Chapter 1: XML Syntax

References:

- Boc DuCharme: XML — The Annotated Specification. Prentice Hall, 1999.
- Tim Bray, Jean Paoli, C.M. Sperberg-McQueen: Extensible Markup Language (XML) 1.0, 1998. [http://www.w3.org/TR/REC-xml] See also: [http://www.w3.org/XML].
- Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, John Cowan: Extensible Markup Language (XML) 1.1 (Second Edition), W3C Recommendation 16 August 2006, edited in place 29 September 2006. [http://www.w3.org/TR/xml11].
- Elliotte Rusty Harold, W. Scott Means: XML in a Nutshell, A Desktop Quick Ref., 3rd Ed. O'Reilly, Okt. 2004, ISBN 0-596-00764-7, 689 Seiten, 37 Euro.

Objectives

After completing this chapter, you should be able to:

- write syntactically correct XML.
- check given XML documents for syntax errors.
- explain the tree-structure of XML data.
- read XML Document Type Definitions (DTDs).
- validate an XML document against a given DTD.

Overview

1. Introduction
2. XML Documents (Syntax)
3. Document Type Definitions (DTDs)
4. DOCTYPE, XML Declaration
5. Entities, Notations, Marked Sections

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. The current version is XML 1.1 (2nd Ed.) from August 2006.

- XML/SGML has two levels:
  - ◇ It is a syntax formalism, in which (X)HTML and similar markup languages can be defined.
  - ◇ For a given DTD (grammar), XML/SGML documents contain the data or the text.

Introduction (2)

- XML/SGML is only a data format (syntax).
- It says nothing about the semantics of the data that are coded in XML/SGML.
- In contrast to SGML, where a DTD is required, XML can also be used without DTD:
  - ◇ "Well-formed XML": Basic syntax rules (proper nesting of tags) are satisfied. No DTD is needed.
  - ◇ "Valid XML": In addition, only tags defined in a DTD are used, and the content of each "tag" (element) satisfies the constraints of the DTD.

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://ww.w3.org/1999/xhtml">
    <head>
        <title>My first XHTML document</title>
    </head>
    <body>
        <h1>Greeting</h1>
        <p>Hello, world!</p>
    </body>
</html>
```


Database Example (1)

STUDENTS

SID	FIRST	LAST	EMAIL
101	Ann	Smith	···
102	Michael	Jones	(null)
103	Richard	Turner	···
104	Maria	Brown	···

EXERCISES

CAT	ENO	TOPIC	MAXPT
H	1	Rel. Algeb.	10
H	2	SQL	10
M	1	SQL	14

RESULTS

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

Database Example (2)

- Table rows can be directly translated to XML:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<GRADES-DB>
  <STUDENT SID='101' FIRST='Ann' LAST='Smith'/>
  <STUDENT SID='102' FIRST='Michael' LAST='Jones'/>
  ...
  <EXERCISE CAT='H' ENO='1' TOPIC='Rel. Algeb.'/>
  ...
  <RESULT SID='101' CAT='H' ENO='1' POINTS='10'/>
  ...
</GRADES-DB>
```

Database Example (3)

- One can also use nested elements for table entries:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<GRADES-DB>
  <STUDENTS>
    <STUDENT>
      <SID>101</SID>
      <FIRST>Ann</FIRST>
      <LAST>Smith</LAST>
    </STUDENT>
    ...
  </STUDENTS>
  ...
</GRADES-DB>
```


DB Example with Nesting

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<COURSE-DB>
    <PROFESSOR NAME='Brass' PHONE='55-24740'>
        <COURSE TERM='Summer 2004'
                TITLE='Database Design'>
            <CLASS DAY='MON' FROM='10' TO='12'/>
            <CLASS DAY='THU' FROM='16' TO='18'/>
        </COURSE>
        <COURSE TERM='Winter 2004'
                TITLE='Foundations of the WWW'>
            <CLASS DAY='WED' FROM='14' TO='16'/>
        </COURSE>
    </PROFESSOR>
    ...
```


Overview

1. Introduction
2. XML Documents (Syntax)
3. Document Type Definitions (DTDs)
4. DOCTYPE, XML Declaration
5. Entities, Notations, Marked Sections

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 (2)

- The text part from the begin of a start tag to the end of the corresponding end tag is called an element.
- The name in the start tag and the end tag is called the element type. In the example: "`title`".

  Some authors say "element name" instead of "element type". That avoids the problem that types are something different in XML Schema.
- Quite often, "tag" is used when "element" would be formally right.

  A tag is the string from "<" to ">" (inclusive).

Elements (3)

- Element types are declared in a DTD.

  E.g. the "XHTML 1.0 strict" DTD declares a certain set of element types for HTML documents that includes e.g. "`title`".
- Names (identifiers, used e.g. as element types) can contain letters, digits, periods ".", hyphens "–", underscores "_", and colons ":".

  Plus certain extended characters from the Unicode set. They must start with a letter, an underscore "_", or a colon ":". The colon should only be used in accordance with the namespace specification. All names starting with "`xml`" are reserved.
- Names are case-sensitive.

  In SGML, this can be selected in an SGML declaration.

Elements (4)

- The contents of an element is the text between start-tag and end-tag. E.g. the contents of the example element (Slide 1-12) 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).

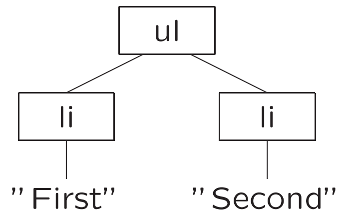
Elements (5)

- 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 (6)

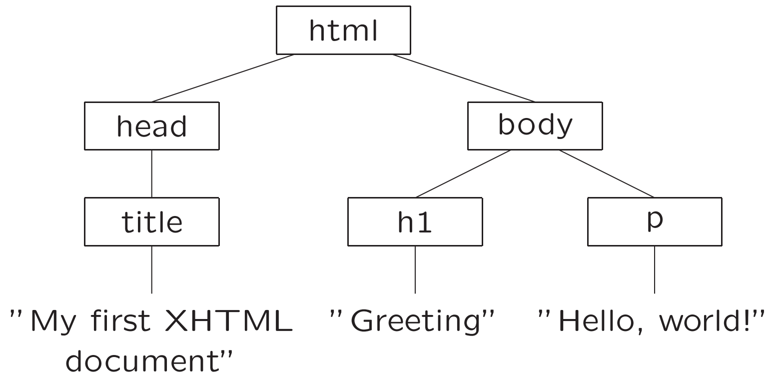
- E.g. the unordered list above has this structure:



It is called "unordered list" because bullets are used for the list items, not numbers, so presumably the exact sequence is not very important. However, in SGML and XML, the child nodes of a node always have a sequence from left to right (as given in the document). This is a difference to relational databases, where the rows in a table have no sequence.

Elements (7)

- Structure of the XHTML example (Slide 1-6):



Elements (8)

- 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 (9)

- 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.

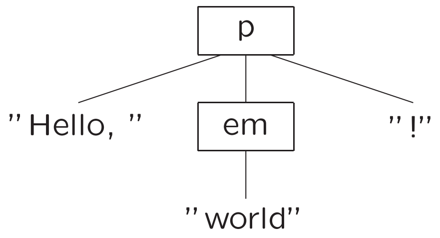
  - ◇ Element types that can contain a mixture of text and other elements ("mixed content model"):

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

  - ◇ Element types that always have empty contents.

Elements (10)

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

  This was one of the few points were 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)

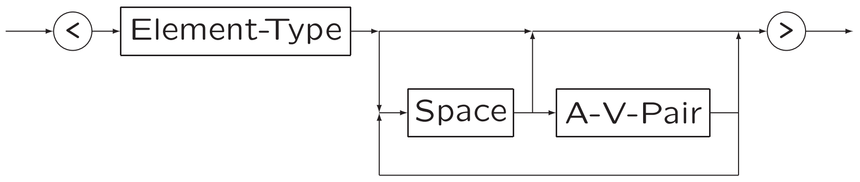
- In the start tag, attribute-value pairs can be optionally specified.
- E.g. in XHTML, links to other documents are marked with the element `a` ("anchor"):

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

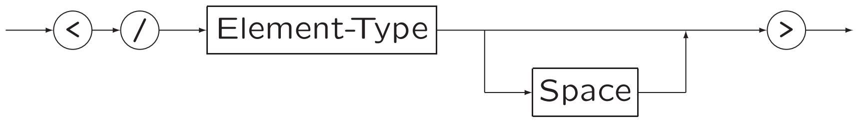
- The text of the reference is given in the element content, the URI of the referenced web page is specified in the attribute "`href`".

Attributes (2)

Start-Tag:

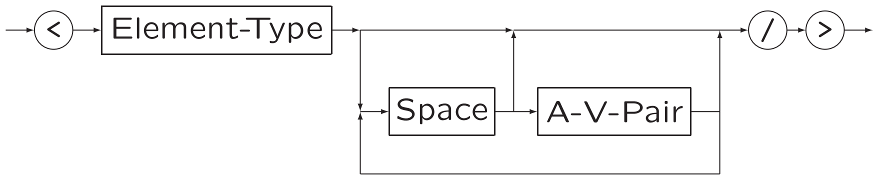


End-Tag:



Attributes (3)

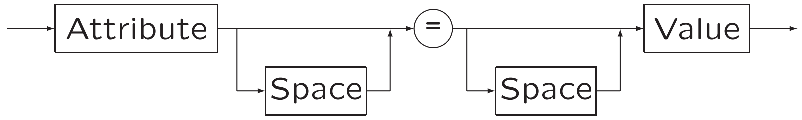
Empty Element Tag:



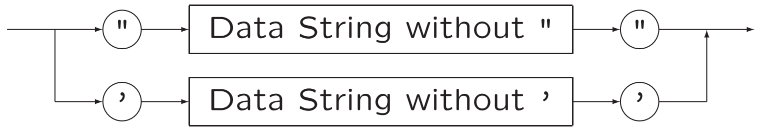
- "Space" (white space) consists of one or more space characters, carriage returns, line feeds, and tabs (ASCII 32, 13, 10, 9).

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: "<" ist 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. Unicode character numbers have usually 16 bit, and extensions are planned. 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.

Character References (3)

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

  `ä`

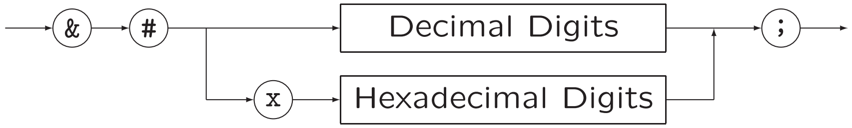
  is an "ä". Hexadecimal notation can also be used:

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

  ISO Latin 1 codes can be used since Unicode is upward compatible to ISO Lation 1.

Character References (4)

Character Reference:



- In DTDs, abbreviations/macros (“entities”) can be defined (see below).
- 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.

Character References (5)

- 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 “`"`”.

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/]
  - ◇ [http://www.validome.org/xml/]

Overview

1. Introduction
2. XML Documents (Syntax)
3. Document Type Definitions (DTDs)
4. DOCTYPE, XML Declaration
5. Entities, Notations, Marked Sections

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|ul)*)>
<!ELEMENT em    (#PCDATA)>
<!ELEMENT ul    (li+)>
<!ELEMENT li    ((#PCDATA|p|em|ul)*)>
```


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 is 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". The text is syntactically analyzed in order to check that it does not contain tags and in order to resolve entity and character references. In SGML (but not in XML) there is also "`CDATA`", which is not syntactically analyzed (like "`verbatim`" in LaTeX). The use of `#PCDATA` in model groups is very restricted in XML, see below.

Content Specifications (2)

- 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 (3)

- Content specifications can be connected with
  - ◇ (A | B): "A or B".

    The content must match A or B.
  - ◇ (A , B): "First A, then B" ("A followed by B").

    A prefix of the content must match A, the rest B.
- In SGML, there is also (not supported in XML):
  - ◇ (A & B): "A and B".

    A and B must both appear, but in arbitrary sequence. This is equivalent to ((A,B) | (B,A)). Therefore, & is not strictly necessary. But rewriting an "and" with many components in this way becomes clumsy. There are also restrictions because of the deterministic parsing requirement, see below.

Content Specifications (4)

- Model groups consisting of more than two components are also possible:
  - ◇ (A_1|...|A_n): "Alternative" / "Choice" (one of the A_i).
  - ◇ (A_1,...,A_n): "Sequence" (all A_i in the given sequence).
- The A_i are
  - ◇ An element type (possibly with ?/*/+).
  - ◇ `#PCDATA` (in XML with restrictions, see below).
  - ◇ A nested model group (possibly with ?, *, +).

Content Specifications (5)

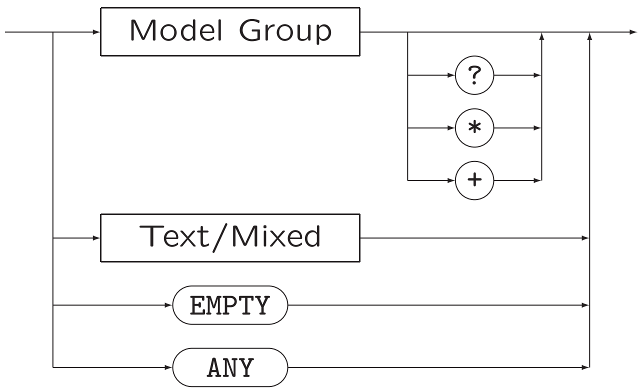
- A content specification ("content model") is
  - ◇ A model group (possibly only of one element),

    Element types must always be specified within parentheses. XML has special restrictions for mixed content, see below.
  - ◇ the keyword `EMPTY`: No content permitted.
  - ◇ The keyword `ANY`: Character data and elements of arbitrary type.
- SGML has also `CDATA` and `RCDATA`.

  For `CDATA`, even the special characters `<`, `>`, `&` can be used. They are not interpreted. `RCDATA` is the same, but `&` is interpreted (i.e. one can use character/entity references). In XML, one can use a CDATA-Section (see below) to include arbitrary text data in an XML document.

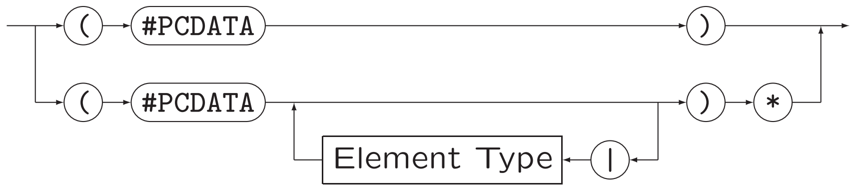
Content Specifications (6)

Content:



Content Specifications (7)

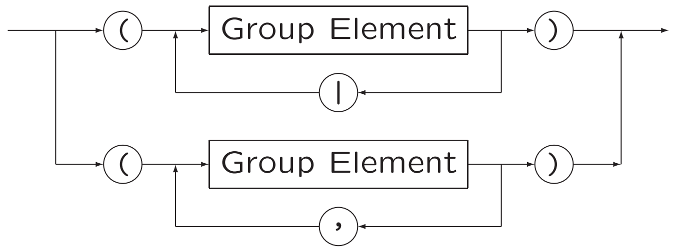
Text/Mixed:



- In XML, the only content models that can contain `#PCDATA` are (SGML has no such restriction):
  - ◇ `(#PCDATA)`
  - ◇ `(#PCDATA | Element-Type | ... | Element-Type)*`

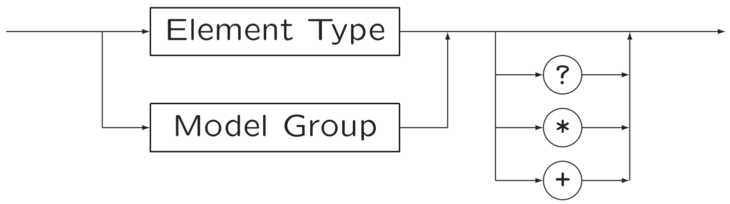
Content Specifications (8)

Model Group:



Content Specifications (9)

Group Element:



Content Specifications (10)

- 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.

Content Specifications (11)

- In SGML and in XML, content models must be not ambiguous. E.g. the following is forbidden:

  ```
  <!ELEMENT E ((A, B?), B)>
  ```

  When the parser has read an A and sees a B, it is not clear whether this is the optional B in the middle or already the required B at the end.

  The parser could solve this problem by looking ahead to see whether after the B in question there is another B. However, the SGML standard explicitly states: "an element or character string that occurs in the document instance must be able to satisfy only one primitive content token [in the content model] without looking ahead in the document instance." A primitive content token is an element type or #PCDATA.

Content Specifications (12)

- Another example for an ambiguous content model:

  ```
  <!ELEMENT E ((A, B) | (A, C))>
  ```

  When the parser sees the element A, it does not know which path to follow in the content model.
- This requirement simplifies the task of checking the input with respect to a given DTD.

  There are standard techniques for generating a nondeterministic finite automaton for a given regular expression. Normally, one would need to translate this into a deterministic automaton, which can lead to an exponential increase in the number of states. SGML and XML are restricted in such a way that the constructed automaton is already deterministic.

Attribute Declarations (1)

- Example (symbol used for marking list items):

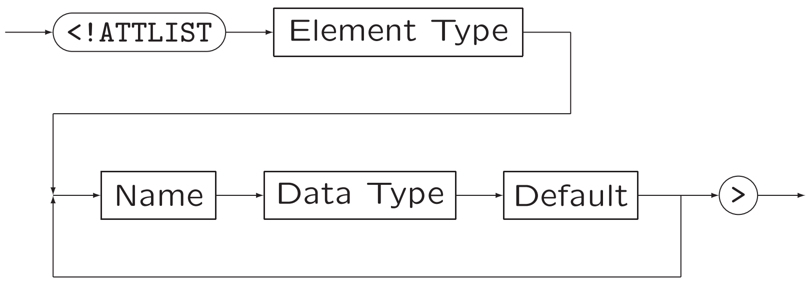
```
<!ATTLIST UL type (disc|square|circle) #IMPLIED>
```

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 Declarations (2)



- For each attribute, three things are defined: Name, data type, and default value.

White space is required between each two components of the `ATTLIST` command, except before the final “>”, where it is optional.

Attribute Declarations (3)

- The same attribute name can appear in an `ATTLIST` declaration only once.

This is clear: There cannot be conflicting definitions for an attribute in the same declaration.

- If there are several `ATTLIST` declarations for the same element type, they are merged. The first declaration for an attribute becomes effective, all other declarations for the same attribute are ignored.

This might be useful if a DTD is constructed in several pieces. It is however recommended (required in SGML?) that for every element type, there is only one `ATTLIST` declaration which defines all its attributes.

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 ":".

- NMTOKENS: List of NMTOKEN-values.
- ENTITY: Name of an entity.

  Entities are a kind of macros or include files (see below). An attribute of type ENTITY takes as value the name of a declared unparsed entity.

- ENTITIES: List of ENTITY-values.
- NOTATION (N_1|...|N_m): One of the notations N_i.

  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.

Attribute Data Types (4)

- In summary, XML supports the following attribute data types:
  - ◇ Enumeration types,
  - ◇ CDATA,
  - ◇ ID, IDREF, IDREFS,
  - ◇ NMTOKEN, NMTOKENS,
  - ◇ ENTITY, ENTITIES,
  - ◇ enumerations of notations.
- SGML has in addition NAME, NAMES, NUMBER, NUMBERS, NUTOKEN, NUTOKENS. E.g. NUMBER: sequence of digits.

Default Values (1)

- One must specify what should happen if an element of the type has not defined a value for the attribute.
- One possibility is to specify a default value:

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

  The quotation marks around the default value are not required in SGML, but they are required in XML. This is a bit inconsistent, since in accordance with SGML, there are no quotation marks in the enumeration of possible values. In SGML, attribute values that are NMTOKENS do not need quotes.

- Then the tag <UL> in the document is equivalent to

  <UL type="disc">.

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 #REQUIRED>
    <!ELEMENT EXERCISE #PCDATA>
    <!ATTLIST EXERCISE ID ID #REQUIRED>
]> <!-- continued on next slide -->
```


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'>Relational Algebra</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>
```

Overview

1. Introduction
2. XML Documents (Syntax)
3. Document Type Definitions (DTDs)
4. DOCTYPE, XML Declaration
5. Entities, Notations, Marked Sections

Well-Formed vs. Valid (1)

- In XML, the document type definition is optional.

  In SGML, a DTD is required for every document. An SGML document can normally not be parsed without knowing the DTD because of markup minimization (optional start and end tags).

- There are two classes of XML documents:
  - ◇ Well-formed documents satisfy the general rules of the XML syntax (e.g. that tags must be properly nested).
  - ◇ Well-formed documents may in addition be valid if they have an associated DTD and satisfy the syntax rules of this DTD.

Well-Formed vs. Valid (2)

- Checking the syntax of a document with respect to a DTD is called "to validate" the document.
- Even if there is a DTD, not every XML processor is required to read it and to validate the document.

  Correspondingly, the XML specification distinguishes "validating" and "non-validating XML processors".

- Web browsers (e.g. Internet Explorer, Firefox) typically do not validate the displayed XML documents.

  However, they do report an error if already the well-formed syntax rules are not satisfied.

System/Public Identifiers (1)

- In SGML/XML DTDs and other objects (e.g. entities, notations, see below) 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 (3)

- An advantage of public identifiers even in the Web age is that the contents of the URI does not have to be retrieved if there is a local copy and the public identifier is mapped to that copy.

  Otherwise one (probably) must retrieve the DTD via the URI each time a document is validated against that URI (there is no guarantee that the DTD stored under the URI does not change).

- In XML, a public ID can only be used in combination with a system ID. Thus, if an XML system does not know the public identifier, it can use the URI.

  SGML permits to specify only a public identifier.

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.

System/Public Identifiers (5)

- 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.

  There are further optional parts, see The SGML Handbook, page 385.

- Public text classes are, e.g., `DTD` and `NOTATION`.

DOCTYPE Declaration (1)

- One usually refers at the beginning of the document to the corresponding DTD:

```
<?xml version="1.0"?>
<!DOCTYPE EMAIL SYSTEM "mail.dtd">
<EMAIL>
    ...
</EMAIL>
```

- The file "mail.dtd" contains the declaration of elements, attributes, and entities as described above.

```
<!ELEMENT EMAIL (TO, FROM, DATE, SUBJECT?,
                 CONTENTS)>
...
```


DOCTYPE Declaration (2)

- One can also specify public and system identifier:

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

- The name of the DTD must always be identical to the name of the outermost element (document element, root of the element tree).

  The DTD itself does not specify what is the root element.

DOCTYPE Declaration (3)

- It is possible to declare the DTD in the document itself:

```
<!DOCTYPE EMAIL [
    <!ELEMENT EMAIL ...>
    ...
]>
<EMAIL> ... </EMAIL>
```

- Also a mixture of both is possible:

```
<!DOCTYPE EMAIL SYSTEM "mail.dtd" [...]>
```


DOCTYPE Declaration (4)

- The part in the document itself ("[...], "internal DTD subset") is processed before the DTD file ("external subset").

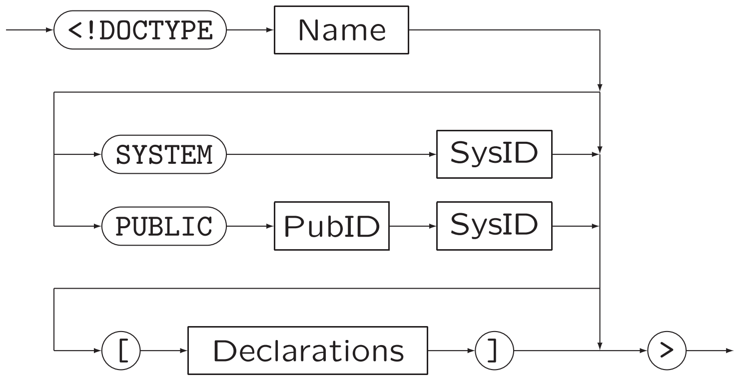
  The same entity (a kind of macro, see below) can be declared several times. Then the first declaration counts, all following declarations are ignored. In this way, the external subset can declare a default value for the entity, which can be overridden in the document.

- In XML, the constructs used in the internal subset of the DTD are somewhat restricted, such that a non-validating XML processor can easily skip it.

  See parameter entity references, marked sections.

DOCTYPE Declaration (5)

DOCTYPE Declaration:



Processing Instructions (1)

- Processing instructions are instructions for the application program that processes the XML/SGML data.
- E.g. they were sometimes used to force a page break at a specific point, but this of course contradicts the idea of rule-based markup.

  At least, as a processing declaration this is clearly separated from the main contents of the file and refers only to the printing program, not to other programs that process the same data.
- Processing instructions can contain any text, and are system- and application-dependent.

Processing Instructions (2)

- Processing instructions start with “<?” and end with “?>”.
- Processing instructions must start with a name that is the “target” for this instruction.

  In this way, on can have processing instructions for different applications in the file. Applications should ignore processing instructions that are not intended for them. In SGML, a processing instruction can be any string, but processing instructions must normally be exchanged when the file is processed with a different application. In SGML, processing declarations by default end with “>”, not “?>”. But of course, SGML is so parameterized that the XML end marker can also be selected.

Processing Instructions (3)

- It is suggested (but not required) to use a notation declaration for the target.
- The special target “xml” (in any capitalization) is reserved (see XML Declaration below).
- One can e.g. use the attribute-value syntax in a processing instruction, but this is not required.
- Processing instructions can appear more or less anywhere in the document (in the same places as comments).

XML Declaration (1)

- XML documents should start with an XML declaration that specifies at least the XML version:

  `<?xml version="1.0"?>`

- Version "`1.0`" is still the most widely used version, but there is now also a version "1.1".

  There are new editions of the W3C recommendation for XML 1.0. but they only clarify/correct a few points. The W3C recommendation for XML 1.0 was published on February 10, 1998. The second edition was published on October 6, 2000. The third edition of XML 1.0 was published on February 4, 2004, together with the first edition of XML 1.1. The current, fourth edition of XML 1.0 was published together with the second edition of XML 1.1 on August 16, 2006, both were edited in place on September 29, 2006. [`http://www.w3.org/XML/`].

XML Declaration (2)

- The changes from version 1.0 to 1.1 are small:
  - ◇ More characters are allowed in names.

    In XML 1.0, the valid characters in names were specified. In XML 1.1, the forbidden characters are specified (and characters are forbidden only if there is a specific reason). This makes a difference because the Unicode standard is developed further and some new languages were discriminated by the old XML standard.
  - ◇ Line ends in IBM mainframes are now permitted.
  - ◇ The rules for control characters change a bit.

    Character references to control characters in the range `x01` to `x1F` are now permitted, control characters in the range `x7F` to x9F (except whitespace) must now be written as character references.
  - ◇ Normalization rules permit binary comparison.

XML Declaration (3)

- For SGML processors, the XML declaration is simply a processing instruction.
- The XML declaration is optional, but it can be only the first command in an XML document.

  Even comments and white space is not allowed in front of it.
- The reason for this is that it can help to automatically detect the encoding used in the file.

  XML processors must at least be able to read at least the UTF-8 and UTF-16 encodings of Unicode. UTF-16 encoded files must start with the "Byte Order Mark" (`#xFeFF`).

XML Declaration (4)

- If one uses a different encoding (not Unicode), the XML declaration at the begin of the document is required, and must specify the encoding:

  `<?xml version="1.0" encoding="ISO-8859-1"?>`

- Also external parsed entities may begin with an XML declaration.

  There it is officially called "text declaration", because in external parsed entities the encoding part is mandatory (otherwise one would not use it), while the XML version is optional. For the XML declaration at the begin of the document entity, the version is mandatory and the encoding part is optional. Also the standalone declaration below is only permitted at the beginning of the document entity.

XML Declaration (5)

- The XML declaration can also specify whether markup declarations that are not contained in the same file (entity) may influence the information returned from the parser to the application program.

```
<?xml version="1.0" encoding="ISO-8859-1"
                     standalone="yes"?>
```

- The default is “no” (if there are external markup declarations), and this is normally correct.

  E.g. default values for attributes, entities used in the document, even element types with element content where white space is inserted in the document would all require “no”.

Summary: XML Document

- In summary, an XML document consists of:
  - ◇ An XML declaration (optional, recommended).
  - ◇ Comments, processing instructions, white space (optional).
  - ◇ A document type declaration (optional).
  - ◇ Comments etc. (optional).
  - ◇ An element (the document element, required).
  - ◇ Comments etc. (optional).

Overview

1. Introduction

2. XML Documents (Syntax)

3. Document Type Definitions (DTDs)

4. DOCTYPE, XML Declaration

5. Entities, Notations, Marked Sections

Entities: Overview (1)

- Entities can be used as macros (abbreviations), e.g. one can declare an entity “`ora`” with the value “`Oracle 8.1.6`” (replacement text):

```
<!ENTITY ora "Oracle 8.1.6">
```

- When the entity is declared, the entity reference

```
&ora;
```

  in the document is replaced by “`Oracle 8.1.6`”.

  In SGML, the “;” is optional if a character follows that cannot be part of the entity name, e.g. a space. In XML, the “;” is always required.

Entities: Overview (2)

- There are different kinds of entities. The above example is a general, internal, parsed entity.
- Entities can be classified as:
  - ◇ General: Used in the document.
    Parameter: Used in the DTD.
  - ◇ Internal: The value is written in the declaration.
    External: The value is contained in another file.
  - ◇ Parsed: The value is SGML/XML text.
    Unparsed: The value is e.g. binary data.

    In SGML, parsed entities are also called SGML entities, other entities are called Non-SGML or data entities.

Entities: Overview (3)

- Of the eight theoretically possible combinations, only five are permitted: Unparsed entities must always be external and general.

  Non-SGML/XML data cannot be directly included in an SGML/XML document and can certainly not be used in the DTD.
- In the SGML/XML literature, entities are seen as the physical units (storage units) of a document.

  I.e. entities are a generalization of files (e.g. they could also be extracted from a database or be computed by a program). Entities are containers for SGML/XML and other data. The main file, where the SGML/XML processing starts, is called the “document entity”. In contrast, elements are seen as the logical units of a document.

Entities: Motivation

- Entities reduce the typing effort (abbreviations).
- The entity name might be easier to remember than its replacement text (e.g. `ä` stands for `ä`).
- Using entities permits simpler updates and leads to higher uniformity.

  If in the above example, the Oracle version changes, one must change only the replacement text in the entity definition (at one place).
- One can also get several versions of a document via differently defined entities.

  E.g. if user interfaces are specified in XML, the language-dependent parts can be defined in entities.

General Entities: Details (1)

- General parsed entities can be referenced in:
  - ◇ the content of an element,
  - ◇ attribute value literals (inside quotes),

    This includes default values of attributes defined in the DTD.
    In XML, only internal (general parsed) entities are allowed in attribute values. It seems that SGML does not have this restriction.
  - ◇ the entity value in the definition of an entity.
- E.g. entity references cannot be used instead of an element type or attribute name within a tag.

  As with whitespace, the SGML/XML grammar specifies where entity references can appear. They are not expanded in System/Public IDs.

General Entities: Details (2)

- SGML/XML try to exclude unexpected parsing mode changes after an entity is referenced.
- This is especially important for XML, because XML can be parsed without DTD.

  Then the replacement text for entities might not be known, but still the general structure of the document should be clear.
- The opening delimiter of a tag, comment, etc. must be in the same entity as the closing delimiter.

  I.e. the replacement text of an entity that is referenced in the content cannot contain an unmatched "<" or ">". If the entity is referenced in an attribute value, these characters have no special meaning.

General Entities: Details (3)

- If an entity reference appears in an attribute value, the delimiters (quotes) `"` and `'` are not interpreted in the replacement text.

  I.e. it is not possible that an entity reference in an attribute value suddenly closes the attribute value.
- XML requires also that if the start tag of an element is contained in an entity, the corresponding end tag must be contained in the same entity.

  In SGML, this is only a recommendation, except for elements with content model "`CDATA`" and "`RCDATA`", where it is required.

General Entities: Details (4)

- Entities can be used in the definition of other entities:

```
<!ENTITY A "xxx">
<!ENTITY B "yyy &A; zzz">
```

- When the entity declaration is processed, the replacement text is simply stored under the name of the entity (including entity references within it).
- Only when "`&B;`" is called later in the document, the replacement text is inserted, and recursively, any entity references within it are substituted.

General Entities: Details (5)

- Entities must be defined before they are used.
- However, because of the delayed ("lazy") processing of references, this rule would even be satisfied if the entities "`A`" and "`B`" would have been declared in the opposite order.

  Since general entities are declared in the DTD and normally only used in the document, "definition before use" is seldom a problem.
- When an entity is referenced in the default value for an attribute in an `ATTLIST` declaration, it is immediately evaluated (and must already be defined).

General Entities: Details (6)

- Of course, recursive definitions are forbidden:

```
<!ENTITY X "This is not allowed &X;">
```

- If the same entity is defined several times, the first definition counts.

  The “internal subset of the DTD” is processed before the part in external files. Then the external subset can contain a default value for the entity, which can be overridden in the internal subset.

General Entities: Details (7)

- The replacement of entities like “&” does not lead to the generation of new entity references.

  E.g. in “AT&T;”, after the first replacement, the parser does not recursively try to replace “&T;”. This follows the general rule that the replacement of entities does not lead to the generation of new structures.

- Character references are already replaced when the entity definition is processed (non-recursively).

  E.g. the entity “amp” is defined as follows: <!ENTITY amp "&">. When this definition is processed, “&” is replaced by “&”. When “&” is later used in the text, it is expanded to “&”, and this is replaced by the character “&” which now has no special meaning.

General Entities: Details (8)

- In XML, the following five entities are predefined:
  - “&” for “&” (ampersand).
  - “<” for “<” (less-than symbol).
  - “>” for “>” (greater-than symbol).
  - “'” for “ ’ ” (apostrophe).
  - “"” for “"” (quotation mark).
- In SGML, these are not predefined. Therefore, they should also be declared in XML.

External Entities

- Entities can also be used as an “include” mechanism for splitting a document into several files:

```
<!ENTITY copyr SYSTEM "copyr.xml">
```

- Then the entity reference “&copyr;” in the document is replaced by by the contents of the file “copyr.xml”.

  The keyword “SYSTEM” indicates that the following string gives a system-dependent way to retrieve the entity. In XML this must be a URI, possibly a relative one. There are also public identifiers (see below).

- This is a general, external, parsed entity.

Parameter Entities (1)

- General entities are used in the document (data).
- However, macros are also useful in the DTD.
- But macros applied in the DTD are not relevant for the user of the DTD, they might even confuse him/her.
- Therefore, two distinct namespaces are used:
  - ◇ General entities are substituted in the document.

    And in the default attribute value in the DTD. They can also be used in the declared value of other entities.
  - ◇ Parameter entities are substituted in the DTD.

Parameter Entities (2)

- The declaration of parameter entities contains an additional "%":

```
<!ENTITY % ltypes "(disc|square|circle)">
```

- Correspondingly, a parameter entity reference uses a percent sign "%" instead of the ampersand "&":

```
%ltypes;
```

- In the document itself, "%" has no special meaning.
- It is even possible to have a general entity and a parameter entity with the same name.

Parameter Entities (3)

- The replacement text of a parameter entity is extended by spaces at the beginning and the end.

  This makes sure that no tokens can merge when parameter entities are replaced.
- In XML, the use of parameter entities in the internal subset of the DTD is quite restricted: A parameter entity reference can only appear in places where an entire declaration would be permitted.

  I.e. there, parameter entities can contain only complete markup declarations. This restriction does not hold for the external subset.

Parameter Entities (4)

- In contrast to general entities (and like character references), parameter entities are immediately replaced, even if they are used in the definition of another entity.
- As for general entities, if a parameter entity replacement text contains the start of a markup declaration ("<"), it must also contain the corresponding end.

Parameter Entities (5)

- Of course, there are also external parameter entities:

```
<!ENTITY % tables SYSTEM "tab.xml">
```

- The contents of the file "`tab.xml`" is inserted in the DTD where the parameter entity is referenced:

```
%tables;
```

- A large DTD can be constructed in this way out of components stored in different files.

  Also the same component might be reused for different DTDs.

Notations (1)

- An SGML/XML-system can also manage entities (files) that do not contain SGML/XML text.
- E.g. a document often includes pictures in formats like GIF, JPG, PNG, TIFF.
- One can define in SGML/XML that e.g. GIF is a name for a notation (data format).
- Then one can define external entities that use the notation "GIF" (and are therefore not syntactically analyzed).

Notations (2)

- A notation can be declared with a system identifier, which might e.g. refer to a program that could display the data ("helper application"):

```
<!NOTATION GIF SYSTEM "file:///local/bin/xv">
```

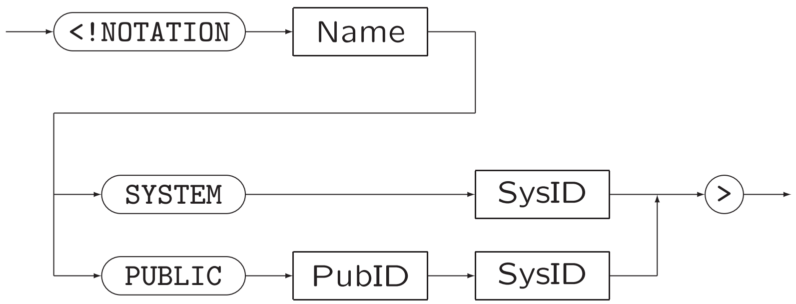
  However, the SGML/XML parser only passes the system identifier to the application program. It depends on this program, how it uses this information.

- In XML, system identifiers must be URIs (without `#`), enclosed in single or double quotes.

  The standard only says "It is meant to be converted to a URI reference ...". Entity references are not evaluated. The URI can possibly be relative to the location of the entity that contains the declaration.

Notations (3)

Notation Declaration:



Public Identifiers (1)

- Normally, public identifiers refer in some way to further information:

```
<!NOTATION POSTSCRIPT PUBLIC
    "+//ISBN 0-201-18127-4::Adobe//NOTATION
        Postscript Language Ref. Manual//EN">
```

- There is no well-known and generally accepted list of public identifiers for notations.

  But see below for examples. There seems to be no central registry for public identifiers.

- However, e.g. HTML versions have public identifiers mentioned in the respective standards.

Public Identifiers (2)

- In the XML Bible [p. 309] the following public identifier is used for GIF (this method could be generalized to arbitrary MIME types):

```
"-//IETF//NONSGML Media Type image/gif//EN"
```

- In my view, the keyword "`NONSGML`" is wrong and must be replaced by "`NOTATION`".

  "`NONSGML`" means a non-SGML data entity.

- The XML Bible uses the following system identifier:

```
"http://www.isi.edu/in-notes/iana/assignments/
                    media-types/image/gif"
```


Public Identifiers (3)

- Public identifiers used in the DocBOOK DTD:
  - BMP: `"+//ISBN 0-7923-9432-1::Graphic Notation//NOTATION Microsoft Windows bitmap//EN"`
  - EPS: `"+//ISBN 0-201-18127-4::Adobe//NOTATION PostScript Language Ref. Manual//EN"`
  - GIF87a: `"-//CompuServe//NOTATION Graphics Interchange Format 87a//EN"`
  - GIF89a: `"-//CompuServe//NOTATION Graphics Interchange Format 89a//EN"`
  - TeX: `"+//ISBN 0-201-13448-9::Knuth//NOTATION The TeXbook//EN"`
  - WMF: `"+//ISBN 0-7923-9432-1::Graphic Notation//NOTATION Microsoft Windows Metafile//EN"`
  - SGML: `"ISO 8879:1986//NOTATION Standard Generalized Markup Language//EN"`
  - FAX: `"-//USA-DOD//NOTATION CCITT Group 4 Facsimile Type 1 Untiled Raster//EN"`
  - CGM-CHAR: `"ISO 8632/2//NOTATION Character encoding//EN"`
  - CGM-BINARY: `"ISO 8632/3//NOTATION Binary encoding//EN"`
  - CGM-CLEAR: `"ISO 8632/4//NOTATION Clear text encoding//EN"`
  - PNG: `"http://www.w3.org/TR/REC-png"`

Unparsed Entities (1)

- An unparsed entity is declared in the following way:

```
<!ENTITY clown SYSTEM "clown.gif" NDATA GIF>
```

- The keyword "`NDATA`" ("Non-SGML Data") must always be followed by a declared notation name.

  SGML has also the keywords "`CDATA`" and "`SDATA`" which are, however, not supported in XML.

- In this way, the SGML/XML system "knows" the data format (media type) of each entity and does not have to guess it from file extensions etc.

Unparsed Entities (2)

- Unparsed entities cannot be used in entity references, but one can declare element types that take entities as attributes:

```
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE SRC ENTITY #REQUIRED>
```

- Then one can write e.g.:

```
<IMAGE SRC="clown"/>
```

The SGML parser then makes system/public identifier (public IDs can normally be mapped to system IDs) of entity and notation available to the application program. The application program can then retrieve the data of the entity by means of the "entity manager" (the layer below the SGML parser, also part of an SGML system).

Unparsed Entities (3)

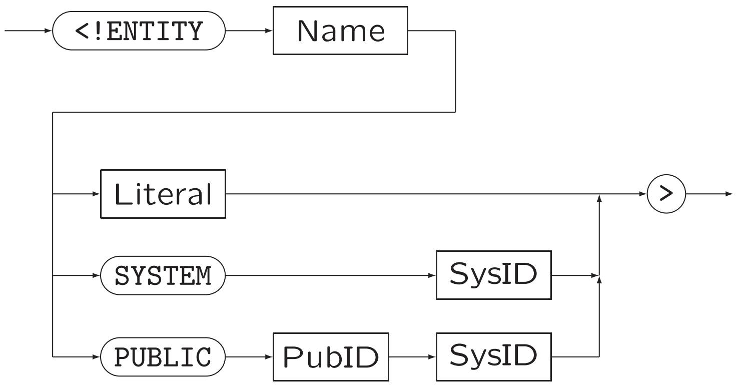
- Of course, parsed general entities can also be used as attribute values (not only unparsed entities).

  Only general entities can appear as attribute values (parameter entities have no meaning in the document itself, i.e. the data).

- One cannot restrict the possible notations for entities of an attribute in the attribute declaration.
- In HTML, one cannot define entities (the given DTD cannot be extended). Therefore, the element type "`IMG`" as an attribute of type "`CDATA`" which directly contains the URI of the image file.

Entity Declaration (1)

General Parsed Entity Declaration:



Entity Declaration (2)

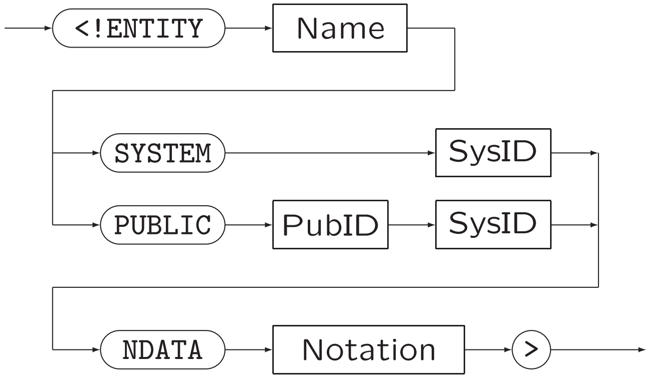
- "Literal" is a string enclosed in single or double quotes. (' or ").
- Parameter entity references and general entity references can be used in the literal.

  Parameter entity references are immediately evaluated, general entity references become part of the replacement text of the entity.

- Entity references are not evaluated in the system and the public identifier.

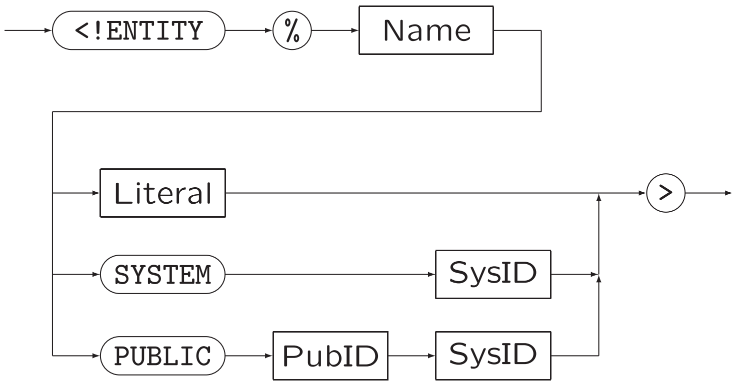
Entity Declaration (3)

Unparsed Entity Declaration:



Entity Declaration (4)

Parameter Entity Declaration:



Marked Sections (1)

- The contents of an IGNORE-section is not processed:

```
<![IGNORE[...]]>
```

- In contrast, the contents of an INCLUDE-section is processed normally:

```
<![INCLUDE[...]]>
```

- One can define an entity which has one of the two values “IGNORE” and “INCLUDE” to get a feature similar to “conditional compilation”, e.g.

```
<!ENTITY % solution "INCLUDE">
```


Marked Sections (2)

- Then one can mark sections that are to be included only in certain versions of the document, e.g. solutions are printed only in the edition for the teacher:

```
<![%solution;[...]]>
```

- In XML, conditional marked sections can only be used in the external subset of the DTD and must contain a sequence of complete markup declarations (not arbitrary text).

  In SGML, marked sections can appear in the DTD and in content (the body of the document).

- Conditional marked sections can be nested.

Marked Sections (3)

- Besides these conditional sections, there are also "verbatim" sections, in which markup is not evaluated.
- `CDATA`-sections can contain the characters "<", ">" and "&" as normal text. They are not interpreted as markup:

```
<![CDATA[...]]>
```

- Of course, `CDATA` sections can only be used in the document body, not in the DTD.

Marked Sections (4)

- `CDATA` sections cannot nest.

  The only markup that is interpreted within a `CDATA` section is its end delimiter "]]>". The parser would not even notice the begin of another such section.

- `CDATA` sections are normally used for showing example XML/HTML code, which should not be interpreted as markup.

  The alternative would be to escape the special characters "<" and "&" one by one with entity or character references. Of course, within a `CDATA` section, entity and character references are also not understood.