*:96 Overheads

Part 2ca: Extensible Markup Language (XML)

More about this course about Internet application protocols can be found at URL:

http://dsv.su.se/jpalme/internet-course/Int-app-prot-kurs.html

Last update: 00-03-24 17.24

HTML Example

```
<h2>False Pretences</h2>
<p><b>By: </b>Margaret Yorke<br>
<b>ISBN: </b>0-312-19975-9<br>
<b>Year: </b>1999</p>
```

XML Example

```
<book><author><surname>Yorke</surname>
<given-name>Margaret</given-name></author>
<title>False Pretences</title>
<isbn>0-312-19975-9</isbn>
<year>1999</year></book>
```

The difference between HTML and XML: In XML you can yourself decide which tags to use. In HTML, you can only use the built-in tags specified in HTML. In the example above, I used the tags **<book>**, **<author>**, **<surname>**, **<given-name>**, **<title>**, **<isbn>** and **<year>**. In another application, I could have chosen other tags.

By **combining** of XML with style sheets, you can still get the documented printed in the same way as if you had been using HTML.

Uses of XML

- For transport of information between data bases.
- For sending of information to be displayed to a user, just like with HTML.
- As a rather readable format in itself (except for encoding of special characters).
- For encoding of network operations, as an alternative to ABNF or ASN.1.

Restrictions of XML

- Binary data must be either encoded as BASE64 or sent outside of the XML document (like in HTML).
- A rather wordy format, but compression can reduce this.

Some acronyms

Standard Generalized Markup Language (SGML)

HTML and XML are both simplifications of SGML.

Application Program Interfaces (APIs)

Document Object Model (DOM) is an API för XML. Supported by newer web browsers.

Simple API for XML (SAX) standard for API to XML partsers. Streambased - the XML content is delivered in increments during its interpretation.

Style sheet languages

The same XML document can be shown in different formats, by using different style sheets.

eXtensible Style Sheet Language (XSL).

eXtensible Style Sheet Language Transformatsions (XSLT).

Cascading Style Sheet, level 1 och 2 (CSS1, CSS2).

Basics of the XML format

XML facility:	Example:
User-selected tags.	`<book>`, `<songs>`, `<position>` or whatever you need for your data.
Tags can have attributes.	`<book author="Margaret Yorke" title="False Pretences">`

Tags which have no embedded data can be closed in the opening tag.	`<book author="Margaret Yorke" title="False Pretences"/>`
	instead of
	`<book author="Margaret Yorke" title="False Pretences"></book>`

Tags can be nested.	`<book><author>Margaret Yorke</author>...</book>`
Tags must be closed.	Not correct: `<book><author>Margaret Yorke</book>`
Certain special character must be encoded.	`<book title="The &quot;queen&quot;of Sheba"/>`

XML is more strict than accepted HTML practice

HTML browsers accept many kinds of formally illegal HTML encodings. This is not allowed in XML. Examples:

```
Legal: <p>First paragraph.</p><p>Second paragraph</p>
Accepted: <p>First paragraph.<p>Second paragraph</p>

Legal: <b><i>Bold and Italics</i></b>
Accepted: <b><i>Bold and Italics</b></i>

Legal: <FONT COLOR="#FFFF66">
Accepted: <FONT COLOR=#FFFF66>
```

Tags are case-sensitive in XML

```
Illegal: <H1>Heading text</h1>
Legal: <H1>Heading text</H1>
```

White space is sometimes relevant in #PCDATA, but normalized in attributes

```
<CHRISTMAS>
     X
    XXX
   XXXXX
```

```
<CHRISTMAS FATHER="Donald
Duck">
```

is identical to

```
<CHRISTMAS FATHER="Donald Duck">
```

Function	HTML	XML
Set of tags	Built-in, predefined set.	Every application can define its own element types and select their tags.
End tag	Not always required.	Always required.
Case sensitive	No, for example, `<TITLE>` and `<title>` are identical.	Yes, `<TITLE>` and `<title>` are different and `<TITLE>` must end with `</TITLE>`, not with `</title>`.
Coding errors	Often accepted	Not accepted
Support in web browsers	Yes.	Yes in some newer versions.
Text layout and style	HTML tags and style sheets.	Style sheets and XSLT transformation code.

Exercise 41

Here is an example of part of an e-mail heading according to current e-mail standards.

```
From: Nancy Nice <nnice@good.net>
To:   Percy Devil <pdevil@hell.net>
Cc:   Mary Clever <mclever@intelligence.net>,
      Rupert Happy <rhappy@fun.net>
```

How might the same information be encoded using XML?

Exercise 41 solution

```
From: Nancy Nice <nnice@good.net>
To:   Percy Devil <pdevil@hell.net>
Cc:   Mary Clever <mclever@intelligence.net>,
      Rupert Happy <rhappy@fun.net>
```

```
<?xml version="1.0" ?>
<!DOCTYPE header SYSTEM "header.dtd">
<header>
  <from>
     <person>
        <user-friendly-name>Nancy Nice</user-friendly-name>
        <local-id>nnice</local-id><domain>good.net</domain>
     </person></from>
  <to>
     <person>
        <user-friendly-name>Percy Devil</user-friendly-name>
        <local-id>pdevil</local-id><domain>hell.net</domain>
     </person></to>
  <cc>
     <person>
        <user-friendly-name>Mary Clever</user-friendly-name>
        <local-id>mclever</local-id>
        <domain>intelligence.net</domain>
     </person><person>
        <user-friendly-name>rupert happy</user-friendly-name>
        <local-id>rhappy</local-id><domain>fun.net</domain>
     </person></cc>
</header>
```

Special Character Encoding in XML

Reserved character	Predefined entity to use instead
<	<
&	&
>	>
'	'
"	"

Document Type Definition (DTD)

An XML document may be connected with a document type definition. But this is not mandatory, you can send XML data without a DTD.

The DTD describes the allowed syntax, i.e. the tags and their allowed attributes.

Example of a DTD

```
<!ELEMENT book (author+)>
<!ATTLIST book
  title CDATA #REQUIRED
  year CDATA #IMPLIED >
<!ELEMENT author (#PCDATA)>
```

Example of XML using this DTD

```
<?xml version="1.0" ?>
<!DOCTYPE book SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/book.dtd">
<book title="False Pretences" year="1999" >
<author>Margaret York</author>
</book>
```

Relation between DTD and XML

Enviroment:	**“ABNF”**	**“ASN.1”**	**“XML”**
Language for specifying the encodings for a particular application.	ABNF	ASN.1	DTD (but not mandatory, and not as strong typing as in ASN.1)
Language used to actually encode data.	Text, often as a list of lines begin-ning with a name, a colon, follow-ed by a value.	BER (or some other ASN.1 encoding rule)	XML

How to specify the DTD in the XML using it

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

Specifies that this is XML-encoded data

```
<!DOCTYPE person
SYSTEM "person.dtd">
```

Specifies where to find the DTD. "Person.dtd" can be a complete URL, which gives a globally unique reference to this DTD.

```
<PERSON>
  <NAME>John Smith</NAME>
  <BIRTHYEAR>1941</BIRTHYEAR>
  <WAGE>57000</WAGE>
</PERSON>
```

Here comes the XML encoded according to this DTD.

DTD ELEMENT with free text content

Example of a DTD

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

Example 1 of XML using this DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE author SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/author.dtd">
<author>Margaret York</author>
```

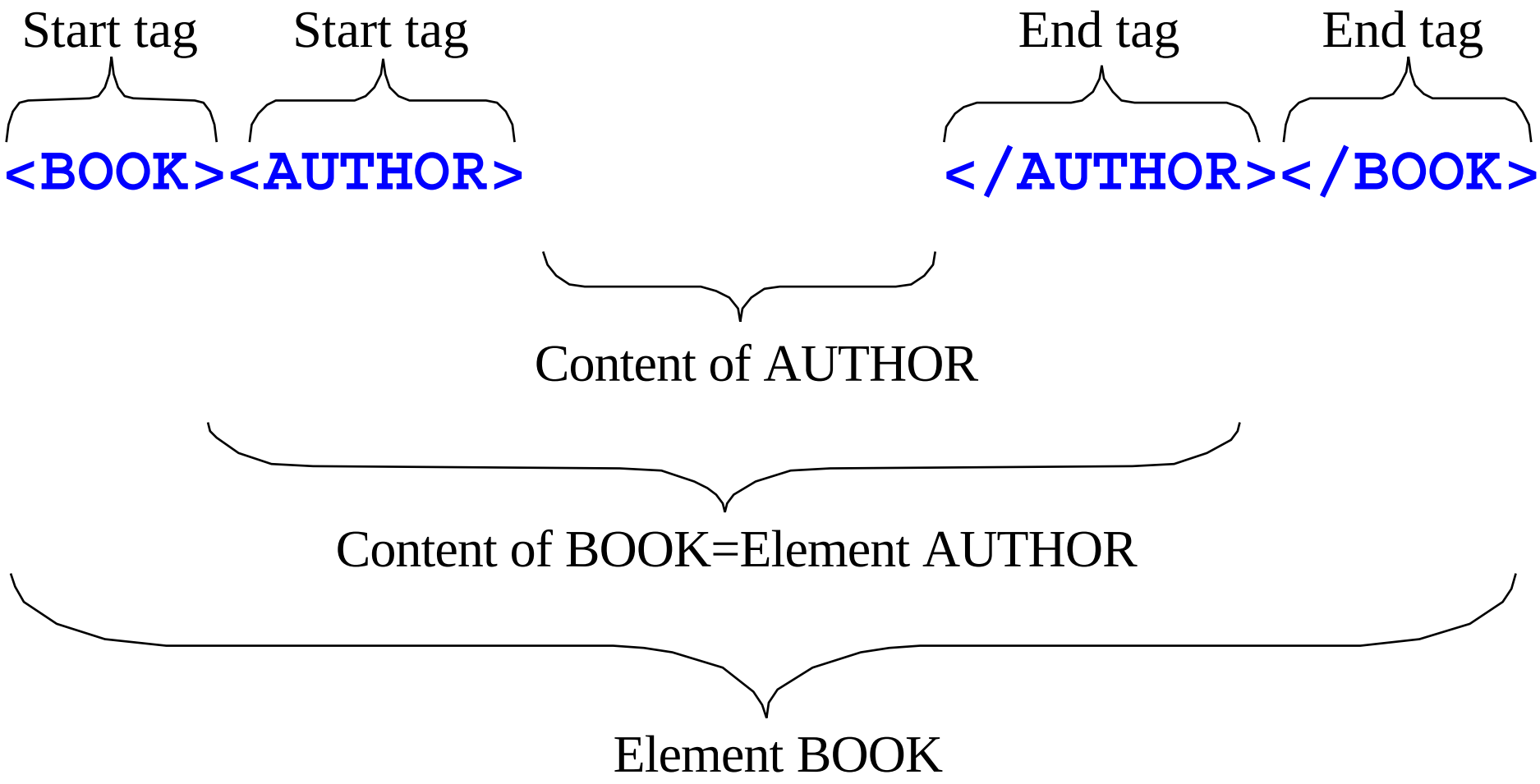
Example 2 of XML using this DTD

```
<author>Text containing &lt; special markup &gt;</author>
```

Example 3 of XML using this DTD

```
<author>
<![CDATA[
Text containing < special markup > like & and " and '
]]>
</author>
```

ELEMENT and TAG



DTD ELEMENT with subelements

`(a,b)` means the element `a` followed by the element `b`.

Example of a DTD

```
<!ELEMENT author (givenname,surname)>
<!ELEMENT givenname (#PCDATA)>
<!ELEMENT surname (#PCDATA)>
```

Example 1 of XML using this DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE author SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/author.dtd">
<author>
<givenname>Margaret</givenname>
<surname>York</surname>
</author>
```

Well-formed = correct XML, but need not have any DTD

Valid = correct XML and in accordance with a specified DTD

DTD ELEMENT with subelements

`(a*)` means that `a` is repeated 0, 1 or more times.

Example of a DTD

```
<!ELEMENT family (father,mother,child*)>
<!ELEMENT father (#PCDATA)>
<!ELEMENT mother (#PCDATA)>
<!ELEMENT child (#PCDATA)>
```

Example 1 of XML using this DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE family SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/family.dtd">
<family>
<father>John</father>
<mother>Margaret</mother>
<child>Eve</child>
<child>Peter</child>
</family>
```

DTD ELEMENT with subelements

`(a+)` means that `a` is repeated 1 or more times.

Example of a DTD

```
<!ELEMENT child-family (father,mother,child+)>
<!ELEMENT father (#PCDATA)>
<!ELEMENT mother (#PCDATA)>
<!ELEMENT child (#PCDATA)>
```

Example 1 of XML using this DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE child-family SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/child-
family.dtd">
<child-family>
<father>John</father>
<mother>Margaret</mother>
<child>Eve</child>
<child>Peter</child>
</child-family>
```

DTD ELEMENT with subelements

`(a?)` means that the element `a` is repeated 0 or 1 times.

Example of a DTD

```
<!ELEMENT basic-family (father?,mother?,child*)>
<!ELEMENT father (#PCDATA)>
<!ELEMENT mother (#PCDATA)>
<!ELEMENT child (#PCDATA)>
```

Example 1 of XML using this DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE basic-family SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/basic-
family.dtd">
<basic-family>
<father>John</father>
<child>Eve</child>
<child>Peter</child>
</basic-family>
```

Exercise 42

Write a DTD for an XML-variant of the e-mail header in Exercise 41.

```
From: Nancy Nice <nnice@good.net>
To: Percy Devil <pdevil@hell.net>
Cc: Mary Clever
    <mclever@intelligence.net>,
    Rupert Happy <rhappy@fun.net>
```

Exercise 42 solution

```
<!ELEMENT header (from, to?, cc?)>
<!ELEMENT from (person)>
<!ELEMENT to (person+)>
<!ELEMENT cc (person+)>
<!ELEMENT person (user-friendly-name,local-id,domain)>
<!ELEMENT user-friendly-name (#PCDATA)>
<!ELEMENT local-id (#PCDATA)>
<!ELEMENT domain (#PCDATA)>
```

Exercise 42

Write a DTD for an XML-variant of the e-mail header in Exercise 41.

```
From: Nancy Nice <nnice@good.net>
To: Percy Devil <pdevil@hell.net>
Cc: Mary Clever
    <mclever@intelligence.net>,
    Rupert Happy <rhappy@fun.net>
```

DTD ELEMENT with subelements

"|" means either-or "," means succession.

EMPTY (without parenthesis) means no contained data.

Example of a DTD

```
<!ELEMENT operations (((get | put),uri)*)>
<!ELEMENT get EMPTY>
<!ELEMENT put EMPTY>
<!ELEMENT uri (#PCDATA)>
```

Example 1 of XML using this DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE operations SYSTEM
"http://dsv.su.se/jpalme/internet-course/xml/operations.dtd">
<operations>
<get/><uri>http://cmc.dsv.su.se/file1</uri>
<get/><uri>http://cmc.dsv.su.se/file2</uri>
<put/><uri>http://cmc.dsv.su.se/file3</uri>
</operations>
```

Note: **<get/>** *is a short form for* **<get></get>**

Exercise 43

Specify DTD and an XML example for a protocol to send either a name (single string), a social-security number (another single string) or both.

Exercise 43 solution page A

DTD specification:	XML examples:
<!ELEMENT id (name | social-security-no | both)> <!ELEMENT both (name, social-security-no)> <!ELEMENT name (#PCDATA)> <!ELEMENT social-security-no (#PCDATA)>	<?xml version="1.0" ?> <!DOCTYPE id SYSTEM "id.dtd"> <id><social-security-no>410201-1410</social-security-no></id>
	<?xml version="1.0" ?> <!DOCTYPE id SYSTEM "id.dtd"> <id><both><name>Eliza Doolittle</name> <social-security-no>410201-1410</social-security-no></both></id>
	<?xml version="1.0" ?> <!DOCTYPE id SYSTEM "id.dtd"> <id><name>Eliza Doolittle</name> </id>

Exercise 43

Specify DTD and an XML example for a protocol to send either a name (single string), a social-security number (another single string) or both.

Exercise 43 solution page B

Note: The following will not work:

```
<!ELEMENT id ( name |
social-security-no |
(name, social-security-
no))>
<!ELEMENT name (#PCDATA)>
<!ELEMENT social-
security-no (#PCDATA)>
```

This will not work, because the receiving program will not be able to know, when it starts to scan <name> whether this is the first or the third branch of the choice.

Operators in lists of subelements:

Code:	Explanation:
`a, b`	Mandatory a followed by mandatory b.
`a | b`	Either a or b.
`a*`	0, 1 or more occurences of a.
`a+`	1 or more occurences of a.
`a?`	0 or one occurences of a.

Any Specification

The ANY specification (example:

`<!ELEMENT miscellaneous ANY>`)

allows any kind of un-specified XML content. This specification should in most cases be avoided, since it makes it difficult for software to check or interpret the content.

DTD ELEMENT with XML attributes

Example of a DTD

```
<!ELEMENT book EMPTY>
<!ATTLIST book
  title CDATA #REQUIRED
  author CDATA 'anonymous'
  weight CDATA #IMPLIED
  format (paper-back | hard-back) 'paper-back'
>
```

Example 1 of XML using this DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE book SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/book.dtd">
<book
  title="False Pretences"
  author="Margaret Yorke"
  format="hard-back"
/>
```

Default values for XML attributes

DTD term:	Example:	Description:
A single value within quotes at the end of the attribute.	`<!ATTLIST book binding (hardback | paperback) "hardback">`	This default value should be assumed if the attribute is not specified in the XML text.
#REQUIRED	`<!ATTLIST book binding (hardback | paperback) #REQUIRED>`	No default value is allowed, the attribute must always be specified in the XML text.
#IMPLIED	`<!ATTLIST book binding (hardback | paperback) #IMPLIED>`	No default value, but the attribute is not required. If the attribute is not given, this might mean that it is unknown or not valid.
#FIXED	`<!ATTLIST book binding (hardback | paperback) #FIXED "hardback">`	The XML can either contain this attribute or not, but if it is there, it must always have this particular value.

Types of XML attributes

Type:	Example:	Description:
CDATA	`<!ATTLIST book title #REQUIRED>`	Any character string.
A list of enumerated values	`<!ATTLIST book binding (hardback | paperback) "hardback">`	Restricted to the listed values only.
ID	`<!ATTLIST book entryno ID #REQUIRED>`	Gives a name to this particular element. No other element in the XML text can have the same name. Unique names on elements are useful in some cases for programs which manipulate the XML text.
IDREF	`<!ATTLIST author authorid ID #REQUIRED>` `<!ATTLIST book authorid IDREF #REQUIRED>`	Reference to the unique name, which was given to another element in the XML text. In the example, every element of type author has an **ID** authorid, and every element of type book has an **IDREF** referring to the **ID** of the element for the author of that book.

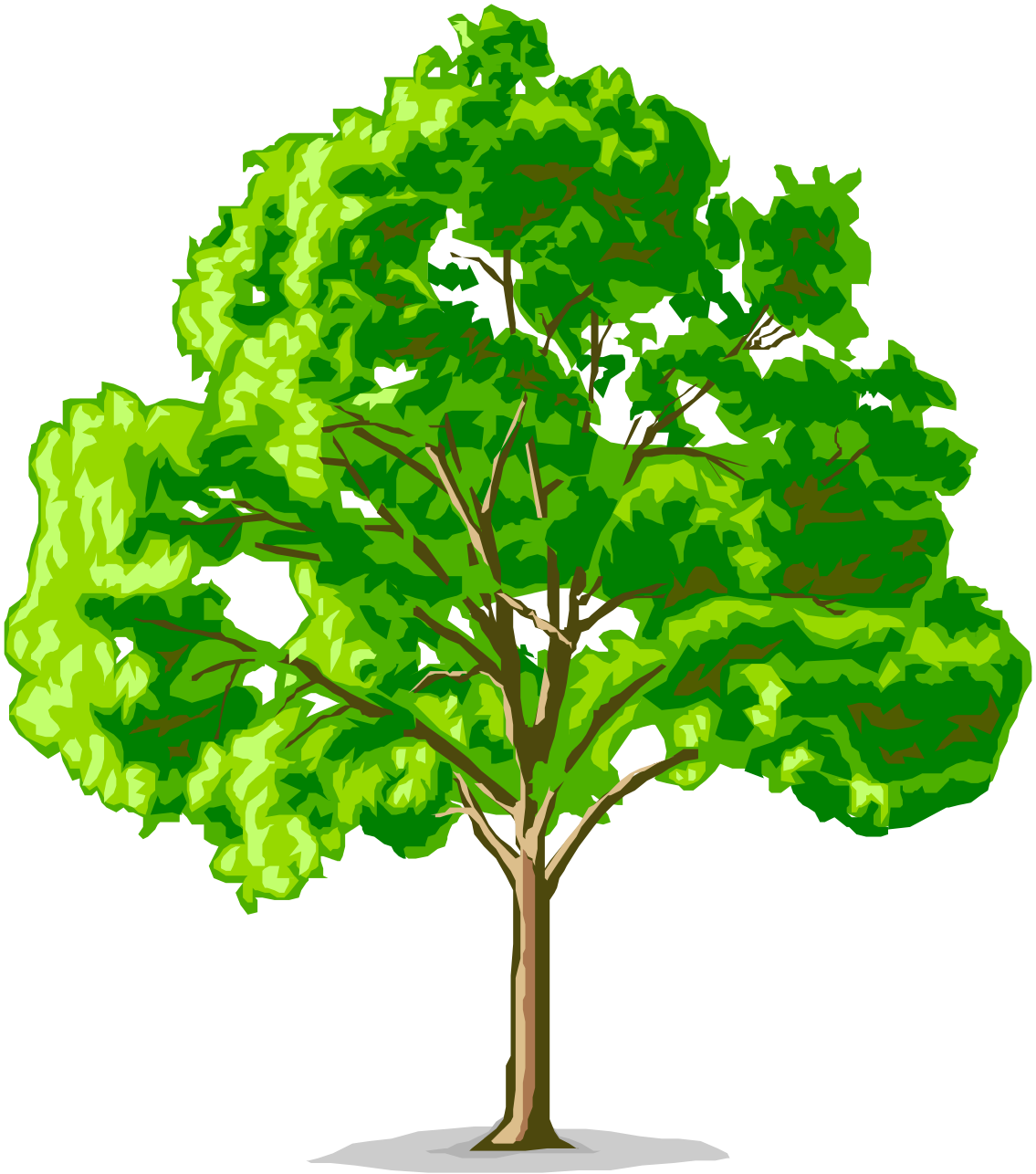
Type:	Example:	Description:
IDREFS	`<!ATTLIST author authorid ID #REQUIRED>` `<!ATTLIST book authorids IDREFS #REQUIRED>`	Similar to **IDREF**, but allows a list of more than one value. Needed in this example, if a book can have more than one author.

ENTITY	DTD text:
	`<!ELEMENT LOGO EMPTY>`
	`<!ATTLIST LOGO GIF-FILE ENTITY #REQUIRED>`
	`<!ENTITY DSV-LOGO SYSTEM "dsv-logo.gif">`
	XML text:
	`<LOGO GIF-FILE="DSV-LOGO"/>`
		This is one way to include binary data in an XML file, by referring to the URI of the binary data. Just like with **<IMG>** tags in HTML, the actual binary file is not included, just referenced.

Type:	Example:	Description:

ENTITIES	DTD text:
	`<!ELEMENT LOGO EMPTY>`
	`<!ATTLIST LOGO GIF-FILE ENTITIES #REQUIRED>`
	`<!ENTITY DSV-LOGO SYSTEM "dsv-logo.gif">`
	`<!ENTITY KTH-LOGO SYSTEM "kth-logo.gif">`
	XML text:
	`<LOGO GIF-FILE="DSV-LOGO KTH-LOGO"/>`
		A list of more than one entity.

NMTOKEN	`<!ATTLIST variable-name #NMTOKEN>`	A name, formatted like a variable name in a computer program. Useful when you use XML to generate source program code.
NMTOKENS	`<!ATTLIST variables #NMTOKENS>`	A list of names, similar as for NMTOKEN above.
NOTATION	`<!ATTLIST SPEECH PLAYER NOTATION ( MP3 | QUICKTIME ) #REQUIRED>`	The name of a non-XML encoding.



Elements versus attributes

```
<book><author><surname>
Yorke</surname><given-
name>Margaret</given-
name></author></book>
```

versus

```
<book author="Margaret
Yorke"/>
```

Elements are like a tree with branches, each branch can split into new branches.

Attributes are like leaves or fruits, they are the end point, cannot be split further. They also give some rudimentary type control.

Exercise 44

Specify DTD and an XML example for a protocol to send a record describing a movie. The record contains a title and a list of people. Each person is identified by the attributes name, and optionally, the attribute role as either actor, photographer, director, author or administrator. As an XML example, use the movie “The Postman Always Rings Twice”, directed by Tay Garnet based on a book by James M. Cain with leading actors Lana Turner and John Garfield.

Exercise 44 solution

DTD specification:

```
<!ELEMENT movie (title,
  person+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT person EMPTY>
<!ATTLIST person
  name CDATA #REQUIRED
  role (actor | photographer |
  director | author |
  administrator)
  #IMPLIED
>
```

Exercise 44

Specify DTD and an XML example for a protocol to send a record describing a movie. The record contains a title and a list of people. Each person is identified by the attributes name, and optionally, the attribute role as either actor, photographer, director, author or administrator. As an XML example, use the movie “The Postman Always Rings Twice”, directed by Tay Garnet based on a book by James M. Cain with leading actors Lana Turner and John Garfield.

XML data:

```
<?xml version="1.0" ?>
<!DOCTYPE movie SYSTEM
  "movie.dtd">
<movie>
<title>
The Postman Always Rings
Twice</title>
<person name="Lana Turner"
   role="actor"/>
<person name="John Garfield"
   role="actor"/>
<person name="Tay Garnet"
   role="director"/>
<person name="James M. Cain"
   role="author"/>
</movie>
```

ENTITIES

Built-in character entities

Example: `&quot; &`

Internal entities

You can add your own additional entity declarations to represent characters or sequences of characters. For example:

```
<!ENTITY KTH "Kungliga Tekniska Högskolan">
<DESCRIPTION>&KTH; is a technical university.</DESCRIPTION>
```

is identical to

```
<DESCRIPTION>Kungliga Tekniska Högskolan is a technical
university.</DESCRIPTION>
```

External entities

```
<!ENTITY polisvåld SYSTEM
"http://www.palme.nu/free/pv.html">

<!ENTITY comic SYSTEM
"http://www.palme.nu/comics/a-11.gif" NDATA GIF87A>
```

Use of entities to reference external DTD files

Example of the DTD book.dtd

```
<!ELEMENT book EMPTY>
<!ATTLIST book
  title CDATA #REQUIRED author CDATA 'anonymous'
  weight CDATA #IMPLIED
  format (paper-back | hard-back) 'paper-back' >
```

Example of the DTD collection.dtd

```
<!ENTITY % book SYSTEM "book.dtd">
 %book;
 <!ELEMENT collection (book+)>
 <!ATTLIST collection owner CDATA #REQUIRED >
```

Example of XML using these DTDs

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE collection SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/collection.d
td">
<collection
  owner="Kungliga Biblioteket"
>
<book
  title="False Pretences"
  author="Margaret Yorke"
  format="hard-back"
/>
<book
  title="Act of Violence"
  author="Margaret Yorke"
  format="paper-back"
/>
</collection>
```

IDs in XML

Unique names can be used to refer between different places in a document.

XML example:

```
<author ref="myorke">Margaret Yorke</author>
...
<book author="myorke">False Pretences</book>
```

Based on the DTD:

```
<!ELEMENT author (#PCDATA)>
<!ATTLIST author
  ref ID #REQUIRED>
<!ELEMENT book (#PCDATA)>
<!ATTLIST book
  author IDREF #IMPLIED>
```

Attribute types:

ID = Name of this object

IDREF = One single ID reference

IDREFS = List of names separated by white space

NMTOKEN, NMTOKENS = Single words or lists of words separated by white space

Name Spaces

Part of the war DTD:

```
<!ELEMENT war:desert
(deserter*)>
<!ELEMENT war:deserter
(#PCDATA)>
```

Part of the geography DTD:

```
<!ELEMENT geography:desert
(#PCDATA)>
```

Use of these two DTDs in a new DTD desertaions-in-deserts:

```
<!ENTITY % war:desert SYSTEM
"war.dtd">
%war;
<!ENTITY % geography:desert
SYSTEM "geography.dtd">
%geography;
<!ELEMENT desertations-in-
deserts (war:desert,
geography:desert)>
<!ATTLIST desertaions-in-deserts
xmlns:war CDATA #IMPLIED
xmlns:geography CDATA #IMPLIED>
```

XML data:

```
<?xml version="1.0" ?>
<!DOCTYPE desertations-in-deserts
SYSTEM "desertations-in-
deserts.dtd">
<desertations-in-deserts
xmlns:war="http://dsv.su.se/jpalm
e/a-book/xml/war.dtd"
xmlns:geography="http://dsv.su.se
/jpalme/a-
book/xml/geography.dtd">
  <war:desert>
    <deserter>
    John Smith</deserter>
  </war:desert>
  <geography:desert>
    Sahara</geography:desert>
</desertations-in-deserts>
```

The `xmlns:war="http://dsv.su.se/jpalme/a-book/xml/war.dtd"` and `xmlns:geography="http://dsv.su.se/jpalme/a-book/xml/geography.dtd"` attributes need not refer to any real file, but should contain a unique URL for this name space.

Putting binary data into XML encodings

All textual encodings have a common problem in that they will not allow binary data, like, for example, a picture in GIF format. There are three ways of handling this problem in XML:

① Encode the binary data, using, for example, the BASE64 method.

② Put the binary data in a separate file, like GIF pictures in HTML:

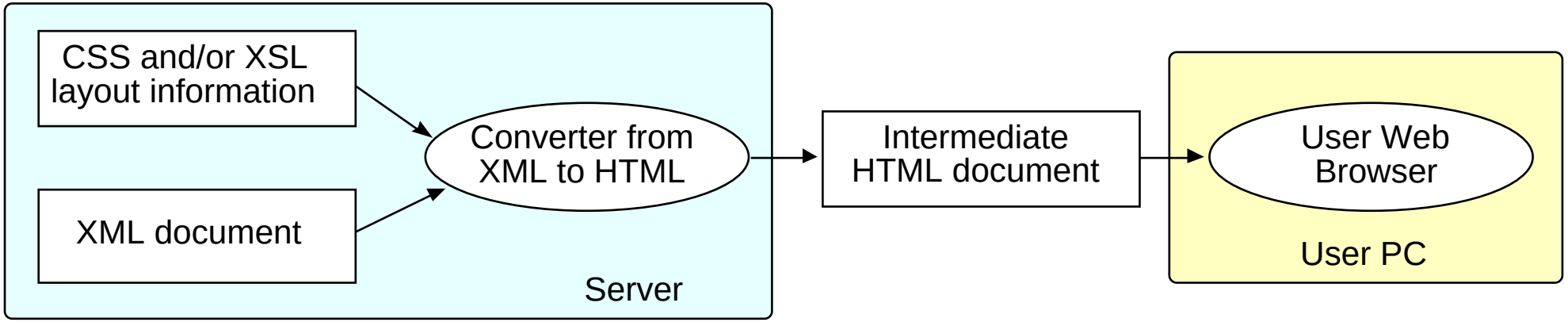
```
<IMG SRC="image.gif">
```

③ Use method ②, but combine it with the MHTML method to concatenate all the files into a single compound file.

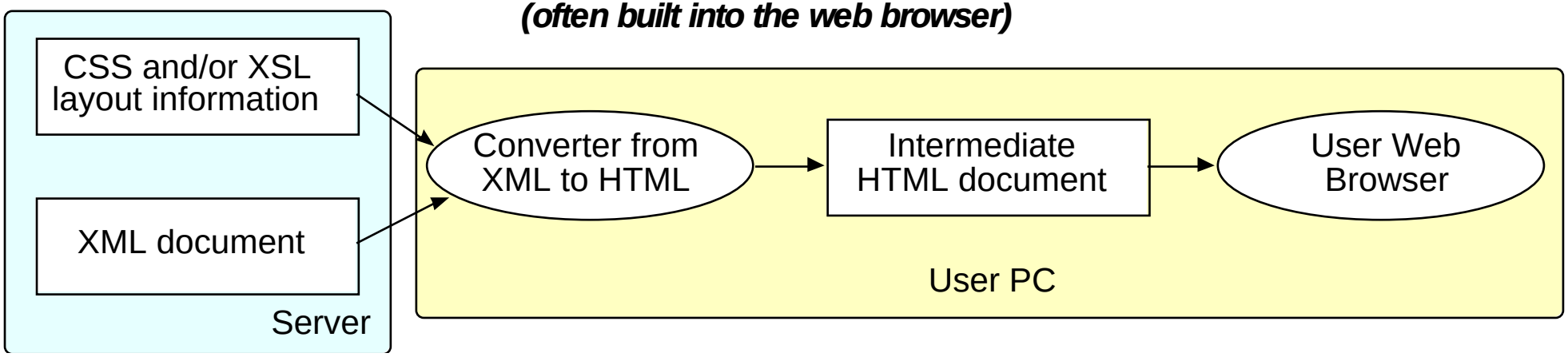
Putting formatting information into XML pages

CSS = Cascading Style Sheets and XSLT = Extensible Style Language Transformations

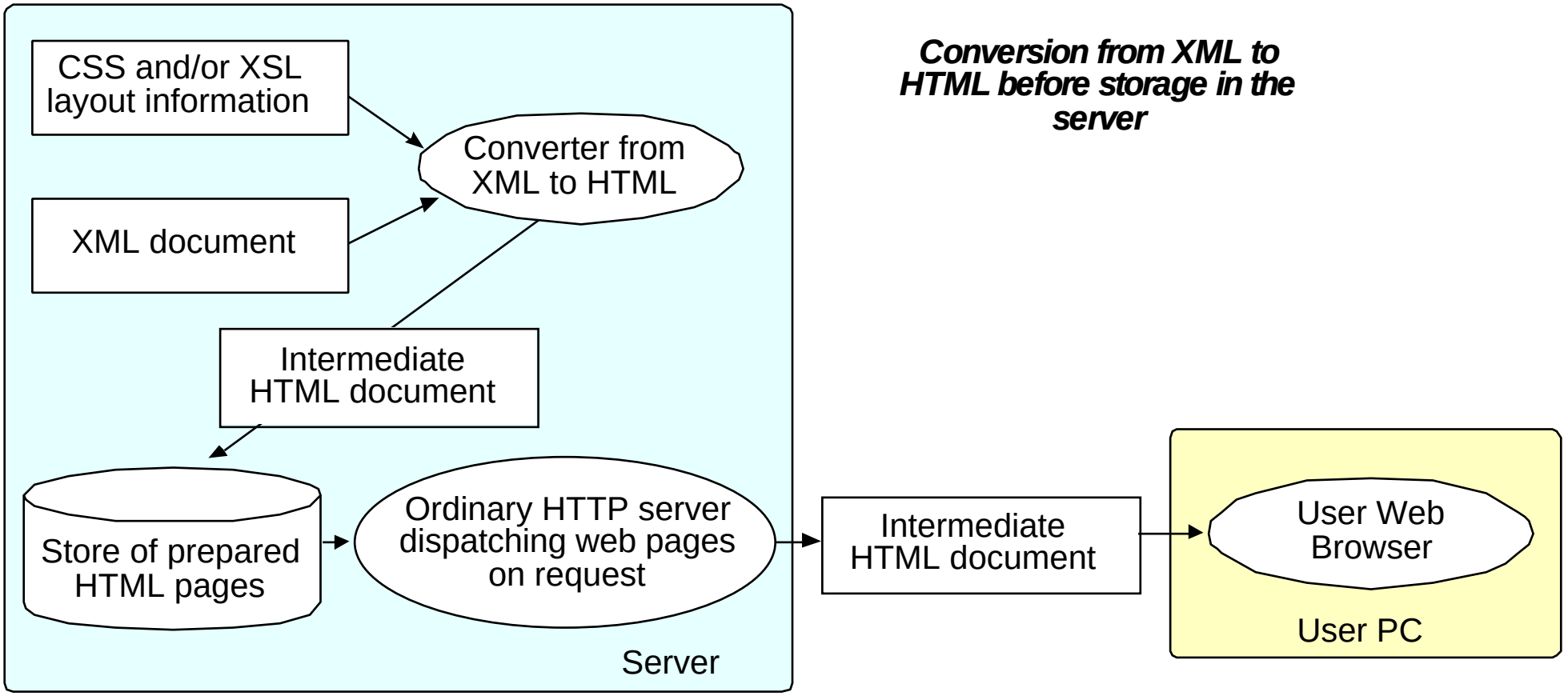
Conversion from XML to HTML in the server, before transmission to the PC



Sending XML to the PC and conversion in the PC (often built into the web browser)



Putting formatting information into XML pages



Conversion from XML to HTML before storage in the server

File ticket.css:

```
TITLE{    position: absolute; width: 121px; height: 31px; top:25px; left: 86px;
          font-family: Verdana, sans-serif; font-size: 24pt; font-weight: bold}
CLASS{    position: absolute; width: 106px; height: 15px; top: 115px; left: 13px;
          font-family: Verdana, sans-serif; font-size: 12pt; font-weight: bold }
FROM {    position: absolute; width: 150px; height: 15px; top: 70px; left: 12px;
          font-family: Verdana, sans-serif; font-size: 14pt; font-weight: bold }
TO   {    position: absolute; width: 150px; height: 15px; top: 70px; left: 166px;
          font-family: Verdana, sans-serif; font-size: 14pt; font-weight: bold; }
DEPART    {    position: absolute; width: 142px; height: 15px; top: 95px; left: 11px;
          font-family: Verdana, sans-serif; font-size: 10pt }
ARRIVE    {    position: absolute; width: 128px; height: 15px; top: 95px; left: 167px;
          font-family: Verdana, sans-serif; font-size: 10pt }
CABIN{    position: absolute; width: 138px; height: 18px; top: 115px; left: 167px;
          font-family: Verdana, sans-serif; font-size: 12pt; font-weight: bold }
SEAT {    position: absolute; width: 138px; height: 18px; top: 115px; left: 247px;
          font-family: Verdana, sans-serif; font-size: 12pt; font-weight: bold }
```

File ticket.xml:

```
<?xml version="1.0" ?>
<!DOCTYPE TICKET SYSTEM "ticket.dtd">
<?XML:stylesheet type="text/css"
href="ticket.css" ?>
<TICKET><TITLE>TICKET</TITLE>
<CLASS>2 Class</CLASS>
<FROM>Oslo</FROM>
<TO>Stockholm</TO>
<DEPART>Mon 13 Jan 12:13</DEPART>
<ARRIVE>Mon 13 Jan 18:45</ARRIVE>
<CABIN>Cabin 3</CABIN>
<SEAT>Seat 55</SEAT></TICKET>
```

Visual rendering:

TICKET

Oslo **Stockholm**

Mon 13 Jan 12:13 Mon 13 Jan 18:45

2 Class **Cabin 3** **Seat 5**

XML validation

When you are developing specifications using DTD and XML, it is essential to be able to check your specifications for correctness. There is software available to do this. I have been using the validator on the net at http://www.stg.brown.edu/service/xmlvalid/ to validate the examples given in this course.

XHMTL

XHTML is a variant of HTML which is at the same time also correct XML. The main differences from ordinary HTML are:

- All tags must be lower case, e.g. **<a href>** and not **<A HREF=>**
- All tags must be ended, e.g. First paragraph
second line.
- No syntax errors allowed, e.g. not ** ** Strong text ** **

ABNF specification:	ASN.1 specification:	DTD specification:

Family = "Family"
CRLF *(Person)
"End of Family"
 	
Person = "Person" CRLF
" Name: " 1*A CRLF
" Birthyear: " 4D
CRLF
" Gender: "
("Male"/"Female") CRLF
" Status: "
("unmarried"/
"married"/
"divorced"/ "widow"/
"widower")
	Family ::= SEQUENCE OF Person
	
	**Person ::= SEQUENCE {
	name VisibleString,
	birthyear INTEGER,
	gender Gender,
	status Status }**
	
	**Gender ::= ENUMERATED {
	male(0), female(1) }**
	
	**Status ::= ENUMERATED {
	unmarried(0), married(1),
	divorced(2), widow(3), widower(4)
	}**
		<!ELEMENT family
		(person+)>
		<!ELEMENT person (name,
		birthyear)>
		<!ELEMENT name (#PCDATA)>
		<!ELEMENT birthyear
		(#PCDATA)>
		<!ATTLIST person
		gender (male | female)
		#REQUIRED
		status (unmarried |
		married | divorced |
		widow | widower)
		#REQUIRED
		>

Example of textual encoding:	Example of BER encoding:	Example of XML encoding:

Family
Person
  Name: John Smith
  Birthyear: 1958
  Gender: Male
  Status: Married
Person
  Name: Eliza Tennyson
  Birthyear: 1959
  Gender: Female
  Status: Married
End of Family
	(Each box represents one octet. Two-character codes are hexadecimal numbers, one character codes are characters)
	30 34
	30 16
	1A 0A J o h n ␣ S m i t h
	02 02 07 A6
	0A 01 00
	0A 01 01
	30 1A
	1A 0E E l i z a ␣ T e n n y s o n
	02 02 07 A7
	0A 01 01
	0A 01 01
		<?xml version="1.0" ?>
		<!DOCTYPE family SYSTEM "family.dtd">
		<family>
		 <person gender="male"
		  status="married">
		  <name>John Smith</name>
		  <birthyear>1958
		  </birthyear>
		 </person>
		 <person gender="female"
		  status="married">
		  <name>Eliza
		  Tennyson</name>
		  <birthyear>1959
		  </birthyear>
		 </person>
		</family>

169 octets (excluding newlines)	54 octets	276 octets (excluding newlines and leading spaces)
18 % efficiency	57 % efficiency	11 % efficiency[1]

The PER (unaligned variant) encoding of the same ASN.1 and the same data would be the following 31 octets:

```
00000010   (number of persons in family)
00001010   (10 characters)
 1001010   J
1 101111   o
11 01000   h
110 1110   n
0100 000
10100 11   S
110110 1   m
1101001    i
 1110100   t
1 101000   h
00 000010  (2 octets)
00 00011110   100110   (1958)
0          (male)
 0 01      (married)
```

```
000011 10  (14 characters)
100010 1   E
1101100    l
 1101001   i
1 111010   z
11 00001   a
010 0000
1010 100   T
11001 01   e
110111 0   n
1101110    n
1111001    y
1 110011   s
11 01111   o
110 1110   n
0000 0010    (2 bytes)
0000 01111010 0111   (1959)
1          (female)
001        (married)
```

Comparison of ABNF, ASN.1-BER and DTD-XML

	ABNF	ASN.1	DTD+XML
Level	Low level, almost any text.	High level, strongly typed.	High level, but not as good typing facilities as ASN.1.
Encoded form.	Text.	BER,. PER, etc.	Text.
Readability of metalanguage	OK.	Good.	Acceptable.
Readability of encoded data	Very good.	Very bad unless special reader program is used.	Very good.
Efficiency of data packing.	Usually not so good.	About 50 % with BER, almost 100 % with PER.	Not so good.
Binary data	Must be encoded, for example using BASE64.	Can easily be included as is.	Must be encoded, for example using BASE64, or sent as separate files.
Layout facilities	None, but the high freedom allows specification of rather readable formats.	None.	Can be combined with layout languages.

Other Encoding Languages

ABNF, ASN.1 and XML are not the only encoding languages. Some other existing languages are Corba and XDR (External Data Representation). Corba is more programmer-oriented, and provides a programming API for transmission of data between applications running on different hosts. And some protocols, for example the Domain Naming System (DNS) do not use any encoding language at all, their encodings are specified in the form of English-language text and tables.

More information about XML

The official XML standards specification (rather difficult to read):

`http://www.w3.org/TR/REC-xml`

Norman Walsh's XML tutorial:

`http://www.xml.com/xml/pub/98/10/guide1.html`

Rolf Pfeiffer's XML tutorial:

`http://www.software.ibm.com/developer/education/tutorial-prog/abstract.html`

Doug Tidwell's XML tutorial:

`http://www.software.ibm.com/developer/education/xmlintro/`

Validator of DTD/XML encodings:

`http://www.stg.brown.edu/service/xmlvalid/`

XML books, like for example:

XML Bible, by Eliott Rusty Harold, IDG Books, Foster City, CA, U.S.A., 1999.