Creating XML

Hans C. Arents

senior IT market analyst

I.T. Works

"Guiding the IT Professional"

Innovation Center, Technologiepark 3, B-9052 Gent (Belgium), Tel: +32 (0)9 241 56 21 - Fax: +32 (0)9 241 56 56

E-mail: hca@itworks.be - Site: http://www.itworks.be/ - Home: http://www.arents.be/

Creating XML

- Part 1: **XML basics + XML DTDs** 13u30 – 15u30

  coffee break

- Part 2: XML Schemas + tools overview 16u00 – 17u30

  questions & answers

XML basics + XML DTDs

Hans C. Arents

senior IT market analyst

I.T. Works

"Guiding the IT Professional"

Innovation Center, Technologiepark 3, B-9052 Gent (Belgium), Tel: +32 (0)9 241 56 21 - Fax: +32 (0)9 241 56 56

E-mail: hca@itworks.be - Site: http://www.itworks.be/ - Home: http://www.arents.be/

XML basics + XML DTDs

- An XML document
  - what's inside an XML document?
  - well-formed vs. valid XML documents
- Well-formed XML documents
- Valid XML documents
  - document instance
  - the role of the DTD
- DTD syntax
  - basic and advanced
- DTD design guidelines

What's inside an XML document?

- Element tags

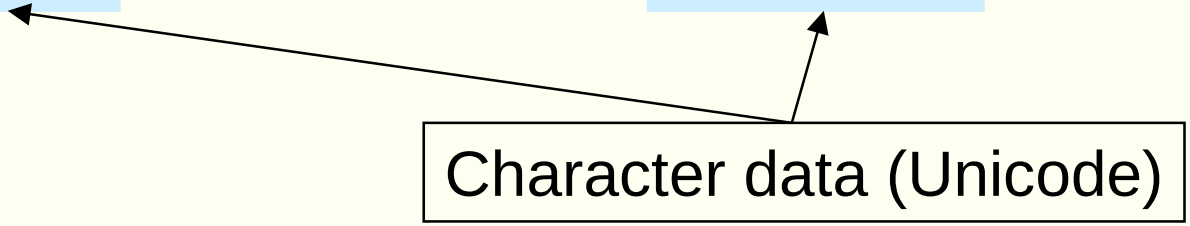
```
<item num="1">Look at <img src="1.jpg"/> now please.</item>
```

- Entity references

```
&lt; &gt; &XML;
```

- Comments

```
<!-- This is a sample XML document -->
```



- Processing instructions

```
<?AUDIO "FADE violin.wav"?>
```

- CDATA sections

```
<![CDATA[ … for (i=0; i<10; i++) … ]]>
```

- Document type declarations

```
<!DOCTYPE document SYSTEM "document.dtd" [ … ]>
```

Inside an XML document: element tags

- Document markup consists primarily of *element tags*
  - the actual characters the processing system has to recognize as markup and not confuse with the document's content

- example:

```
<TITLE>A document title</TITLE>
```

- component description

`<`	start tag open delimiter
`TITLE`	tag's generic identifier
`>`	start tag close delimiter
`</`	end tag open delimiter
`TITLE`	tag's generic identifier
`>`	end tag close delimiter

Inside an XML document: entity references

- purpose:
  - point to frequently occurring text
  - point to external objects (e.g. files)
- syntax:

  ```
  &name;
  ```

- example:

  &XML;

  declared in the DTD as:

  ```
  <!ENTITY XML "Extensible Markup Language">
  ```

- note:
  - the parser replaces the entity reference by its defined value

Inside an XML document: comments

- purpose:
  - insert comments ignored by the parser
  - make XML constructs understandable to people
- syntax:

```
<!-- comments -->
```

- example:

```
<!-- to be done later -->
```

- note:
  - comments are also allowed in the DTD

Inside an XML document: processing instructions

- purpose:
  - insert procedural commands ignored by the parser
  - provide instructions to a processing application
- syntax:

  <?*name* … ?>
- example:

```
<?PrintApp "insert line-break"?>
```
- note:
  - processing instructions that begin with **<?xml** are reserved for XML
  - used for non-standard, application-dependent commands
  - warning: reduce the portability of data/documents

Inside an XML document: CDATA sections

- purpose:
  - markup is not recognized inside
  - useful for hiding source code, examples, etc.
- syntax:

  <![CDATA […]]>
- example:

```
<![CDATA[ … for (i=0; i<10; i++) … ]]>
```
- note:
  - e.g. useful for embedding JavaScript into an XML document

Well-formed vs. valid XML documents

- **Well-formed**
  - conforms to XML syntax
  - parsing allows an element grove (“tree”) to be built
  - can be parsed without regards to a **D**ocument **T**ype **D**efinition (DTD)
- **Valid**
  - is well-formed
  - has a particular DTD
  - conforms to the rules expressed by that DTD

    (note: DTD does not say anything about document semantics)
- Conformance
  - XML processor: must report well-formedness violations
  - validating XML processor: must also report validity violations
  - must pass data to the application as specified in the XML standard

Well-formed XML documents

```
<?xml version="1.0" standalone="yes"?>
<Document>
  <Para align="left">My first <B><I>XML</I> document</B> is<BR/>
  still pretty simple, but my next document will be<BR></BR>
  very complex &amp; interesting indeed.</Para>
</Document>
```

- Well-formedness rules:
  - have a unique root element
  - all element tags must be balanced
  - all element tags must be properly nested
  - all attribute values must be fully quoted
  - empty element tags must end with '/>' or must be made non-empty
  - < and & are only used to start element tags and entity references (only five predefined `< > & &apos; &quot;`)

Well-formedness rules

- all element tags must be balanced and properly nested
  - if an element contains a start tag for an element, it must also contain the corresponding end tag for that element
  - empty elements may appear anywhere
  - every non-root element has a parent element

û
```
<p>… this text is
in <b>bold <i>italic</b>
for emphasis …</p>
```

û
```
<p>… this text is
in <b>bold <i>italic</b></i>
for emphasis …</p>
```

ü
```
<p>… this text is
in <b>bold <i>italic</i></b>
for emphasis …</p>
```

Well-formedness rules

- all attribute values must be fully quoted
  - û `<A HREF=http://www.itworks.be/xml/>`
  - ü `<A HREF="http://www.itworks.be/xml/">`
- correct use of less-than sign (<) and ampersand (&)
  - û `<publisher>O'Reilly & Associates</publisher>`
  - ü `<publisher>O'Reilly & Associates</publisher>`
- only five predefined entity references
  - û `&copy;`, `&reg;`, `&tm;`, `&eacute;`, ` `, etc.
  - ü `&`, `<`, `>`, `&quot;`, `&apos;`

Lexical constraints

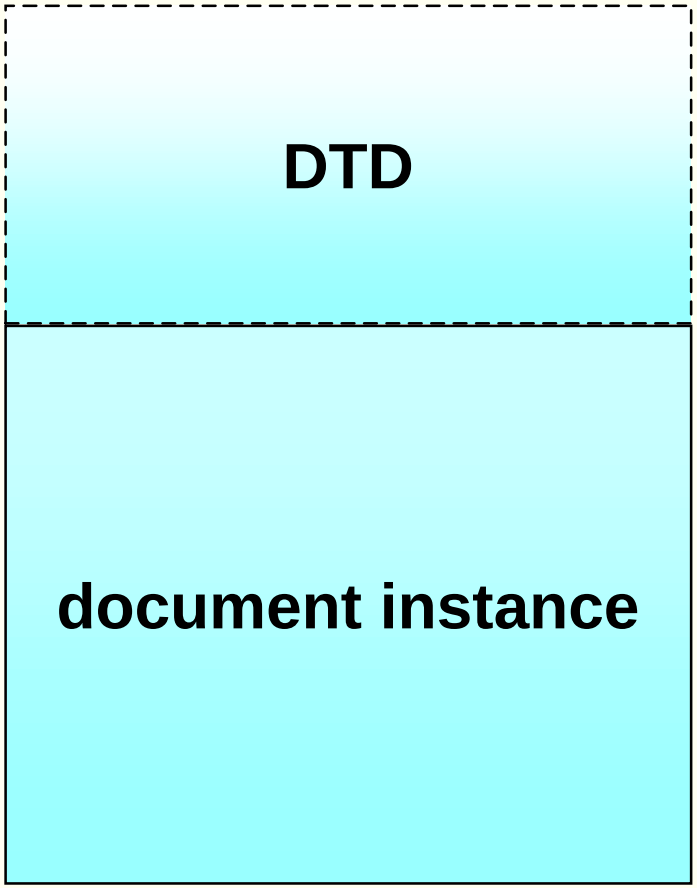
- names
  - first character: a … z, underscore
  - subsequent characters: a … z, 0 … 9, underscore, hyphen, period
  - may not contain whitespace
  - may contain colons only if used for namespaces
- note:
  - element names and attribute names are case-sensitive
  - attribute values are case-sensitive
  - entity names are case-sensitive

  (e.g. in XHTML element and attribute names are lowercase)

Valid XML documents

- a valid XML document consists of
  - a (reference to) **D**ocument **T**ype **D**efinition (DTD)
    - the declarations that specify the rules which the elements, attributes, … have to obey
  - a document instance
    - an actual document which conforms to the DTD
- a document instance contains
  - markup
    - the tags that delineate the content of the different document components
  - content
    - Unicode character data

Valid XML documents



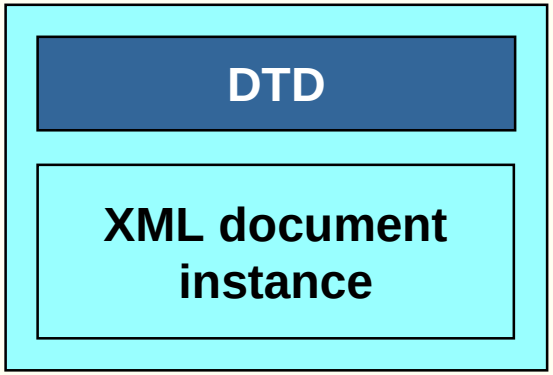
- (a reference to) a DTD
  - set of rules

    `DOCTYPE, ELEMENT, ATTLIST, ENTITY, NOTATION,` … declarations and comments
- document instance
  - markup

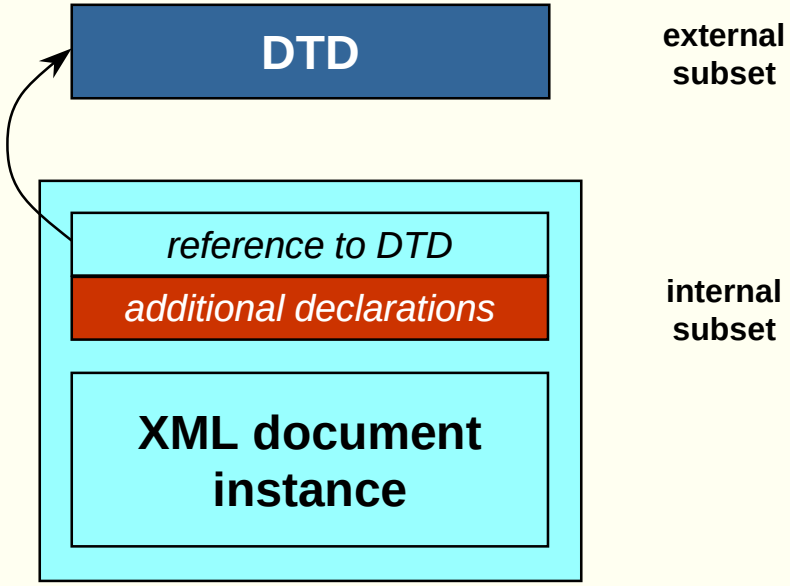
    start-tags, end-tags, character references, entity references, comments, processing instructions, CDATA sections, ...
  - content

    Unicode character data

Including/referring to a DTD

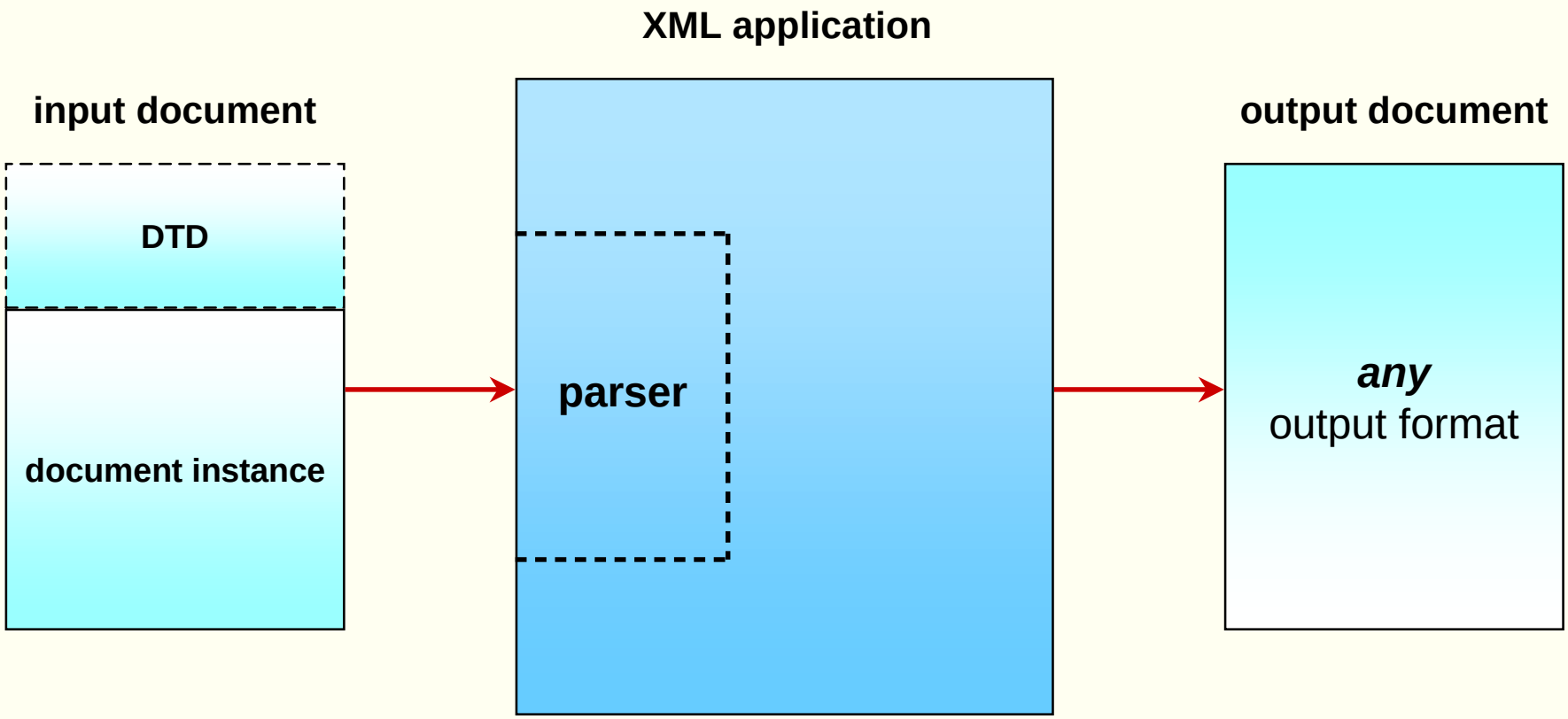



```
<?xml version="1.0">
<!DOCTYPE Document
[
<!ELEMENT Document … >
…
]>

<Document>…</Document>
```




```
<?xml version="1.0">
<!DOCTYPE Document SYSTEM
 "http://www.xml.org/doc.dtd"
[
…
]>

<Document>…</Document>
```

An XML processing application



The role of the DTD

- **D**ocument **T**ype **D**efinition = the “blueprint” of a document
  - valid element names
  - valid attribute names and values
  - how elements can follow each other (*sequencing*)
  - how elements can nest in one another (*occurrence*)
- Why is validation important?
  - guarantee what is to be delivered
    - ensure that the document/data format created is correct before sending it to another program
  - know/check what is to be expected
    - ensure that the document/data format received is correct before starting to process it

Basic DTD syntax

- A DTD contains declarations for:
  - elements
  - attributes of elements
- format of a declaration:

  <!keyword parameter associated parameter(s)>
  - example of an element declaration:

    ```
    <!ELEMENT document (front, body, annex?)>
    ```
  - example of an attribute declaration:

    ```
    <!ATTLIST document docid ID #REQUIRED>
    ```

Element declarations

- purpose of elements:
  - define the *structural components* of the document
    - e.g. heading, section, paragraph, …
  - define the *semantic components* of the document
    - e.g. warning, address, …
- syntax:

  `<!ELEMENT element_name content_model>`

  defines
  - the name of the element
  - the content model of the element

    (occurrence & sequencing expressed using a regular expression-like grammar)

Content model

- purpose:
  - define the allowed content of the element
    - only text or also other elements?
- consists of:
  - **textual content** *or*
  - **combined content** = a combination of
    - textual content
    - names of other elements
    - names of parameter entities

    combined by logical operators
- types of logical operators:
  - *occurrence indicators* **? * +**
  - *connectors* **, |**

Content model: textual content

- the element contains:
  - only text
  - no subelements
- reserved keyword for declaring textual content:
  - **#PCDATA** (parseable character data) string without markup
  - **EMPTY** (empty element)
  - **ANY** (any content)

Content model: textual content

- #PCDATA example:

```
<!ELEMENT YEAR (#PCDATA)>
```

- ü Valid:

```
<YEAR>1999</YEAR>

<YEAR>99</YEAR>

<YEAR>
The year of our
Lord one thousand,
nine hundred, and
ninety-nine
</YEAR>
```

- û Invalid:

```
<YEAR>
<MONTH>January</MONTH>
<MONTH>February</MONTH>
<MONTH>March</MONTH>
<MONTH>April</MONTH>
<MONTH>May</MONTH>
<MONTH>June</MONTH>
<MONTH>July</MONTH>
…
</YEAR>
```

Content model: textual content

- using XML Schema syntax:

```
<xsd:element name="YEAR" type="xsd:date"/>
```

  - using a built-in XML Schema datatype date
    - elements declared to be of type date must follow this form: CCYY-MM-DD

- Valid:

```
<YEAR>1999-06-23</YEAR>
```

- Invalid:

```
<YEAR>1999-06-31</YEAR>
<YEAR>23-06-1999</YEAR>
<YEAR>23rd of June 99<YEAR>
```

Content model: textual content

- EMPTY example:

```
<!ELEMENT IMG EMPTY>
```

- Valid:

```
<IMG … />
<IMG …></IMG>
```

- note:
  - in XHTML use <br/>, <hr/>, and <img/> instead of <br>, <hr>, and <img>
  - Web browsers (i.e. Netscape) sometimes treat these empty tags inconsistently so use <br />, <hr />, <img />

Content model: combined content

- the element contains:
  - textual content
  - names of other elements
  - names of parameter entities
- combined by logical operators and grouped with parentheses:

Notation	What it means
`A?`	matches A or nothing; optional A
`A+`	matches one or more occurrences of A
`A*`	matches zero or more occurrences of A
`A | B`	matches either A or B but not both
`A , B`	matches A followed by B, in that order
`(A , B)+`	matches one or more occurrences of A followed by B
`(A | B)*`	matches zero or more occurrences of either A or B

Content model: combined content

```
<!ELEMENT name (first, last)>
```

ü
```
<name><first>Hans</first>
      <last>Arents</last></name>
```

û
```
<name><last>Arents</last>
      <first>Hans</first></name>
```

û
```
<name><last>Arents</last></name>
```

û
```
<name><first>Hans</first></name>
```

```
<!ELEMENT name (first | last)>
```

ü
```
<name><first>Hans</first></name>
```

ü
```
<name><last>Arents</last></name>
```

û
```
<name><first>Hans</first>
      <last>Arents</last></name>
```

Content model: combined content

```
<!ELEMENT note (title, para)>

<!ELEMENT note (title?, para*)>

<!ELEMENT note ((title, subtitle)?, para+)>


<!ELEMENT list (title?, item+)>

<!ELEMENT list (title?, item, item+)>
```

Content model: mixed content

- Both text (#PCDATA) and child elements in an OR sequence
  - #PCDATA must come first
  - #PCDATA cannot be used in an AND sequence

```
<!ELEMENT P (#PCDATA | A | B | I)*>
```

```
<P>Text <B>and</B> child elements
may be freely intermingled in
<A href="http://www.xml.com/">XML</A>
documents which have several
<I>mixed content</I> elements.</P>
```

Attribute declarations

- purpose of attributes:
  - specify *defining characteristics* of an element
  - specify *values* for those characteristics
    - e.g. access (public or confidential), status (draft or final), …
- types of attributes:
  - optional *or* mandatory
- syntax:

```
<!ATTLIST element_name
          attribute_name values default>
```

  defines
  - the name of the attribute
  - the allowable values of the attribute
  - the default value of the attribute

Allowable values of an attribute

- enumerated list of values

```
<!ELEMENT document (title?, para+)>
<!ATTLIST document
          status   (draft|reviewed|final) "draft">
```

```
<document status="final">
```

- note:
  - attribute value must be quoted
  - attribute names cannot be omitted
  - attribute names must be unique within element

Allowable values of an attribute

- reserved keyword for allowable values:
  - **CDATA**

    string not containing a less-than sign (<) or quotation marks (")
  - **ID**

    an identifying value (unique for the document)
  - **IDREF(S)**

    one (or more) reference(s) to an `ID(S)` in the same document
  - **NMTOKEN(S)**

    one (or more) legal nametoken(s)
  - **ENTITY** or **ENTITIES**

    one (or more) entitie(s) declared in the DTD
  - **NOTATION**

    value is the name of a notation declared in the DTD

Default value of an attribute

- a value from an enumerated list of values
- reserved keyword for default value:
  - **`#REQUIRED`**

    an attribute value is required

    no default value is provided in the DTD

    the document author must provide a value
  - **`#IMPLIED`**

    an attribute value is optional

    the document author may provide a value

    if not present, value may be provided by the processing application
  - **`#FIXED`**

    the attribute value is the same for all elements

    a default value must be provided in the DTD

    the document author may not change the default value

Allowable / default value(s) of an attribute

- CDATA example:

```
<!ELEMENT img EMPTY>
<!ATTLIST img label CDATA #REQUIRED>
  …  seen in <img label="Sleepy dog"/> mentioned …
```

- ID and IDREF example:

```
<!ELEMENT p    (#PCDATA | link)*  >
<!ATTLIST p    nr ID      #IMPLIED>
<!ELEMENT link (#PCDATA)>
<!ATTLIST link ref IDREF #REQUIRED>
  <p>This concept is explained in more detail
  in <link ref="p381">the paragraph</link> below.</p>
  …
  <p nr="p381">A definition of … </p>
```

Allowable / default value(s) of an attribute

- example

```
<!ELEMENT img EMPTY>
<!ATTLIST img
          src     ENTITY                    #REQUIRED
          label   CDATA                     #IMPLIED
          align   (left|center|right) "left"
          valign (top|middle|bottom) "bottom"
          scale   CDATA                     "100"
          copyright CDATA                   #FIXED "hca">
```

Attributes vs. elements

- Attributes
  - if the user does not need to see the information
  - for **meta-data**: properties of document components
    - IDs, URLs, presentation hints
      and other information not directly relevant to the user
  - when the information does not have too much internal structure
- Elements
  - if the user needs to see the information
  - for **data**: actual content of the document components
  - when the information has internal structure
    - which needs to be easily processed
    - which has to be adaptable to future changes

Attributes vs. elements

- Information has structure which we do not want to exploit

```
<para creationdate="22-05-2000" …> … </para>
```

- Information has structure which we want to exploit

```
<para …>
  <creationdate>
    <day>22</day>
    <month>05</month>
    <year>2000</year>
  </creationdate>
  …
</para>
```

Advanced DTD syntax

- A DTD also contains declarations for:
  - entities
  - notations

  and conditional sections

- format of a declaration:

  <!keyword parameter associated parameter(s)>

  - example of an entity declaration:

    ```
    <!ENTITY UN "United Nations">
    ```

  - example of a notation declaration:

    ```
    <!NOTATION GIF SYSTEM "Graphics Interchange Format">
    ```

Entity declarations

- purpose:
  - shorthand notation for textual and data content (“macro’s”)
- types of entities:
  - **general entities** (referred to within DTD and document instance)
    - character entities
  - **parameter entities** (referred to only within DTD)
- use:
  - an entity must be defined using an *entity declaration*
  - an entity can be called using an *entity reference*

Character entities

- purpose:
  - to represent one of Unicode's more than 50,000 characters
    - not accessible from an ordinary ASCII editor
    - only five predefined character references (`< > & &apos; &quot;`)
  - by giving the number of the particular Unicode character
    - in decimal or hexadecimal form
  - can only be used in element content, attribute values and comments
- examples:

  Greek small letter mu: μ decimal: `&#956;` hexadecimal: `&#x3BC;`

```
<p>This feature is only 5 &#x3BC;m large.</p>
```

- note:
  - define frequently used character references as general entities

```
<!ENTITY mu "&#x3BC;">
<!ENTITY micron "&mu;m">
```

General entities

- purpose:
  - include frequently occurring text
  - include external objects (e.g. files)
- syntax:

  entity declaration in the DTD

  ```
  <!ENTITY  name  value  type>
  ```

  entity reference in the DTD or document instance

  ```
  &name;
  ```

General entities

- to include frequently occurring text

```
<!ENTITY XML "Extensible Markup Language">
  <p>This seminar on &XML;
  is organized by IT Works.</p>
```

- to include external objects (e.g. files)

```
<!ENTITY chap1 SYSTEM "chapter1.xml">
<!ENTITY chap2 SYSTEM "chapter2.xml">
  <book>
  &chap1;
  &chap2;
  </book>
```

Parameter entities

- purpose:
  - include frequently occurring text *only in the DTD*
  - include external text files *only in the DTD*
- syntax:

  entity declaration in the DTD

  ```
  <!ENTITY  %  name  value  type>
  ```

  entity reference in the DTD

  ```
  %name;
  ```

Parameter entities

- to reuse content models:

```
<!ENTITY % textblock "(para|quot)"          >
<!ELEMENT abstract  (%textblock;)+          >
<!ELEMENT legalnote (%textblock;|copyright)+>
```

  which the parser internally expands to:

```
<!ELEMENT abstract  ((para|quot))+          >
<!ELEMENT legalnote ((para|quot)|copyright)+>
```

- to make customized "keywords":

```
<!ENTITY % yesorno "CDATA">
<!ENTITY % yes     "1">
<!ENTITY % no      "0">
…
<!ATTLIST document indexed %yesorno; %no;>
```

Identifiers

- purpose:
  - specify logical addresses that locate physical resources for the XML processor
  - reusing resources
    - DTDs
    - subdocuments
- types:
  - **system identifiers**
    - point to resources on a local system or a remote system
  - **public identifiers**
    - point to resources that are publicly available

Identifiers

- primarily used to refer to a DTD
  - tell the parser which DTD to use
    to parse and validate the document instance
- in the document instance itself

```
<!DOCTYPE memo [ … declarations … ]>
```

- using a system identifier

```
<!DOCTYPE memo SYSTEM "c:\dtds\memo.dtd">
<!DOCTYPE memo SYSTEM "http://www.xml.org/memo.dtd">
```

- using a public identifier

```
<!DOCTYPE article
      PUBLIC "ISO 12083:1993//DTD Article DTD//EN”
      SYSTEM "c:\dtds\article.dtd">
```

Notations

- purpose:
  - tell the parser how to handle non-textual, binary content
- example:

```
<!NOTATION GIF SYSTEM "Graphics Interchange Format">
…
<!ELEMENT img EMPTY>
<!ATTLIST img src ENTITY #REQUIRED>
…
<!ENTITY pic1 SYSTEM "pic1.gif" NDATA GIF>

   <p>This dish <img src="pic1"/>
   looks quite tasteful.</p>
```

Conditional sections

- purpose:
  - construct *conditional* DTDs,
    i.e. DTDs that include or exclude
    declarations according to certain circumstances
  - useful for managing editorial and publishing processes
- conditional sections are allowed
  - in the external DTD *only*
- syntax:
  - do use for validation

```
<![INCLUDE [ … ]]>
```

  - do not use for validation

```
<![IGNORE [ … ]]>
```

Conditional sections

```
…
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >
…
<![%draft;[
  <!ELEMENT book (comment*, head, body, appendix?)>
]]>
…
<![%final;[
  <!ELEMENT book (head, body, appendix?)>
]]>
…
```

Conditional sections

```
…
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >
…
<![INCLUDE[
  <!ELEMENT book (comment*, head, body, appendix?)>
]]>
…
<![IGNORE [
  <!ELEMENT book (head, body, appendix?)>
]]>
…
```

Conditional sections

```
…
<!ENTITY % draft 'IGNORE' >
<!ENTITY % final 'INCLUDE' >
…
<![%draft;[
  <!ELEMENT book (comment*, head, body, appendix?)>
]]>
…
<![%final;[
  <!ELEMENT book (head, body, appendix?)>
]]>
…
```

Conditional sections

```
…
<!ENTITY % draft 'IGNORE' >
<!ENTITY % final 'INCLUDE' >
…
<![IGNORE [
  <!ELEMENT book (comment*, head, body, appendix?)>
]]>
…
<![INCLUDE[
  <!ELEMENT book (head, body, appendix?)>
]]>
…
```

An example DTD

```
<!-- DTD for simple memoranda                                    -->
<!ENTITY % text "#PCDATA | bold | ital"                           >

<!--      NAME CONTENT MODEL                                     -->
<!ELEMENT memo (head , body)                                     >
<!ELEMENT head (from , to+ , cc* , subj)                         >
<!ELEMENT body (para+ , sign?)                                   >
<!ELEMENT from (#PCDATA)                                         >
<!ELEMENT to   (#PCDATA)                                         >
<!ELEMENT cc   (#PCDATA)                                         >
<!ELEMENT subj (#PCDATA)                                         >
<!ELEMENT para (%text; | img)*                                   >
<!ELEMENT sign (%text;)*                                         >
<!ELEMENT bold (#PCDATA)                                         >
<!ELEMENT ital (#PCDATA)                                         >
<!ELEMENT img  EMPTY                                             >

<!--      ELEMENT NAME     VALUE                    DEFAULT    -->
<!ATTLIST memo    priority (Low | Medium | High) "Low"          >
<!ATTLIST img     src      ENTITY                  #REQUIRED
                  type     CDATA                   #FIXED "GIF">
```

DTD design: where to start?

- Use an existing DTD
  - public domain DTDs
    - now: large collection at http://www.schema.net/
    - future: global repository at http://www.xml.org/
  - industry-standard DTDs

• online help	DocBook
• newspapers	XMLNews, NITF
• business & finance	OFX, FpML, FinXML, …

- Adapt an existing DTD
  - learn from it
  - add/subtract from it
- Develop your own DTD from scratch or have it done for you
  - using established DTD design methodologies
    - already exist for document modeling, under development for data modeling

DTD design: how to proceed?

- Define the environment
  - what are your different types of documents?
  - who will be using these documents? for what purpose?
  - what existing tools/standards do you have to keep respecting?
- Analyse sample documents
  - identify potential components
  - classify components:
    - block components (information units) e.g. para, warning, example, …
    - inline components e.g. bold, italic, link, …
    - meta-information components e.g. author, keywords, …
  - confirm the need for a particular component
  - build the document hierarchy

DTD design: how to test?

- To test the validity of the document model

  as expressed by the DTD:
  - mark up the sample documents
    used in the document analysis phase
  - markup typical and unusual sample documents
    not used in the document analysis phase
- Two possible types of problems:
  - the document model is too constrained
    - e.g. not enough markup options to capture desired future use
  - the document model is too broad
    - e.g. too much freedom of choice in selecting markup options
- Test the DTD in the real world
  - usable by the XML processing system and its intended users