, see below).
 - A nested model group (possibly with $?$, $*$, $+$).

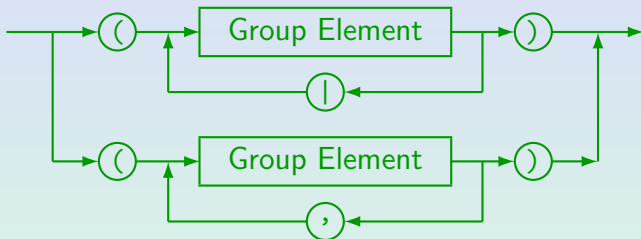
Content Specifications (4)

Content:

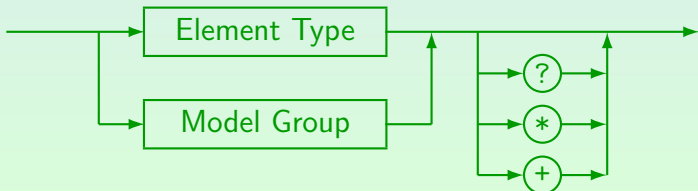


Content Specifications (6)

Model Group:



Group Element:



Content Specifications (7)

- In SGML and XML, the possible occurrence of white space is defined by the grammar.
- It is permitted but not required between each two tokens (“word symbols”) in content models, except before the occurrence indicators “?”, “*”, “+”.
- The keyword “#PCDATA” requires the symbol “#” (RNI, “Reserved Name Indicator”) in order to distinguish it from an element type named “PCDATA”.

Other keywords like “EMPTY” do not use it, since in the element type declaration, they appear outside of parentheses, while user-defined names must appear inside parentheses.

Attribute Declarations (1)

- Example (symbol used for marking list items):

```
<!ATTLIST UL type (disc|square|circle) #IMPLIED>
```

In HTML 4.01 Strict this attribute was removed.

- Several attributes (of one element type) can be declared in a single `ATTLIST` command.
- E.g. some attributes of images in HTML:

```
<!ATTLIST IMG src CDATA #REQUIRED  
alt CDATA #REQUIRED  
width CDATA #IMPLIED  
height CDATA #IMPLIED>
```


Attribute Data Types (1)

- E.g. (**yes**|no): Enumeration type.

The attribute value must be one of the listed values. Each value is a “name token” (NMTOKEN), i.e. a sequence of characters that can appear anywhere in identifiers (letters, digits, and certain special characters). E.g. a sequence of digits would be valid. In SGML, it is forbidden that same enumeration type value is used for two attributes of the same element type. In XML, this is recommended “for interoperability”.

- **CDATA**: Sequence of arbitrary characters.

The character “&” is interpreted, i.e. one can use character and entity references in the attribute values. In XML, “<” is forbidden in attribute values (so that missing quotes are easier found), and “>” is not interpreted (treated as data). In SGML, “<” and “>” are valid, but not interpreted. Thus, attribute values still cannot contain elements.

Attribute Data Types (2)

- **ID**: A name that uniquely identifies this element (within the entire document).

The syntax is the same as for element type names (sequence of letters and digits plus `_`, `:`, `.`, `-`, starting with letter or `_`, `:`, `.`). Two elements must not have the same value for an attribute of type **ID**. This even holds for elements of different type. The same element type cannot have two attributes of type **ID**. One should use the same name for all attributes of type **ID**, and the attribute name "**ID**" is very common.

- **IDREF**: A name that appears as value of an **ID**-attribute somewhere in the document.
- **IDREFS**: List of **IDREF**-values.

The single values are separated by white space.

Attribute Data Types (3)

- **NMTOKEN**: Sequence of name characters.
An arbitrary sequence of letters, digits, “_”, “-”, “.”, and “:”.
This is the most restricted type one can use for numbers. There is no numeric type in XML.
- **NMTOKENS**: List of **NMTOKEN**-values.
- **ENTITY**: Name of an entity (not in exam).
Entities are a kind of macros or include files (see appendix). An attribute of type **ENTITY** takes as value the name of a declared unparsed entity.
- **ENTITIES**: List of **ENTITY**-values (not in exam).
- **NOTATION** ($N_1 | \dots | N_m$): One of the notations N_i .
(Not in exam.) The N_i must be declared as notations (data formats). Only one attribute of an element type can have the type **NOTATION**. This attribute defines the format of the content of the element.

Default Values (2)

- Instead of a default value, one can also specify:
 - **#IMPLIED**: The attribute is optional.

I.e. the default value is a "null value" different from all possible normal values. The name for the keyword was chosen because it is assumed that the application program can compute a value for the attribute. E.g. a chapter number is usually the number of the last chapter plus 1.

- **#REQUIRED**: An attribute value must be specified.
- **#FIXED "Value"**: The attribute can have only this single value that is specified in the DTD.

This is e.g. used when many/all element types have an attribute with the same name, and for each element type a (possibly different) value is declared in the DTD.

Exercise (1)

Please find syntax errors:

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<!DOCTYPE GradesDB4 [  
    <!-- contains syntax errors -->  
    <!ELEMENT GradesDB4 (STUDENT, RESULT)*>  
    <!ELEMENT STUDENT RESULT+>  
    <!ATTLIST STUDENT FIRST CDATA #REQUIRED  
                    LAST CDATA #REQUIRED>  
    <!ELEMENT RESULT #EMPTY>  
    <!ATTLIST RESULT EX_ID IDREF #REQUIRED  
                    POINTS NMTOKEN #OPTIONAL>  
    <!ELEMENT EXERCISE #PCDATA>  
    <!ATTLIST EXERCISE ID ID #REQUIRED>  

```

Exercise (2)

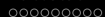
Please validate against DTD on last slide:

```
<GradesDB4>
  <student sid='101' first='Ann' last='Smith'>
    <email>smith@acm.org</email>
    <result ex_id='H1' points='A+'/>
    <result ex_id='2' points='8'/>
    <result ex_id='M1' points='12 points'/>
  </student>
  <student first='Maria' last='Brown'/>
  <exercise id='H1'>ER</exercise>
  <exercise id='2'>SQL</exercise>
</GradesDB4>
```

Exercise (3)

Please develop a DTD for this document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<smallbusiness>
  <product id='P01' name='Apple' price='0.40'>
    Really <em>delicious</em>Apples!</product>
  <product id='P02' name='Banana' price='0.50'>
    The best bananas!</product>
  <order id='R100' customer='Ann Smith'>
    <item prodid='P01' />
    <item prodid='P02' quantity='5' /> </order>
  <order id='R100' customer='Maria Brown'>
    <item prodid='P01' quantity='3' /> </order>
</smallbusiness>
```



Contents

- 1 Introduction
- 2 XML Documents
- 3 DTDs
- 4 DOCTYPE Decl.

System/Public Identifiers (1)

- In SGML/XML DTDs and other objects (e.g. entities, notations, see appendix) can be identified by system and public identifiers.
- In XML, the system identifier is more important.
- In XML, the system identifier must be a URI/URL (without fragment identifier, i.e. without #).
- Local file names are relative URIs and are therefore permitted.

In SGML, the system identifier typically was a local file name. Since the directory structure can be different on different computers, the system identifier was system dependent.

System/Public Identifiers (2)

- Public identifiers are system-independent and very stable.
 - They were especially important in SGML: Otherwise it was quite likely that documents had to be changed when they were moved from one system to another. For XML, this problem is much smaller, because a URI is typically “global” and relatively stable (at least URIs for globally used DTDs).
- However, public identifiers must be translated into system identifiers.
 - In the end, there must be the possibility to retrieve the file with the DTD (unless the DTD is built into the software, e.g. a web browser does not need to read the HTML DTD). Normally, an SGML system contains a configuration file that maps public IDs into system IDs.

System/Public Identifiers (4)

- Public identifiers can be any string of letters, digits, certain special characters, spaces and line breaks (enclosed in single or double quotes: ' or ").

Allowed special characters in XML: '()+,.-./:|=?!*#@\$%_%. Sequences of line breaks and spaces are replaced by a single space, and ignored at the very beginning or end.

- Example: `"-//W3C//DTD HTML 4.01//EN"`

A subset of public identifiers are called "formal public identifiers". They have more structure and must be composed from an owner identifier, a double slash "//", and a text identifier. The owner identifier starts with "ISO" for ISO publications, "+//" for registered owners, and "-//" for unregistered owners. The text identifier starts with the public text class, followed by a space, a description, a double slash "//", and the language of the text. Public text classes are, e.g., DTD and NOTATION.

