Chapter 5: Introduction to SGML and XML

References:

- Liora Alschuler: ABCD . . . SGML — A User's Guide to Structured Information. International Thomson Computer Press (ITP), 1995, ISBN 1-850-32197-3, 414 pages.
- Charles F. Goldfarb, Yuri Rubinsky: The SGML Handbook. Clarendon Press, 1990.
- Henning Lobin: Informationsmodellierung in XML und SGML. Springer-Verlag, 1999.
- C. M. Sperberg-McQueen and Lou Burnard (Eds.): A Gentle Introduction to SGML. [http://www-tei.uic.edu/orgs/tei/sgml/teip3sg/index.html]
- On SGML and HTML (in the HTML 4.01 Specification). [http://www.w3.org/TR/html401/intro/sgmltut.html]
- Yuri Rubinsky, SoftQuad: The SGML Primer. [http://www.softquad.com/top_frame.sq?page=resources/content_sgml_primer.html]
- Martin Bryan (The SGML Centre): An Introduction to the Standard Generalized Markup Language (SGML). [http://www.personal.u-net.com/~sgml/sgml.htm]
- Harvey Bingham: SGML Syntax Summary. [http://www.oasis-open.org/cover/sgmlsyn/contents.htm]
- Charles F. Goldfarb's SGML Source Home Page: [http://www.sgmlsource.com/]
- 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].

Objectives

After completing this chapter, you should be able to:

- explain the relationship between SGML, XML, and HTML.
- enumerate possible applications of SGML/XML.
- write syntactically correct SGML/XML documents.

  You should know more or less all of XML, and SGML only as far as it is used in HTML.
- read the DTD syntax, e.g. in the HTML specification.

Overview

SGML (1)

- SGML: "Standard Generalized Markup Language".
- HTML is an application of SGML.
- SGMLs two levels:
  - ◇ SGML is a syntax formalism, in which HTML and similar markup languages can be defined.

    Syntax definition (grammar): DTD (Document Type Definition).
  - ◇ For a given DTD, SGML documents contain the data or the text.

    HTML documents are SGML documents for a specific DTD. The most important part of the HTML specification is a DTD for HTML in the SGML syntax.

SGML (2)

HTML-Document (Example of an SGML Document):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
    <HEAD>
        <TITLE>My first HTML document</TITLE>
    </HEAD>
    <BODY>
        <P>Hello, world!
    </BODY>
</HTML>
```


SGML (3)

Another SGML Document:

```
<!DOCTYPE EMAIL SYSTEM "/home/brass/mail.dtd">
<EMAIL>
    <TO>Stefan.Brass@informatik.uni-halle.de
    <FROM>sbrass@sis.pitt.edu
    <DATE>Mon, 31 Mar 2003 15:23:04 -0500 (EST)
    <SUBJECT>Test
    <CONTENTS>Does my new email address work?
        Und <URL>http://www.stefan-brass.de</URL>?
        Viele Gruesse!
    </CONTENTS>
</EMAIL>
```


SGML (4)

- A DTD defines for a class of documents which tags (elements) can be used, how they can be nested, which attributes they have, etc.

  A DTD is a syntax formalism that is similar to context free grammars, only specialized, because the general SGML syntax is given (although there are many parameters defined in an SGML declaration).

- SGML is an ISO-Standard (Nr. 8879 von 1986).

  It was developed by Charles F. Goldfarb and others.

- XML, the foundation of the future "semantic web", is basically a subset (simplification) of SGML.

  The browser vendors considered the full SGML as too complicated.

SGML (5)

- SGML is only a data format (syntax).
- It says nothing about the semantics of the data that are coded in SGML.

  E.g. although one can declare tags like "`<BOLDFACE>`", one cannot define in SGML that the enclosed text should be printed boldface. This is done by means of stylesheets in languages like DSSSL (Document Style Semantics and Specification Language) or XSL/XSLT.

- SGML can be easily translated into other data formats.
- SGML is especially important for the exchange of data/documents, e.g. between companies.

SGML (6)

- An SGML application consists of four parts:
  - An SGML Declaration: It defines e.g. the character set (and further parameters, see below).
  - A DTD (defines Tags/Elements etc.).
  - A specification of the semantics of the tags.

    It can also contain further syntax constrains which cannot be expressed in the DTD.
  - Documents that satisfy the rules in SGML declaration and DTD and contain the real data.

    Each document refers to its DTD.

Markup Languages

- In earlier times, a manuscript for a book was written on a typewriter and instructions for the typesetter were added by hand.

  Later these marks of text parts were replaced by commands for typesetting programs, e.g. `{\it ...}` in TEX/LATEX.
- Bright yellow text markers: important text pieces.
- Charles Goldfarb invented the term "Markup Language" in 1969, in order to match the shorthand GML ("Generalized Markup Language") with its designers Goldfarb, Mosher and Lorie.

Appearance-based Markup (1)

- First, the "markup" of the text were only typesetting instructions, e.g. 18pt, italics, indented.
- This works as long as the only use of the text is to print it in exactly this format.
- Other ways to use texts:
  - Spelling checker: E.g. names must be marked.
  - Automatic generation of a table of contents.
  - Search and replace.
  - Printing in other formats, on other media.

Appearance-based Markup (2)

- E.g. italics is used for emphasizing as well as for names: The distinction is important for the spell checker.
- E.g. large and boldface is used for chapter headlines, but also for special warnings in the text: The distinction is important for generating a table of contents.
- E.g. searching a letter about "John Smith's will" (in the sense of "last will"): will is a common word.

Appearance-based Markup (3)

- WYSIWYG ("What you see is what you get"): "What you see is all you've got" (Brian Kernighan).
- Appearance-based (or presentation-oriented) markup is also called physical or procedural markup.
- The opposite is structure-based markup, which is also called contents-oriented, logical or descriptive markup.

Structure-based Markup (1)

- Structure-based markup normally contains more information than appearance-based markup because it distinguishes text components that will later be printed in the same way.
- The information how a certain text part (e.g. a chapter headline) is printed is not defined in the text itself, but in a "style sheet" (or "stylesheet").
- At the beginning, style sheets were simply macro definitions. E.g. the command "chapter headline" was replaced by the commands "18pt", "boldface".

Structure-based Markup (2)

- XSLT style sheets for XML contain recursive rules that can describe very general tree transformations.
- The distinction between appearance-oriented and structure-oriented markup was first realised in Brian Reid's SCRIBE-System (1978).
- The formatting of a text for printing it is called "Rendering" (like an artist renders a music piece).
- One can have several style sheets for the same input text which define different output formats.

Structure-based Markup (3)

- "Write once, use everywhere."
- When one writes a text (enters it into a computer), one has to think about what one wants to do with the data in future (how it will be processed).
- Experiments have shown that authors use a relatively large part of their valuable time for the outward appearance of their texts (about 30%).

  "And after 18 months, one gets new hard- and software, and everything starts again from the beginning."

Structure-based Markup (4)

- With pure structure-based markup, one has less influence on the appearance of the printed text (one can only specify general rules in the style sheets).
- With structure-based markup better control of the text structure is possible (data integrity). Greater uniformity can be enforced.

  SGML DTDs can describe documents that basically correspond to relations as in databases (strongly structured data). The entire range to unstructured text is covered. Semistructured data, which lies between classical database data and text, is a current research topic. An example would be e.g. BibTeX data.

HTML (1)

- HTML intermixes structure-based and appearance-based markup.
- E.g. HTML has `CODE`, `KBD`, `SAMP` for program code, keyboard input, and program output.

  This is structure-based markup. Netscape prints them all as `TT`.
- But HTML also permits direct font selection (`TT`).

  This is appearance-based markup.
- First HTML had mainly structure-based markup, but then browser vendors added many appearance-based tags.

HTML (2)

- The HTML version "HTML 4.01 strict" tries to remove the appearance-based markup from HTML and to replace it by style sheets.
- Obviously, a single set of tags cannot be sufficient for all different kinds of documents in the web if one wants powerful structure-based markup.
- E.g. tags like "`PRODUCT`" and "`PRICE`" would be useful to make online offers of e.g. books understandable for automatic price comparison engines.

Energy Metaphor (1)

- Formats, in which textual information is encoded, can be compared with different energy levels.

  [Liora Alschuler: ABCD ... SGML, ITP, 1995]
- To move from lower levels to higher levels (up-conversion) is difficult and requires energy.
- To move from higher levels to lower levels (down-conversion) is easy

  Although it does not really release energy.

Energy Metaphor (2)

- Representations of text, from lowest energy level to highest:
  - ◇ Hardcopy, Bitmap.

    More and more pictures are used for text on web pages, because the authors want special fonts or other effects. Search engines cannot read this text, also speech output for blind people cannot work with it. Up-conversion to ASCII text is possible, but it requires OCR software and manual post-processing.
  - ◇ Pure ASCII-text.

    Converting from ASCII to a bitmap/hardcopy is easy.
  - ◇ Text with appearance-based markup.
  - ◇ SGML, Structured information.

Independence Declaration (1)

- "SGML is the information provider's declaration of independence."

  [Liora Alschuler: ABCD ... SGML, ITP, 1995]
- Independence from a software vendor.

  SGML is an ISO standard, and there are SGML tools from quite a number of different vendors. XML is a W3C recommendation, and there are many free XML tools available (and, of course, also commercial ones).
- "Whoever owns the format, owns the information."

  The licence fee for the program that makes the data usable is like a lease payment for the information. One has to pay regularly for updates, or the information becomes unusable.

Independence Declaration (2)

- Operating-system/platform independent.

  SGML/XML tools are available for many different platforms, Microsoft Word is available only for Windows (and maybe the MacOS).
- Independent form the output (e.g. print or online).
- Independent from the application program.

  A Word file can basically only be processed with Word. The same SGML/XML-encoded data file can be input for many different application programs. It is also relatively easy to develop new such programs: Several libraries that parse XML are available as free source code. Furthermore, languages like XSLT make it simple to specify conversions.
- "Long-term conversion insurance." [Aluscher]

Historical Remarks (1)

- 1969: Goldfarb, Mosher and Lorie developed the Generalized Markup Language (GML) at IBM.

  They worked in a project for managing legal documents.
- 1974: Goldfarb developed a program for checking the syntactical correctness of documents with respect to a DTD.
- 1978–1986: Goldfarb was technical director of a committee that developed the ISO standard 8879 for SGML.

Historical Remarks (2)

- SGML became the de-facto standard for exchanging large, complex documents, e.g. maintainance manuals for airplanes or test documentation for new medical drugs.

  SGML is also the basis of many EDI systems ("Electronic Data Interchange" between companies, e.g. between a car manufacturer and its sub-suppliers.

- 1985: CALS (US Department of Defense): SGML-based standards for technical documentation.

Historical Remarks (3)

- 1989: T. Berners-Lee proposed a project at CERN which became the basis of the World Wide Web.

  Only the version he developed 1990 together with Robert Caillian was accepted.

- Anders Berglund (a collegue at CERN) suggested to use SGML, but HTML was first developed only by means of examples.

  When finally a formal DTD for HTML was defined, there were already thousands of incorrect HTML documents.

Historical Remarks (4)

- When the web was successful, the web pages were not only displayed in browsers, but also processed e.g. by search engines and price comparison services.
- Therefore, it would have been natural to use the complete SGML, but the browser vendors considered that as too complicated.

Historical Remarks (5)

- 1998: After long negotiations a (slightly modified) SGML-subset was selected, which was called XML (eXtensible Markup Language). The W3C working group was led by Jon Bosak (SUN).
- Today: Many XML-based standards are being developed: DOM, XPath, XSL, XLink, XML Schema, XHTML, etc. etc.

Overview

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

Elements (1)

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

  `<TITLE>My first HTML 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 HTML document TITLE

Elements (2)

- Special characters like "<" and ">" can be defined in the SGML declaration (they are not built into SGML, one can choose other characters).

  However, these characters are used in the "Reference Concrete Syntax", and they are used in HTML.

- XML is less parameterized than SGML (it has a fixed SGML declaration), and in particular, the characters "<" and ">" are built into XML and cannot be changed.

Elements (3)

- 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.

- Quite often, "tag" is used when "element" would be formally right.

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

Elements (4)

- Element types are declared in a DTD. E.g. the “HTML 4.01 strict” DTD declares a certain set of element types for HTML documents that includes e.g. “TITLE”.

  A DTD defines much more, see below.

- In the SGML declaration, one can define
  - ◇ which characters are permitted in names (identifiers), e.g. for element types, and
  - ◇ whether names are case-sensitive or not.

Elements (5)

- In HTML, names contain only letters and digits.

  However, the SGML declaration for HTML would permit letters, digits, hyphens “-”, periods “.”, and colons “:”. This is important, because if an attribute value is a valid name, one does not need quotes (see below).

- In HTML, names are not case-sensitive.
- In XML, names can contain letters, digits, periods “.”, hyphens “-”, underscores “_”, colons “:”.

  Plus certain extended characters from the Unicode set. They must start with a letter, an underscore “_”, or a colon “:”.

- In XML, names are case-sensitive.

Elements (6)

- The contents of an element is the text between start-tag and end-tag. E.g. the contents of the example element (Slide 5-30) is

  `My first HTML document`

- For each elements 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 HTML (one cannot nest any other elements inside).

Elements (7)

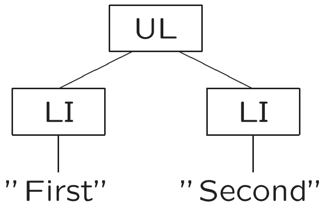
- The element type “`UL`” (unordered list) contains a sequence of elements of the type “`LI`” (list item):

```
<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 (8)

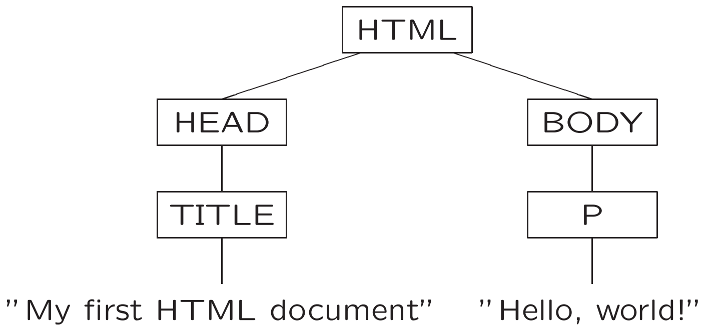
- E.g. the unordered list above has this structure:



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 (9)

- Structure of the HTML example (Slide 5-5):



Elements (10)

- 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 (11)

- 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.

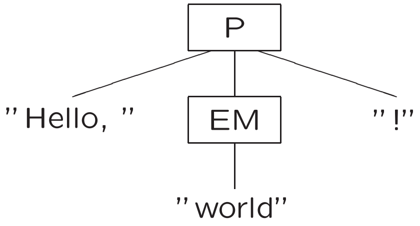
  - ◇ Element types that can contain a mixture of text and other elements. ("mixed content model"):

    ```
    <P>Hello, <EM>world</EM>!</P>
    ```

  - ◇ Element types that always have empty contents.

Elements (12)

- 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 HTML.

Tag Minimization (1)

- SGML has options for the tag minimization, which are set in HTML (`OMITTAG` and `SHORTTAG`).

  These options can be set in the SGML declaration.

- XML does not permit markup minimization.
- Some start- and end-tags, which can be reconstructed from the context, can be left out.
- E.g. "`</LI>`" is not required:

```
<UL><LI>First<LI>Second</UL>
```


Tag Minimization (2)

- The DTD specifies that `LI`-elements cannot be directly nested inside `LI`-elements.
- Therefore, when the second `LI`-tag is opened, it is clear that a closing `LI`-tag is missing.
- In an SGML DTD, one can define for each element type, whether the opening and/or closing is optional.

  Of course, one can specify that a tag is optional only if the tag can be uniquely reconstructed from the context.

Tag Minimization (3)

- The HTML DTD specifies that the closing `LI`-tag is optional.
- Thus, when the parser sees the second opening `LI`-tag, it does not print an error message, but assumes that the first `LI`-element is closed directly in front of it (i.e. it reconstructs the omitted "`</LI>`" tag).
- In the same way, `P`-elements (paragraph) cannot be directly nested in HTML, and only the start tag is required (the closing tag is declared as optional).

Tag Minimization (4)

- Since every HTML document begins with "<HTML>" (after the document type declaration) and ends with "</HTML>", both tags can be omitted.

  The "HTML-element at the root of the tree is still there (it is reconstructed by the SGML parser).

- Within the HTML-element, there must be always first a HEAD-element and then a BODY-element.
- Again, both tags can be uniquely reconstructed, since the HEAD element can contain only elements like TITLE which cannot appear in the body.

Tag Minimization (5)

- The HTML DTD defines start- and end-tags of HTML, HEAD, and BODY as optional.
- Thus, the HTML example from Slide 5-5 can be shortened to:

```
<!DOCTYPE HTML PUBLIC
              "-//W3C//DTD HTML 4.01//EN">
<TITLE>My first HTML document</TITLE>
<P>Hello world!
```

- The parser reconstructs the missing tags. The tree representation on Slide 5-38 remains unchanged.

Tag Minimization (6)

- Tag minimization is only useful when SGML documents are written with a standard text editor.
- Tag minimization is not necessary when
  - ◇ SGML documents are written with special editors that know the SGML syntax,
  - ◇ SGML documents are generated by a program (e.g. for exporting data from a database).
- However, it leads to a significant complication of the parser. This is why it was excluded in XML.

Tag Minimization (7)

- Another reason, why tag minimization is not supported in XML, is that it should be possible to parse XML documents even without a DTD.

  E.g. simple XML parsers do not need to read and understand the DTD. But for reconstructing missing tags, the DTD is necessary.

Empty Elements

- In SGML, no end tag may be specified for elements that are declared as empty in the DTD (e.g. “BR”).

  It were anyway not useful, since it would always have to appear immediately after the opening tag. But in SGML, it is actually a syntax error to write “
</BR>”. If the element type is not declared as empty, but this specific element by chance has empty content, it is legal (and might even be required depending on the element type declaration).

- In XML, no tags can be omitted, but one can use the special syntax “
” for empty elements.

  This is equivalent to “
</BR>” and can be used no matter whether the type is declared as empty or not. Also, “
</BR>” is legal in XML, even if “BR” is declared as empty.

Line Ends (1)

- 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).
- E.g. the following is equivalent to “<P>Text</P>”:

```
<P>
Text
</P>
```

- Within the content of an element, line ends are not ignored (they are often treated like spaces, but that depends on the application program).

Line Ends (2)

- Also line ends are ignored after a line that contained only processing instructions or markup declarations (see below).
- Within tags or declarations, line ends are treated as spaces.
- In XML, line ends or empty space is not ignored.

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

- 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)

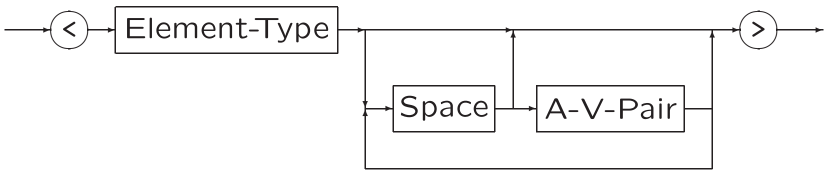
- In the start tag, attribute-value pairs can be optionally specified.
- E.g. in HTML, links to other documents are marked the the element A (“anchor”):

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

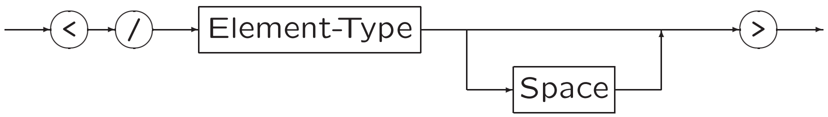
- The text of the reference is given in the element content, the URI of the referenced web page is specified in the attribute “HREF”.

Attributes (2)

Start-Tag:

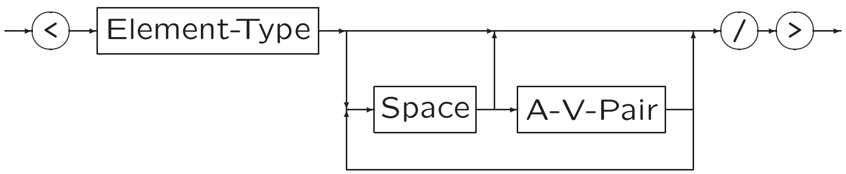


End-Tag:



Attributes (3)

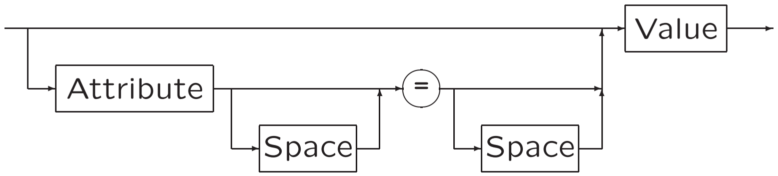
Empty Element Tag (XML only):



- "Space" (white space) consists of one or more space characters, carriage returns, line feeds, and tabs (ASCII 32, 13, 10, 9).
  - In XML and normally in SGML, but SGML is highly parameterized and one can select other characters there.

Attributes (4)

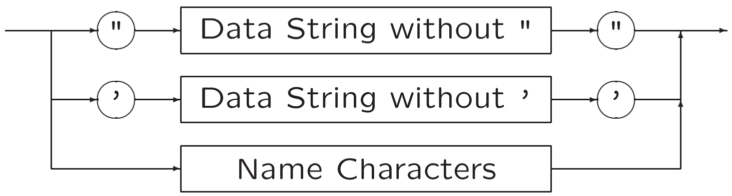
A-V-Pair (SGML):



- One can specify an attribute value without the attribute name only if the attribute is declared in the DTD as an enumeration type and if the markup minimization option "`SHORTTAG`" is selected.

Attributes (5)

Value (SGML):



- 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).

Attributes (6)

- If the attribute value consists only of letters, digits, hyphens, underscore, period, colon, no quotation marks are required in SGML.
- E.g. the element "UL" had an attribute "TYPE" in HTML 3.2, and one could write "<UL TYPE=DISC>" instead of "<UL TYPE="DISC">".

  The attribute defines the type of bullet to be used for the list items. It was removed in HTML 4.0 strict, because it is appearance-based. One should use style sheets for this task. However, the attribute is still contained in "HTML 4.0 transitional".
- XML always requires quotes.

Attributes (7)

- In HTML 3.2, the element type "DL" (description list) had an attribute "COMPACT".

  A description list can e.g. be used for a glossary. It lists terms (in the element "DT") and their definition (in the element "DD"). If "COMPACT" is selected and the terms are short, the definition starts on the same line. Without "COMPACT", the term is always printed on its own line. Since this is again a setting for the appearance, it was removed in HTML 4.0 strict. It is still contained in HTML 4.0 transitional.
- The attribute "COMPACT" has only the single possible value "COMPACT", but it can also be undefined (no value). Therefore, it is really a boolean attribute.

Attributes (8)

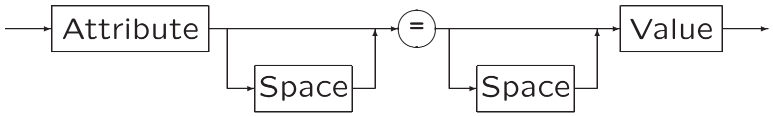
- In HTML, the option "SHORTTAG" is selected, and one can write "<DL COMPACT>" instead of

  <DL COMPACT="COMPACT">.
- Actually, the long form may not even be understood by all browsers.

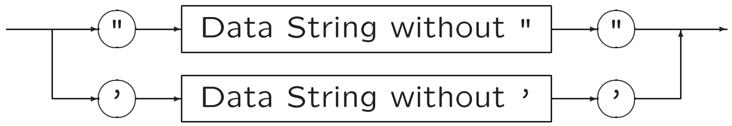
  Browsers do not contain a full SGML parser and therefore do not support seldom used features of SGML. Since the long form looks a bit strange, every few HTML authors used it.
- However, in XML, the attribute name is always required: One must use the long form.

Attributes (9)

A-V-Pair (XML):



Value (XML):



Attributes (10)

- Attribute values cannot contain elements.
- In XML, 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).
- In SGML, it depends on the SGML declaration whether “<” can appear in attribute values. However, if it is permitted (like in HTML), it is not interpreted.

  It is simply a data character, and does not signal the begin of a tag.

Attributes (11)

- 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)

- The character set used in the SGML documents is defined in the SGML declaration.
- More precisely, one must distinguish between the repertoire of characters and the encoding of these characters in bytes for exchanging documents.
- HTML 4.01 and XML are both based on the Unicode character set (Repertoire).
- 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.

Character References (2)

- Other encodings contain e.g. cyrillic or japanese characters.
- The encoding is specified in the HTTP protocol (see Chapter 4).
- For XML documents, the XML declaration at the beginning of the file defines the encoding.
- The HTML 3.2 standard mentions only ISO Latin 1.

Character References (3)

- Browsers do not necessarily support the full Unicode character set.

  The characters in the ISO Latin 1 character set are also contained in the Unicode character set 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 (4)

- 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:

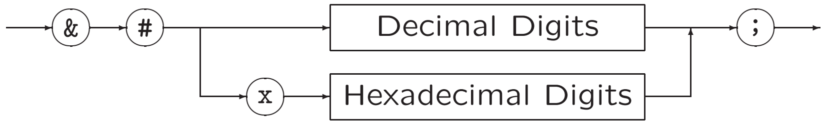
  `ä`

- The numbers refer to the repertoire (e.g. 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 (5)

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).

Character References (6)

- 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 do 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 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 the document type declaration they can appear only at places permitted by the grammar, i.e. between markup declarations.

  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.

Overview

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

Example

Simple DTD for a HTML-Subset:

```
<!ELEMENT HTML  O O (HEAD, BODY)>
<!ELEMENT HEAD  O O (TITLE)>
<!ELEMENT TITLE - - (#PCDATA)>
<!ELEMENT BODY  O O ((#PCDATA|P|EM|UL)*)>
<!ELEMENT P     - O ((#PCDATA|EM|UL)*)>
<!ELEMENT EM    - - (#PCDATA)>
<!ELEMENT UL    - - (LI+)>
<!ELEMENT LI    - O ((#PCDATA|P|EM|UL)*)>
```


Element-Type Declarations (1)

An Element-Type Declaration consists of:

- “<!” (MDO: „Markup Deklaration Open Delimiter“), followed by the keyword “`ELEMENT`”.
- Name of the element type to be declared.

  Such names are officially called “Generic Identifiers”. One can also specify several element types, see below.

- Specifications for markup minimization (if `OMITTAG`):
  - ◇ “`O`” (letter o) if the tag is optional
  - ◇ “-” (hyphen) if the tag is required

Element-Type Declarations (2)

Components of Element-Type Declaration, continued:

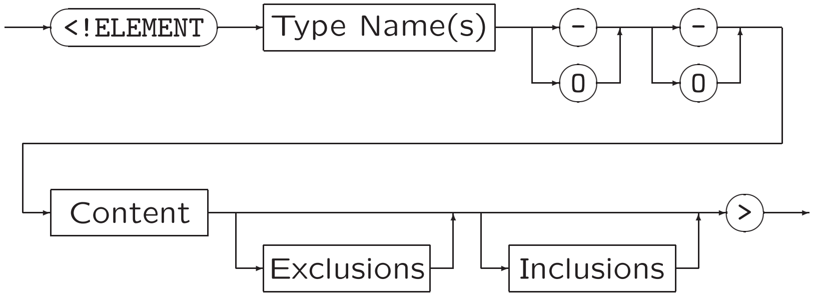
- First one writes the specification (`O` or `-`) for the start tag, then for the end tag.

  In XML, this part of the element type declaration is missing, since there is no markup minimization.

- Then one specifies what is permitted as content of this type of elements (“content model”).
- Optionally, exceptions for the content model (exclusions, inclusions) can be specified (not in XML).
- “>” (MDC: Markup Declaration Close Delimiter).

Element-Type Declarations (3)

Element-Type Declaration (SGML):



White space is required between each two components of the declaration except before the final “>” where it is optional. Between “<!” and the keyword “`ELEMENT`”, no white space is permited.

Element-Type Declarations (4)

- If several element types have the same content model and tag minimization rules, one can combine their definitions. E.g.

```
<!ELEMENT (B|I|U|TT) - - (#PCDATA)>
```

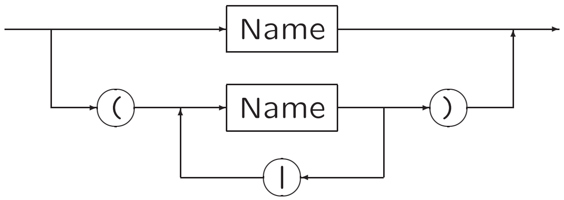
  is equivalent to

```
<!ELEMENT B  - - (#PCDATA)>
<!ELEMENT I  - - (#PCDATA)>
<!ELEMENT U  - - (#PCDATA)>
<!ELEMENT TT - - (#PCDATA)>
```

- This example defines some tags for specifying fonts in HTML (boldface, italics, underlined, teletype).

Element-Type Declarations (5)

Type Name(s):



- In XML, only a single name can be specified.
- SGML also supports ranked elements that consist of a stem and a numeric suffix (e.g. `P1`, `P2`, ...).

  One can always declare element types `P1`, `P2`, etc., but with this feature one can also write "<P>" and SGML picks the current level.

Element-Type Declarations (6)

Element-Type Declaration (XML):

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.

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. There is also "`CDATA`" (SGML only), which is not syntactically analyzed (like "`verbatim`" in LaTeX).

Content Specifications (2)

- 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.
  - ◇ $(A$ `&` $B)$: "A and B".

    This is equivalent to `(`$(A$`,`$B)$ `|` $(B$`,`$A)$`)`. A and B must both appear, but in arbitrary sequence.
- In XML, only "|" and "," are supported.

  "`&`" could be replaced with the other two operators, but an "and" of many components becomes clumsy (see also deterministic parsing).

Content Specifications (3)

- 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).
  - ◇ $(A_1$`&...&`$A_n)$: "And" (all A_i in any sequence).

Content Specifications (4)

- 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 (5)

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

    Element types must always be specified within parentheses.
  - ◇ the keyword `EMPTY`: No content permitted.
  - ◇ the keyword `CDATA`: Arbitrary character data.

    Even the special characters `<`, `>`, `&` can be used. They are not interpreted within an element with content "`CDATA`".
  - ◇ the keyword `RCDATA`: Like `CDATA`, `&` is interpreted.

    I.e. one can use character and entity references.
  - ◇ The keyword `ANY`: Character data and elements of arbitrary type.

Content Specifications (6)

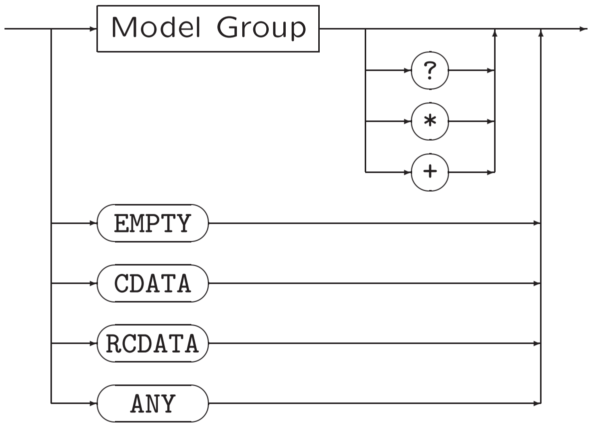
- XML does not support CDATA and RCDATA.
- Exceptions (inclusions, exclusions) can be specified in SGML for model groups and for ANY, but not for EMPTY, CDATA, RCDATA.

  These content models cannot contain elements, therefore exceptions do not make sense for them. XML does not support exceptions at all.
- SGML has also a feature called "data tags", which is, however, not included in XML, and not used in HTML.

  It permits that data characters can be interpreted as end tags.

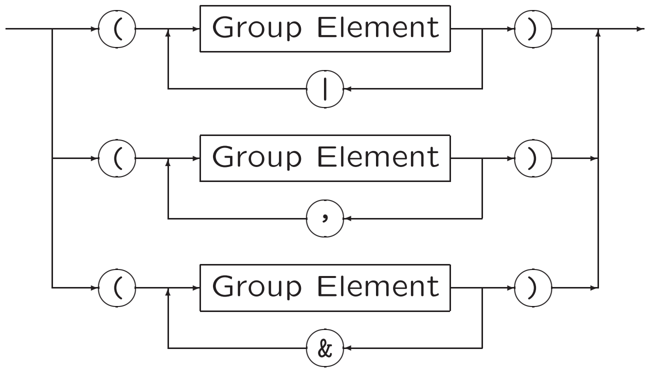
Content Specifications (7)

Content (SGML):



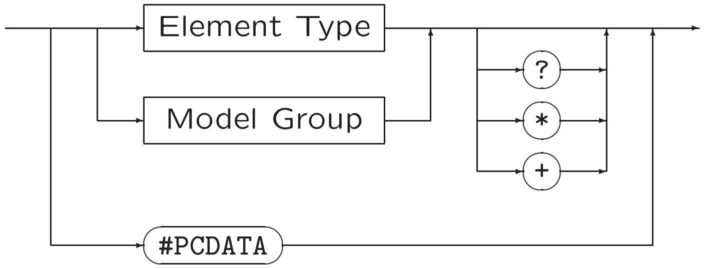
Content Specifications (8)

Model Group (SGML):



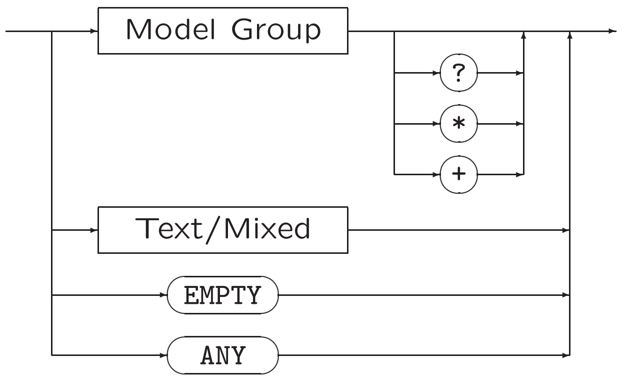
Content Specifications (9)

Group Element (SGML):



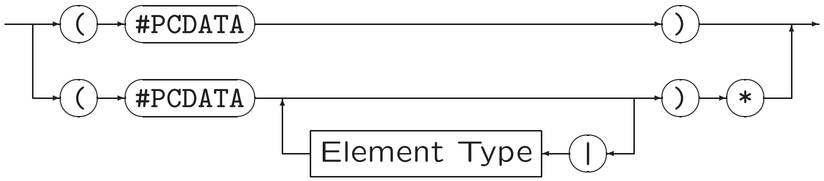
Content Specifications (10)

Content (XML):



Content Specifications (11)

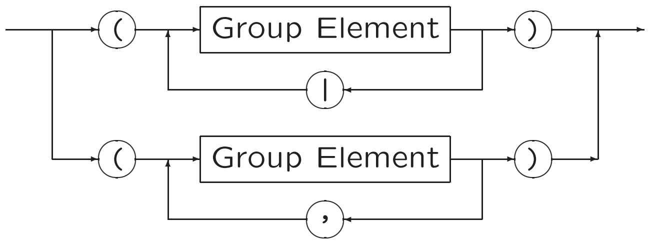
Text/Mixed (XML):



- In XML, the only content models that can contain `#PCDATA` are:
  - ◇ `(#PCDATA)`
  - ◇ `(#PCDATA | Element-Type | ... | Element-Type)*`

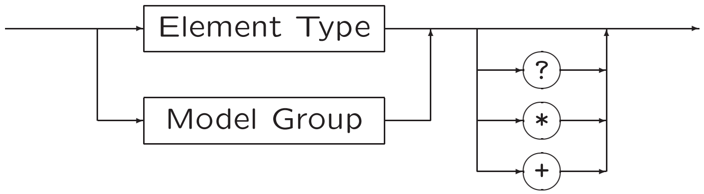
Content Specifications (12)

Model Group (XML):



Content Specifications (13)

Group Element (XML):



Content Specifications (14)

- 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 (15)

- 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 (16)

- 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.

Exceptions (1)

- Sometimes there are element types that should be permitted anywhere in the document.
- E.g. it should be possible to add a note with a element NOTE anywhere (like a comment).
- However, the DTD would get very complicated if one had to adapt every content model such that the element "NOTE" is really everywhere permitted.
- Therefore, one can specify inclusions in SGML element declarations, i.e. element types that are permitted anywhere inside the declared element type.

Exceptions (2)

- Example:
```
<!ELEMENT DOCUMENT - -(TITLE, AUTHOR, SECTION+)
                      +(NOTE)>
```
- With this declaration, `NOTE` elements can appear everywhere inside the `DOCUMENT` element:
  - ◇ as direct children, i.e. in front of/between/after `TITLE`, `AUTHOR`, `SECTION`,
  - ◇ also indirectly inside these elements (in the whole subtree), unless they are explicitly excluded.

    The declaration of `TITLE`, `AUTHOR`, `SECTION` and their subelements does not have to be changed.

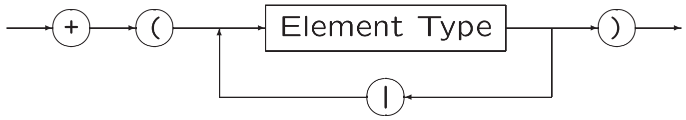
Exceptions (3)

- Since `NOTE` is now allowed anywhere inside `DOCUMENT`, it can also be nested.
- This would be a bit strange. One can apply the other kind of exception, an exclusion, to forbid `NOTE` inside `NOTE`:
```
<!ELEMENT NOTE - - (#PCDATA) -(NOTE)>
```
- XML does not support inclusions/exclusions.

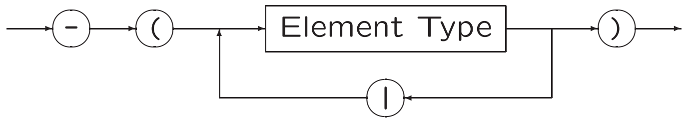
  Also in SGML, one should not use this feature too much, since otherwise the really possible tree structures can become unclear.

Exceptions (4)

Inclusions (SGML only):



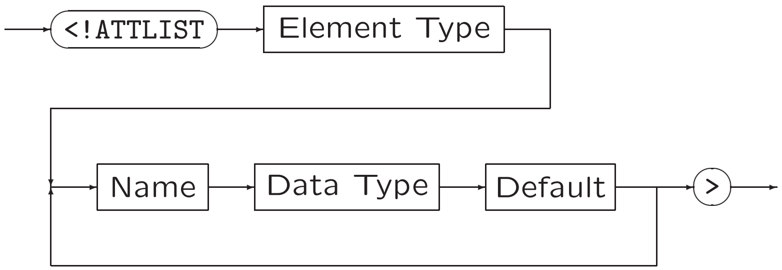
Exclusions (SGML only):



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)

- In SGML, but not in XML, an `ATTLIST` declaration can define attribute lists for several element types:

```
<!ATTLIST (TH|TD) COLSPAN NUMBER 1>
```

- The same attribute name can appear in an `ATTLIST` declaration only once.
- It is recommended (required in SGML?) that for every element type, there is only one `ATTLIST` declaration which defines all its attributes.

  In XML, 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.

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. The characters “<” and “>” are not interpreted (attribute values cannot contain elements). In SGML, both are valid, in XML, “<” is forbidden. In this way, missing quotes are easier found.

Attribute Data Types (2)

- `NAME`: A valid name.

  It depends on the SGML declaration which characters are permitted. E.g. a sequence of letters and digits starting with a letter. If the SGML declaration specifies that names are case-insensitive, the SGML parser will turn all names to uppercase. The application program does not have to worry about this. XML does not support `NAME`.

- `NAMES`: List of `NAME`-values, separated by white space.

  E.g. `"abc def"`. White space means one or more spaces, tabs, and line ends (can be configured in the SGML declaration). The SGML parser will normalise the value such that the names are separated by a single space, and there is no space before the first and after the last name. XML does not support `NAMES`.

Attribute Data Types (3)

- `ID`: `NAME`-value that is uniquely identifies this element (within the entire document).

  I.e. it is impossible that two elements 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`. It is recommended to use the same name for all attributes of type `ID`, and the attribute name "`ID`" is commonly used.

- `IDREF`: `NAME`-value 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 (4)

- `NMTOKEN`: Sequence of name characters.

  E.g. in XML, this is an arbitrary sequence of letters, digits, underscores "_", hyphens "-", periods ".", and colons ":". In SGML, the possible characters depend on the SGML declaration.

- `NMTOKENS`: List of `NMTOKEN`-values.
- `NUMBER`: List of digits. (Not supported in XML.)
- `NUMBERS`: List of `NUMBER`-values.
- `NUTOKEN`: Like `NMTOKEN`, but must start with a digit.

  E.g. "`10.5cm`". Not supported in XML.

- `NUTOKENS`: List of `NUTOKEN`-values.

Attribute Data Types (5)

- `ENTITY`: Name of an entity.

  Entities are explained fully below, they are a kind of macros or include files. An attribute of type `ENTITY` takes as value the name of a declared unparsed entity.

- `ENTITIES`: List of `ENTITY`-values.
- `NOTATION` $(N_1|\ldots|N_m)$: 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 (6)

- In summary, XML supports only the following attribute data types:
  - ◇ Enumeration types,
  - ◇ `CDATA`,
  - ◇ `ID`, `IDREF`, `IDREFS`,
  - ◇ `NMTOKEN`, `NMTOKENS`,
  - ◇ `ENTITY`, `ENTITIES`,
  - ◇ enumerations of notations.

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.

- 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.

Default Values (3)

- Default value, `#IMPLIED`, `#REQUIRED`, `#FIXED` are supported in XML.
- SGML has in addition:
  - ◇ `#CURRENT`: The last specified value is used.

    The attribute is required for the first element of the type. Thereafter, it is optional. If no value is specified, the attribute value of the last element of the same type is used.
  - ◇ `#CONREF`: If a value for this attribute is specified, the content of the element must be empty.

    "`CONREF`" stands for "content reference". In a way determined by the application, using this attribute is an alternative to specifying content of the element.

Comments in DTDs

- Within the markup declaration, one can write comments which are enclosed in "--".

```
<!ATTLIST IMG -- Image --
  src CDATA #REQUIRED -- URI of the image --
  alt CDATA #REQUIRED -- Text if no image --
  width CDATA #IMPLIED -- width in pixels --
  height CDATA #IMPLIED -- height in pixels -->
```

- This is only possible in SGML, not in XML.
- A markup declaration can also contain only a comment. This gives the comments as explained above:

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

Overview

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. If a line end follows the entity name, this line end is removed. Some SGML users define an empty entity especially for this purpose. In XML, the ";" is always required.

Entities: Overview (2)

- In SGML, there are different parameters for the case-sensitivity of entity names and of all other names (especially element types).

  "`NAMING ... NAMECASE GENERAL YES ENTITY NO`" means that element type names will be case-insensitive (lowercase letters are replaced by their uppercase counterpart) while enity names will be case-sensitive.

- E.g. in HTML, element type names ("tags") are not case sensitive, but entity names are.

  E.g. "`ä` is an "ä", while "`Ä` is an "Ä".

- In XML, names are always case-sensitive.

Entities: Overview (3)

- There are different kinds of entities. The above example is a geneneral, 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 (4)

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

  Non-SGML data cannot be directly included in an SGML 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 (1)

- Entities reduce the typing effort (abbreviations).

  The entity name may be much shorter than the replacement text. If the replacement text needs to appear several times in the document, one saves keystrokes.

- If in the above example, the Oracle version changes, one must change only the replacement text in the entity definition (at one place).

  If one did not use an entity, one has to edit all source files with "Search&Replace".

- The entity name might be easier to remember than its replacement text (e.g. `ä` stands for `ä`).

Entities: Motivation (2)

- Using entity names for special characters also improves the portability of the document to systems using other codes.
- Using entities leads to a higher uniformity.

  Otherwise one might write e.g. ORACLE 8.1.6 in some places, and Oracle 8.1.6 in others, and Oracle 8.1.5 in still others.

- 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.

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" (the part in the document itself, see below) 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 "/sgml/copyr.sgm">
```

- Then the entity reference "&copyr;" in the document is replaced by by the contents of the file "/sgml/copyr.sgm".

  The keyword "SYSTEM" indicates that the following string gives a system-dependent way to retrieve the entity. 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)

- In SGML, the grammar describes where exactly parameter entity references are permitted.
- It seems that the rules mean that if a parameter entity reference would be replaced by a space, the entire declaration must still be syntactically valid.
- The XML specification says that the grammar rules are only meant to hold after all parameter entities are replaced (i.e. here they act as real macros).

Parameter Entities (6)

- 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)

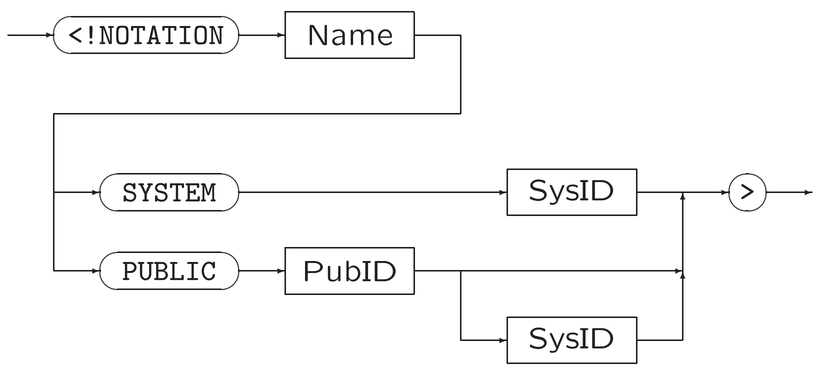
- A notation can be declared with a system identifier, which typically refers to a program that could display the data:

```
<!NOTATION GIF SYSTEM "xv.exe">
```

- However, the SGML parser only passes the system identifier to the application program. It depends on this program, how it uses this information.
- Besides system identitifiers, there are also public identifiers which are supposed to be not system-dependent.

Notations (3)

Notation Declaration:



Notations (4)

- Public and system identifier (PubID, SysID) are strings enclosed in single or double quotes (' or ").

  Public identifiers can use only a restricted character set in order to make them very portable, see below. Entity references are not evaluated in public and system identifiers.

- In SGML (not in XML), even after the keyword "SYSTEM", the system identifier can be left out under certain conditions.
- In SGML (not in XML), notations can have attributes ("data attributes").

Public Identifiers (1)

- Usually, an SGML system has a table (e.g. in a configuration file) that maps public identifiers to system-dependent information.
- Normally, public identifiers are globally unique and 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 (but see below for examples).

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" (see below).
- 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"`

Public Identifiers (4)

- Public identifiers can be any string of letters, digits, certain special characters, spaces and line breaks.

  Special characters in SGML: `'()+,-./:=?`. These characters are expected in any character set. In XML: `'()+,-./:=?;!*#@$_%`. Sequences of line breaks and spaces are replaced by a single space, and ignored at the very beginning or end.

- 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.

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`.

  "`NONSGML`" means a non-SGML data entity, so the public identifier used in the XML Bible seems incorrect.

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. (in XML):

```
<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)

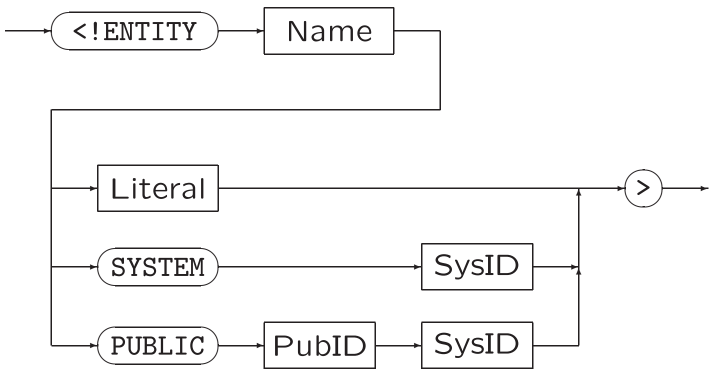
- Of course, parsed general entities can also be used as attribute values (not only unparsed entities).

  Only general entities can appear as attribute values (parameter entities have no meaning in the entity).

- 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)

- System identifier and public identifier are strings enclosed in single or double quotes (`'` or `"`).

  In SGML, the system identifier is optional under certain conditions ("if the system can generate it from the public identifier and/or other information available to it"). In XML, the system identifier is required (as shown in the syntax diagram).

- Entity references are not interpreted in system identifier and public identifier.
- In XML, the system identifier is a URI, which may be used to retrieve the entity.

Entity Declaration (3)

- “Literal” is a string enclosed in single or double quotes. (' or ").

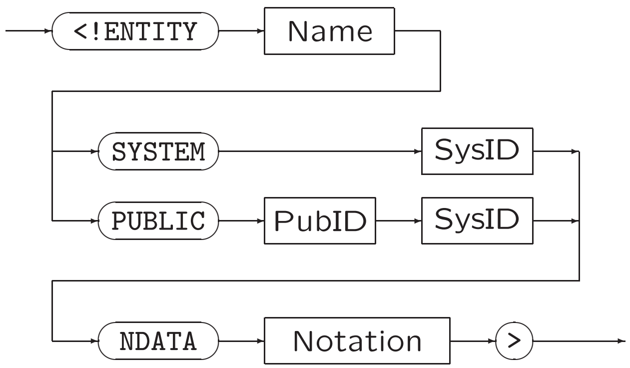
  Parameter entity references and general entity references can be used here. Parameter entity references are immediately evaluated, general entity references become part of the replacement text of the entity.

- Entity declarations in SGML have additional features not shown in the syntax diagram.

  E.g. one can specify that the value of the entity is always treated as character data (uninterpreted), regardless of the context in which the entity is referenced. There is also “bracketed text” as a syntactic convenience for entering the entity value in the entity declaration.

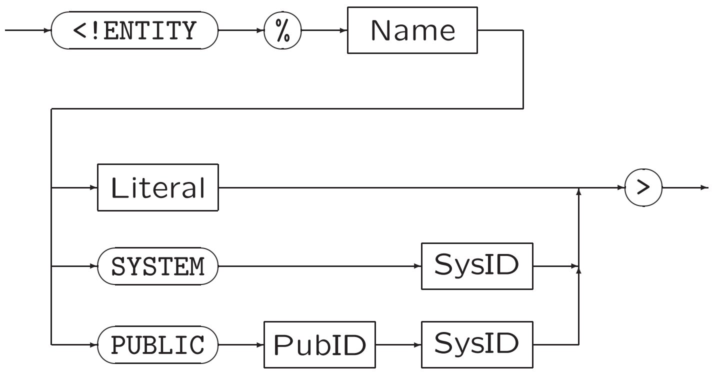
Entity Declaration (4)

Unparsed Entity Declaration:



Entity Declaration (5)

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

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

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

  Even in the body of the document, one uses parameter entities for the keyword, since marked sections starts with "<!" and count as declaration.

- In XML, conditional marked sections can only be used in the external subset of the DTD.

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 are supported in SGML and XML.
- `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.

Marked Sections (5)

- `RCDATA` sections are similar, but permit entity references, i.e. "&" is interpreted, but not "<" and ">".
- Finally, section that should be deleted or reworked, can be marked as follows:

  `<![TEMP[...]]>`

- "`RCDATA`" and "`TEMP`" are only supported in SGML, not in XML.

Overview

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

DOCTYPE Declaration (1)

- For every SGML document, there is a DTD that defines the syntax of the document.
- The DTD does not need to be declared in the document, it can e.g. also be built into the software.

  Formally, an SGML document entity consists of optional white space, an SGML declaration (“`<!SGML ...>`”), a prolog which contains a document type declaration (“`<!DOCTYPE ...>`”) plus possibly comments, processing instructions, and white space, then the document element, followed possibly by comments, processing instructions, and white space (SGML also supports multiple document type declarations and document elements, but this is probably not used often.). The SGML declaration is often contained in a separate file or built into the parser. Entity and file structure need not be the same.

DOCTYPE Declaration (2)

- In XML, the DTD is optional.
- 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.

DOCTYPE Declaration (3)

- 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”.

- In contrast, an SGML document can normally not be parsed without knowing the DTD because of markup minimization (optional start and end tags).

DOCTYPE Declaration (4)

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

```
<!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 (5)

- Instead of a system identifier, one can also use a public identifier, if the DTD is well known.

  SGML systems come with a folder that contains several DTDs and a configuration file that maps public identifiers to these DTDs. XML accepts a public identifier only together with a system identifier which is a URI under which the DTD can be retrieved.

- Example:

```
<!DOCTYPE HTML
    PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
    ...
</HTML>
```


DOCTYPE Declaration (6)

- One can also specify public and system identifier (in XML the system identifier is always required):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/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 (7)

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

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

- Also a mixture of both is possible:

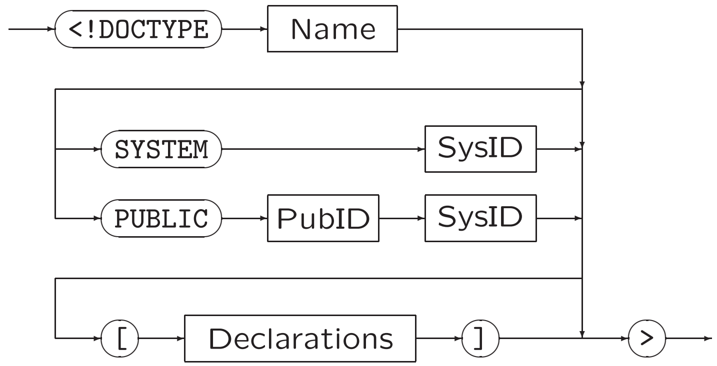
```
<!DOCTYPE EMAIL SYSTEM "mail.dtd" [...]>
```

DOCTYPE Declaration (8)

- The part in the document itself (“[...], “internal DTD subset”) is processed before the DTD file (“external subset”).
- In SGML and XML, the same entity 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.

DOCTYPE Declaration (9)

DOCTYPE Declaration:



DOCTYPE Declaration (10)

- In SGML, the system identifier can be left out if the SGML parser can somehow reconstruct it.

  E.g. from a configuration file that maps public identifiers to files.
- In XML, the system identifier is required. It must be a URI (possibly a relative one).
- In XML, the constructs used in the internal subset of the DTD are somewhat restricted (see parameter entity references, marked sections), such that a non-validating XML processor can easily skip it.

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.
- Processing instructions can contain any text, and are system- and application-dependent.

Processing Instructions (2)

- Processing instructions start with “<?” and end with “>” (SGML) or “?>” (XML).

  As mentioned before, SGML is heavily parameterized, and it is of course possible to choose “?>” for the parameter “pic” (processing instruction close). Probably “?>” was chosen in XML in order to permit “>” inside the processing instruction.

- In XML, 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.

Processing Instructions (3)

- In XML, 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).