SGG 3643

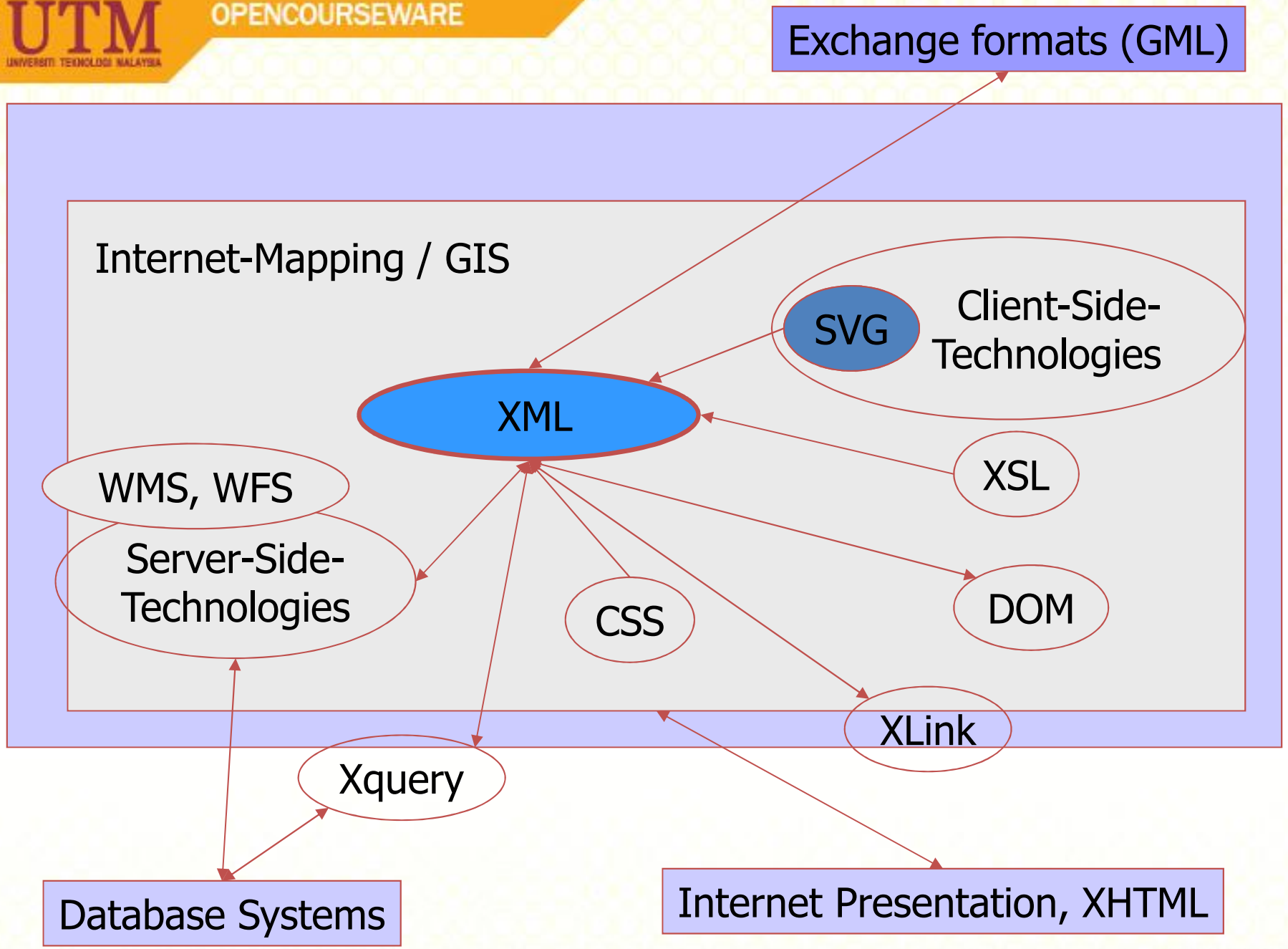
Computer Programming III

XML & XSLT

Ivin Amri Musliman

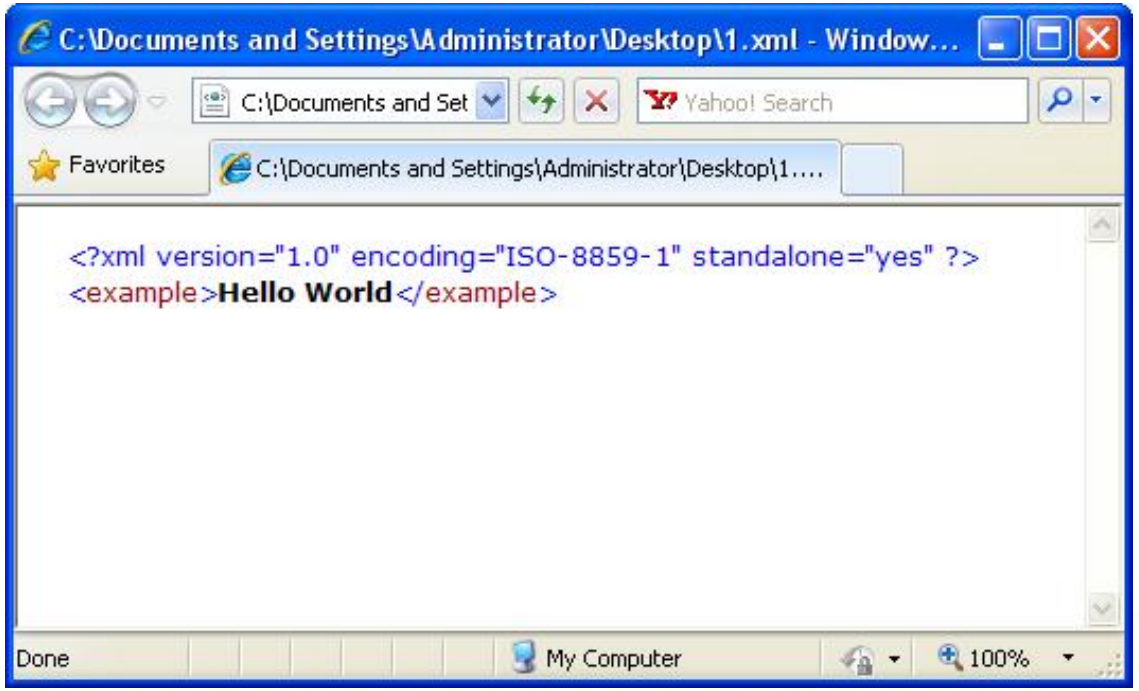
Outline

- Introduction
- XML as a family of standards
- Structure and content of XML documents
- Document Type Definition (DTD)
- Presentation:
  - XML and CSS
  - XML and XSL
- XSLT


Exchange formats (GML)
Internet-Mapping / GIS
SVG
Client-Side-Technologies
XML
WMS, WFS
XSL
Server-Side-Technologies
CSS
DOM
XLink
Xquery
Database Systems
Internet Presentation, XHTML

The usual example

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



XML – What is it?

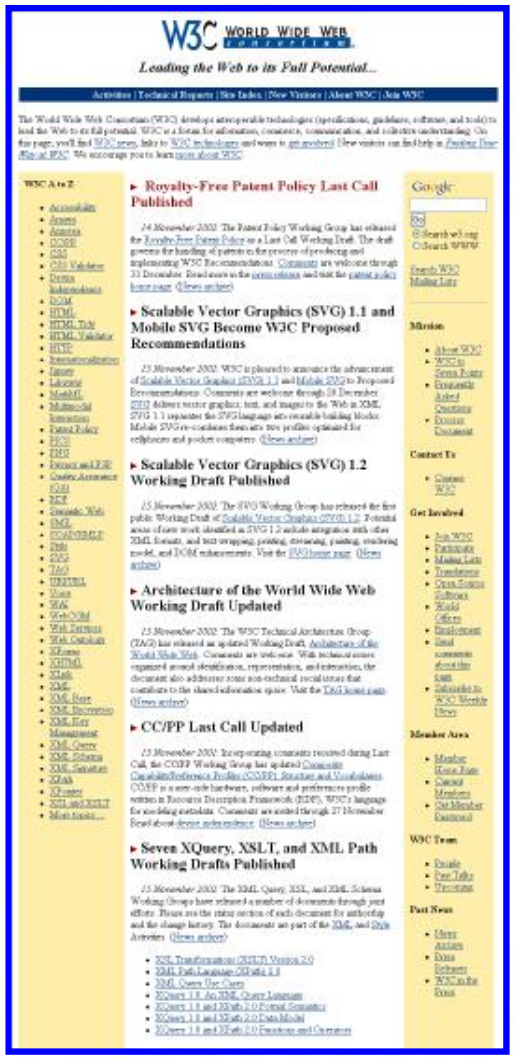
- XML stands for e*X*tensible *M*arkup *L*anguage
- XML was designed to describe and to contain data.
- XML is a markup language much like HTML, but XML tags are not predefined in XML. You must define your own tags
- XML uses a Document Type Definition (DTD) or an XML Schema to define the structure of the data.
- XML with a DTD or XML Schema is designed to be self-descriptive.

Differences between XML and HTML

- XML and HTML were designed with different goals:
  - XML was designed to describe data and to focus on what data is.
  - HTML was designed to display data and to focus on how data looks.
  - HTML is about displaying information, while XML is about describing information.

References

- World Wide Web Consortium, XML
  http://www.w3c.org/XML/
- W3C Schools
  http://www.w3schools.com/xml/xml_elements.asp
- XSL-Processor
  http://saxon.sourceforge.net
- Beginning XML Part 1 - Well-formed XML,
  http://tutorials.findtutorials.com/read/category/81/id/100



XML and GIS

- XML is often used for the description of metadata.
- Open Geospatial Consortium (OGC)
  - Geography Markup Language (GML)
  - XML-based messaging: GetCapabilities, GetFeature.
- LandXML.org
- World Wide Web Consortium (W3C)
  - Scalable Vector Graphics (SVG)
- ESRI: Arc eXtensible Markup Language (ArcXML), the file format ArcIMS uses for communication between ArcIMS components.

Example 1: WMS Capabilities

```
<?xml version='1.0' encoding="UTF-8" standalone="no" ?>
<!DOCTYPE WMT_MS_Capabilities SYSTEM
"http://www.digitalearth.gov/wmt/xml/capabilities_1_1_1.dtd"
[
<!ELEMENT VendorSpecificCapabilities EMPTY>
]> <!-- end of DOCTYPE declaration -->
<WMT_MS_Capabilities version="1.1.1" updateSequence="0">
<!-- Service Metadata -->
<Service>
<!-- The WMT-defined name for this type of service -->
<Name>OGC:WMS</Name>
...
```

Examples 2: GML

```
<?XML Version="1.0"?>
<mke:street, fid="3490">
  <mke:majorRoad>
  <mke:streetName>Jalan Tun Abdul Razak</mke:streetName>
  <mke:speedLimit>80</mke:speedLimit>
  <mke:numberLanes>6</mke:numberLanes>
    <gml:centerLineOf>
      <gml:LineString srsName="EPSG:4326">
        <gml:coordinates>5.5,80.0 60.5,130.5</gml:coordinates>
      </gml:LineString>
    </gml:centerLineOf>
  <mke:majorRoad />
</mke:street>
```

Schema Definitition (.xsd file)

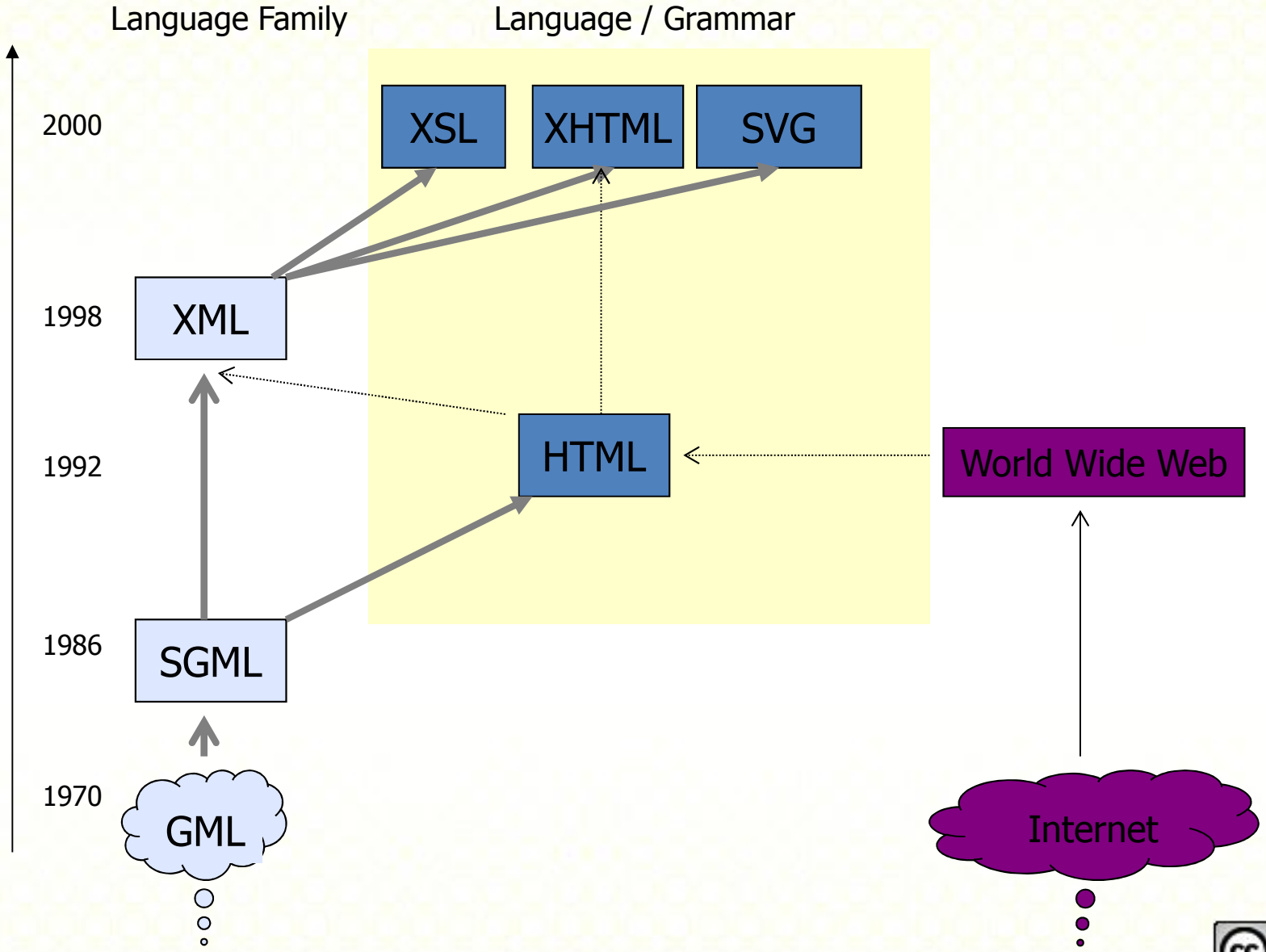
GML (.xml or .gml file)

```
<LineString srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <coordinates>100.0,100.0 230.0,80.0 350.0,130.0 </coordinates>
</LineString>
```

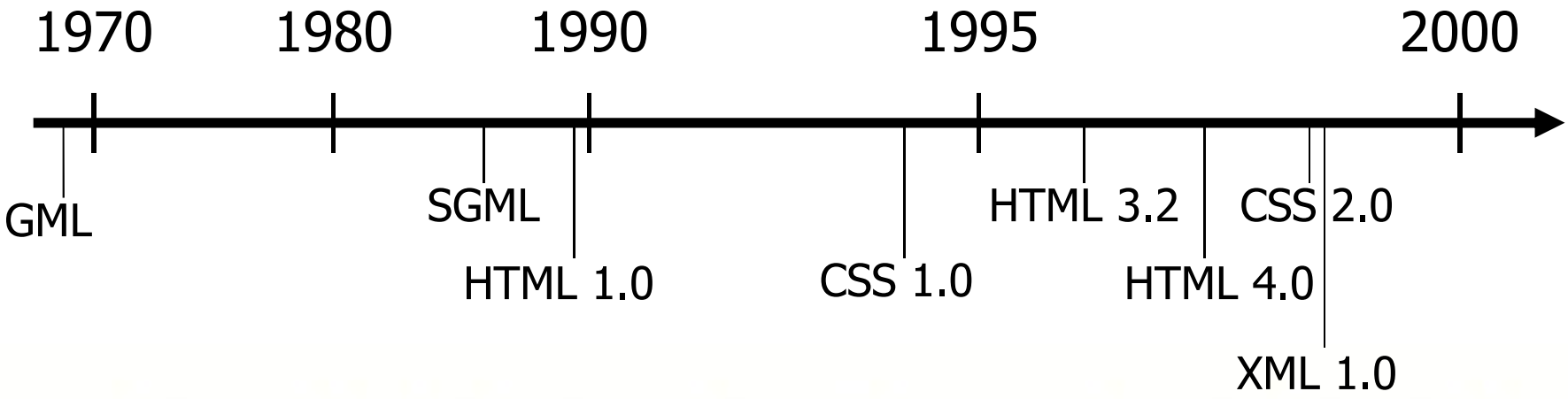
XML - Pro

- XML open, highly accepted standard.
- XML supports clear distinction between data and presentation.
- XML is text oriented.
- XML is extensible.
- XML is self-descriptive.
- XML can be localized, i. e. it can be used with different fonts.
- XML is independent of hardware plattforms and independent of programming languages.
- XML can be evaluated by computers.
- XML is suited for data storage **à** XML databases.
- XML can be easily transformed (using XSL).

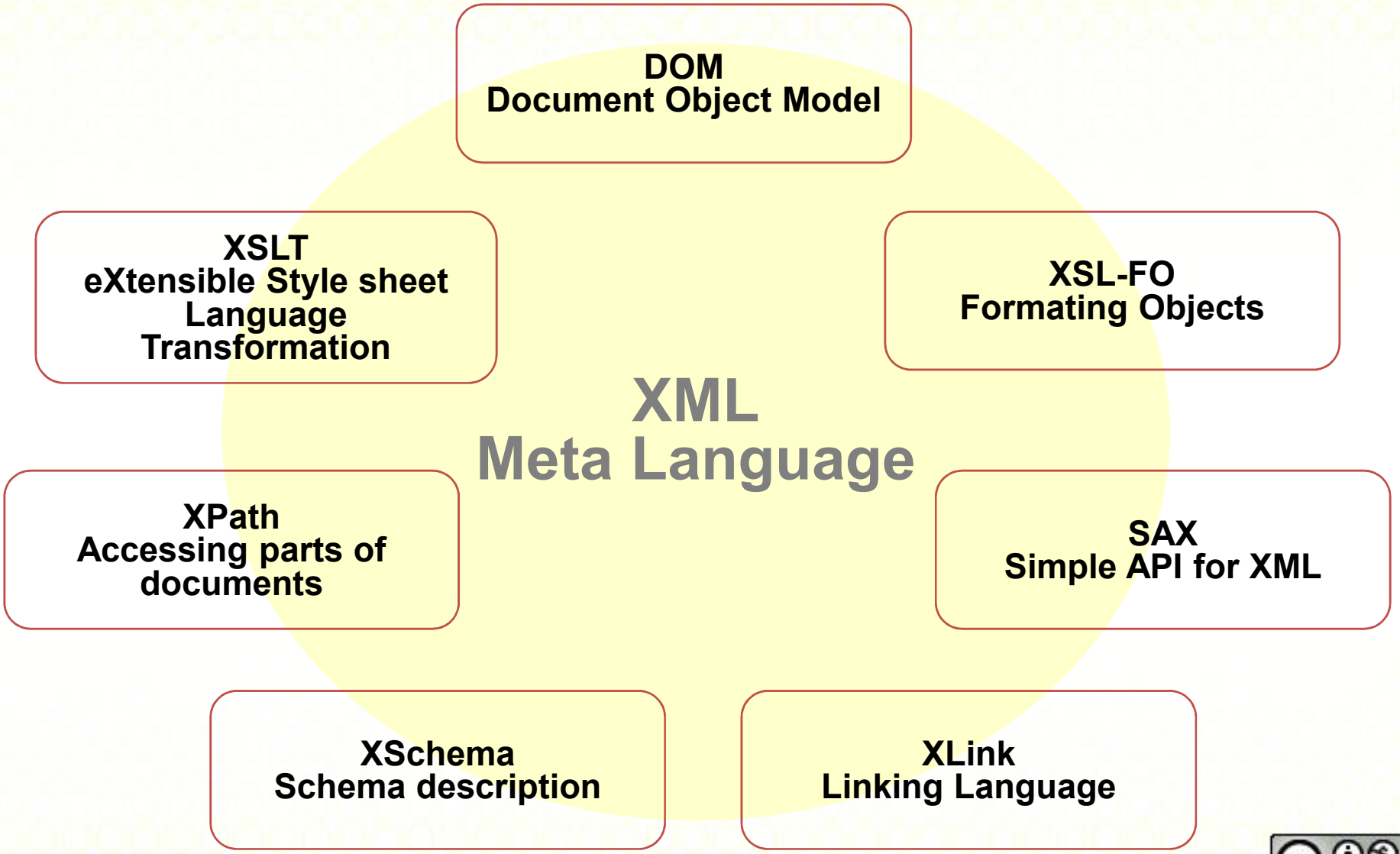
History



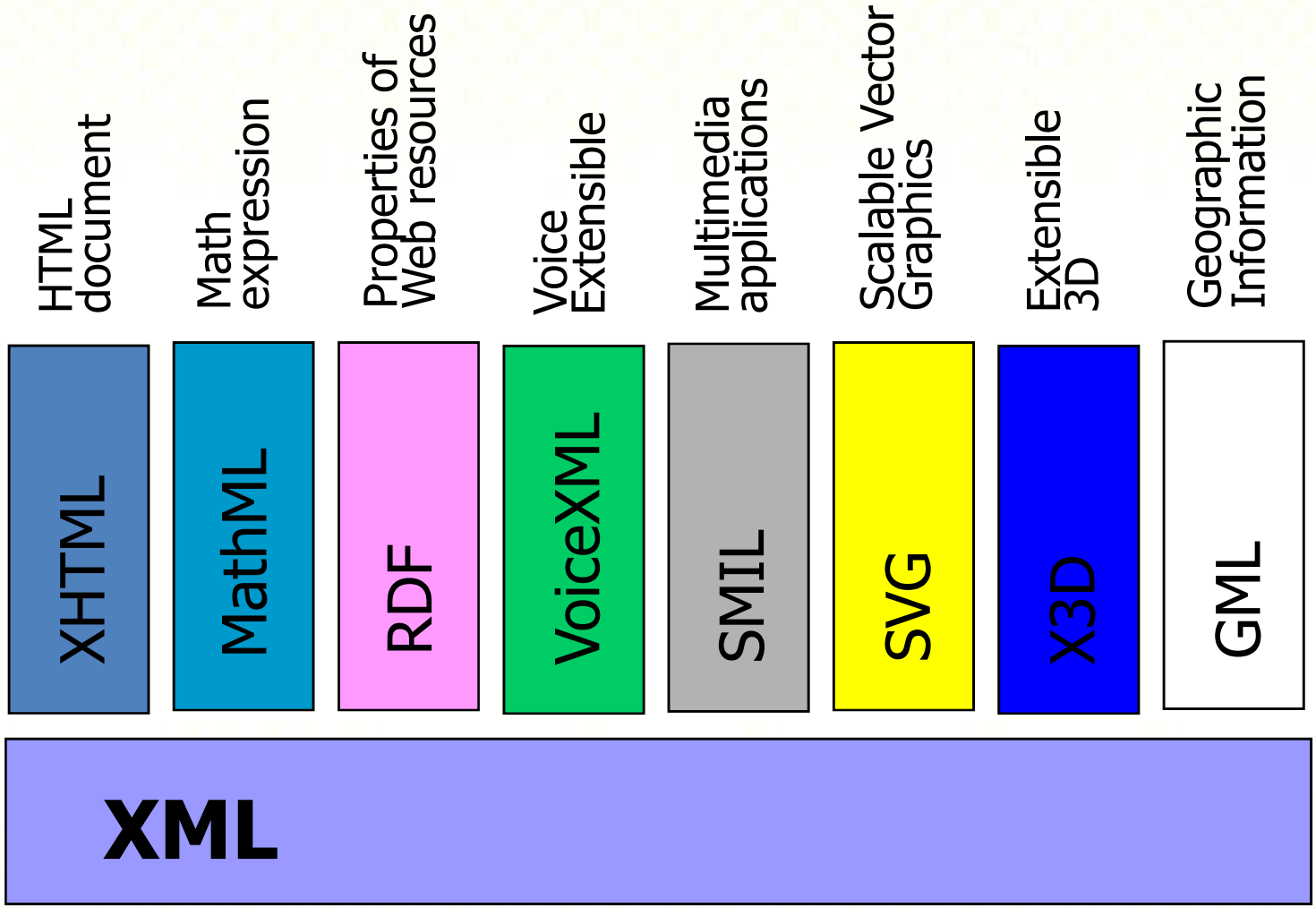
Timeline



XML is a Family of standards and technologies



XML Applications



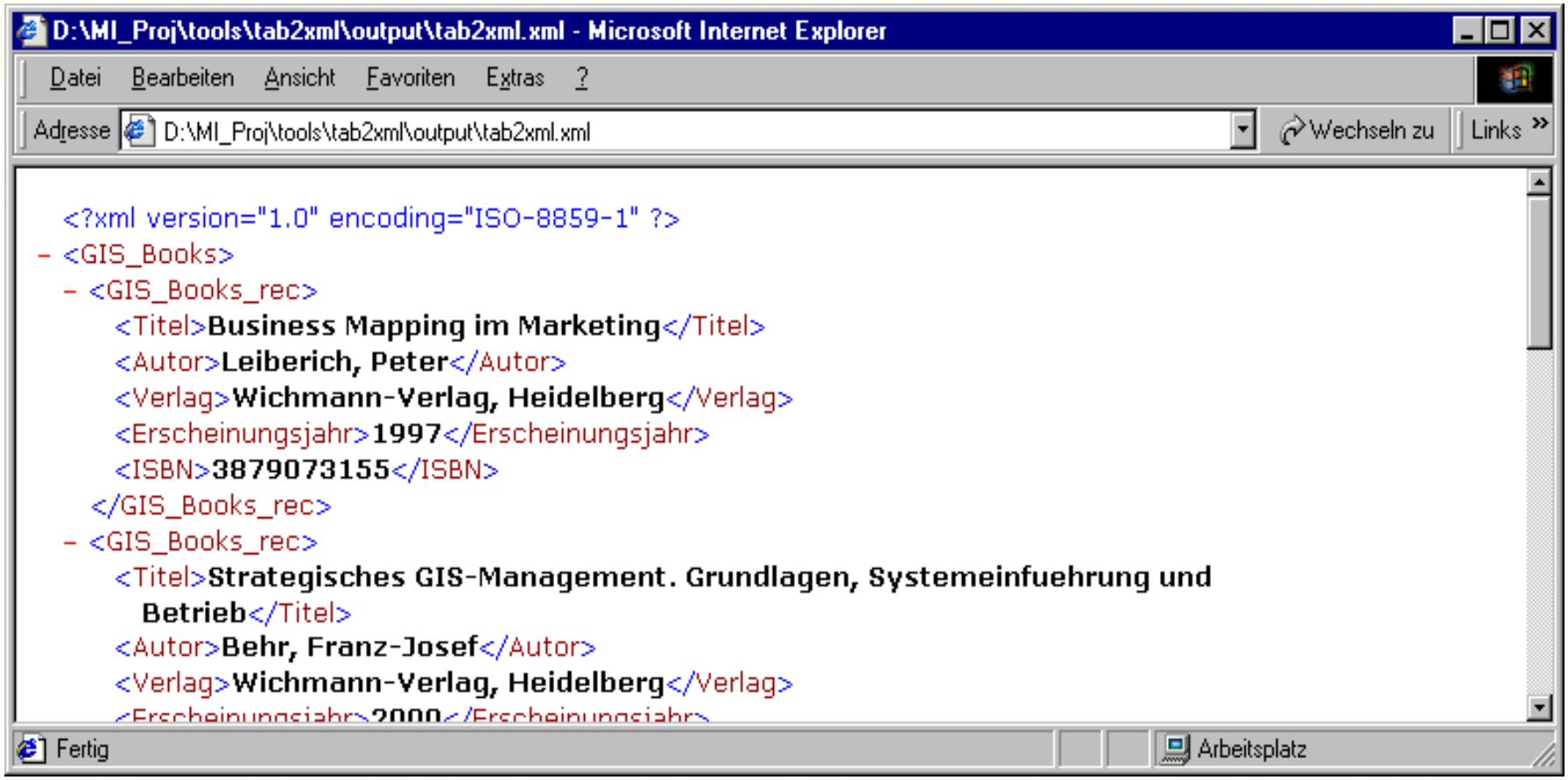
XML in Internet Explorer

- Internet Explorer (from version 5.x) supports most of the international standards for XML 1.0 and the XML DOM (Document Object Model).
  - Viewing of XML documents
  - Full support for W3C DTD standards
  - XML embedded in HTML as Data Islands
  - Binding XML data to HTML elements
  - Transforming and displaying XML with XSL
  - Displaying XML with CSS
  - Access to the XML DOM

Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<GIS_Books>
<GIS_Books_rec>
   <Titel>Business Mapping im Marketing</Titel>
   <Autor>Leiberich, Peter</Autor>
   <Verlag>Wichmann-Verlag, Heidelberg</Verlag>
   <Erscheinungsjahr>1997</Erscheinungsjahr>
   <ISBN>3879073155</ISBN>
</GIS_Books_rec>
<GIS_Books_rec>
   <Titel>Strategisches GIS-Management. Grundlagen,
   Systemeinfuehrung und Betrieb</Titel>
   <Autor>Behr, Franz-Josef</Autor>
   <Verlag>Wichmann-Verlag, Heidelberg</Verlag>
   <Erscheinungsjahr>2000</Erscheinungsjahr>
   <ISBN>3879073503</ISBN>
</GIS_Books_rec>
...
</GIS_Books>
```

XML document in Internet Explorer



XML file with CSS

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml:stylesheet type="text/css" href="tab2xml.css" ?>
<GIS_Books>
<GIS_Books_rec>
   <Title>Internet GIS</Title>
   <Author>Peng Tsou</Author>
   <Publisher>Wiley, UK</Publisher>
   <PublicationDate>1999</PublicationDate>
   <ISBN>3-87907-325-2</ISBN>
</GIS_Books_rec>
...
```

CSS-File

```
Title {display: block;
    font-family:Verdana, Arial;
    font-size:10pt;
    color:blue;
    font-weight:bold;
    text-align:left;
    margin-top:1cm;
    margin-left:2cm;
    text-decoration:underline;}
Author {display: block;
    margin-top:0.1cm;
    margin-left:2cm;
    margin-right:2cm;
    margin-bottom:0cm;
    font-family:Verdana, Arial;
padding:0pt; }
...
```

Cascading Style Sheets, level 2
CSS2 Specification: http://www.w3.org/TR/REC-CSS2/

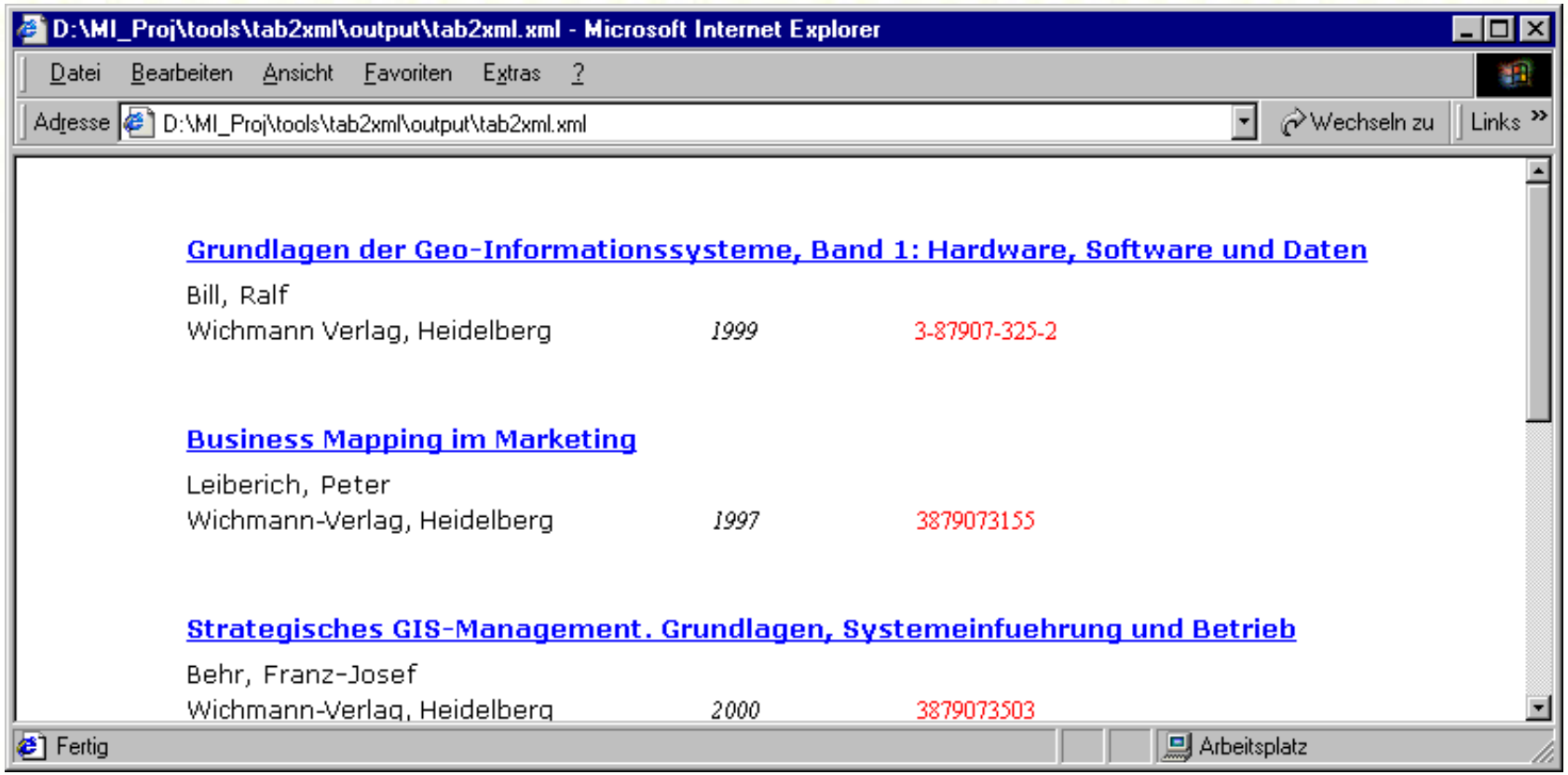
Corresponding CSS file

- Element are selected using selectors
- Presentation is defined using style properties as in HTML
- CSS syntax is not XML compliant!

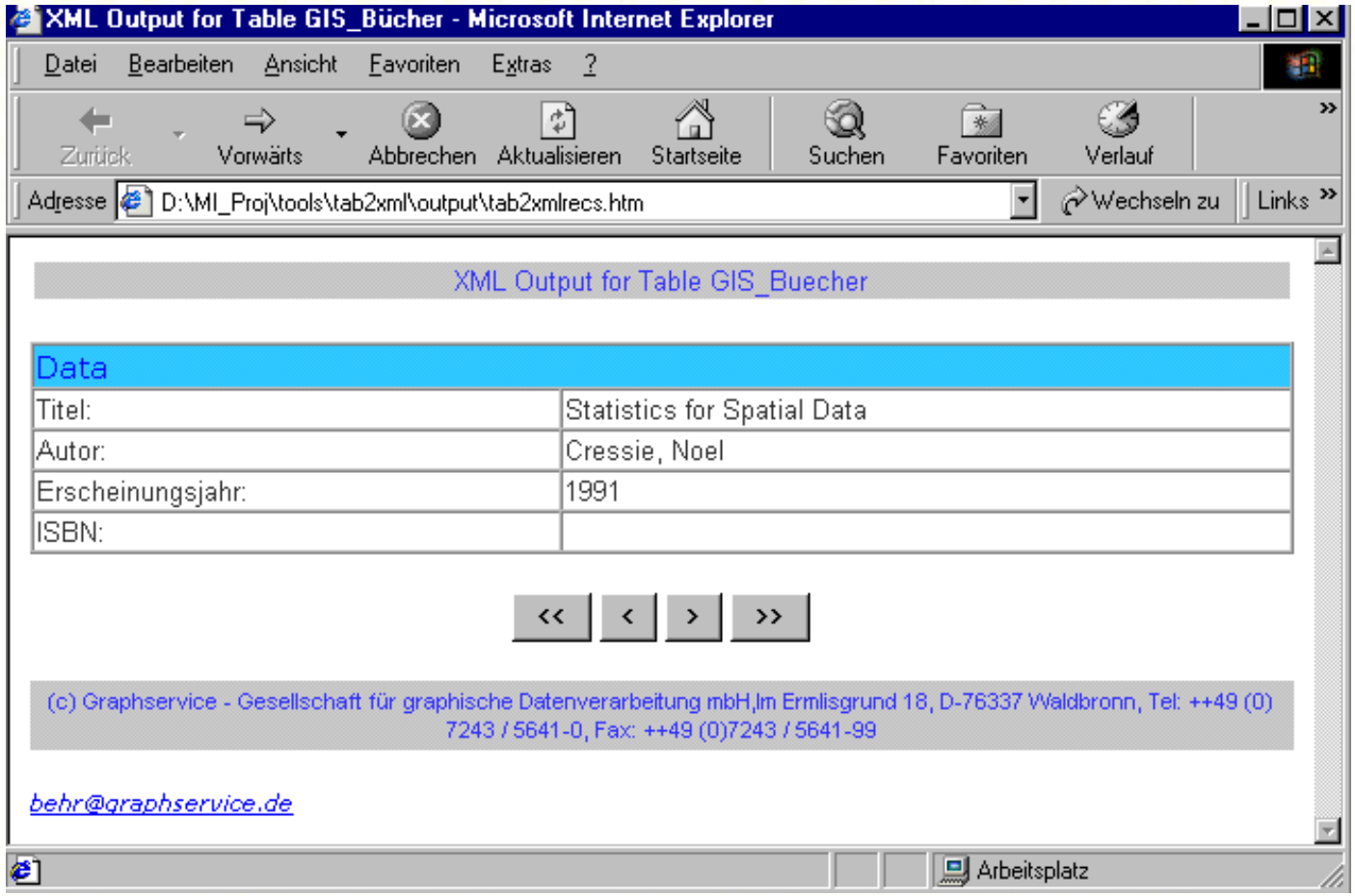
```
Title {display: block;
       font-family:Verdana, Arial;
       font-size:10pt;
       color:blue;
       font-weight:bold;
       text-align:left;
       margin-top:1cm;
       margin-left:2cm;
       text-decoration:underline;}

Author {display: block;
       margin-top:0.1cm;
       margin-left:2cm;
       margin-right:2cm;
       margin-bottom:0cm;
       font-family:Verdana, Arial;
       font-size:10pt;
       padding:0pt; }
...
```

Cascading Style Sheets, level 2
CSS2 Specification: http://www.w3.org/TR/REC-CSS2/

XML+ CSS in IE


D:\MI_Proj\tools\tab2xml\output\tab2xml.xml - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Adresse D:\MI_Proj\tools\tab2xml\output\tab2xml.xml
Wechseln zu
Links
Grundlagen der Geo-Informationssysteme, Band 1: Hardware, Software und Daten
Bill, Ralf
Wichmann Verlag, Heidelberg 1999 3-87907-325-2
Business Mapping im Marketing
Leiberich, Peter
Wichmann-Verlag, Heidelberg 1997 3879073155
Strategisches GIS-Management. Grundlagen, Systemeinfuehrung und Betrieb
Behr, Franz-Josef
Wichmann-Verlag, Heidelberg 2000 3879073503
Fertig
Arbeitsplatz

Using of IE



One short example...

```
<?xml version="1.0" encoding="ISO-8859-1"
standalone="yes"?>
<!DOCTYPE example [
<!ELEMENT example (#PCDATA)>
]>
<example>
Hello World
</example>
```

prolog

DTD

content

Extend sample of XML document

```
<?xml version="1.0" encoding="ISO-8859-1"
standalone=„no"?>
<?xml-stylesheet href= "musicians.xsl" type="text/xsl"?>
<musicianlist>
  <musician>
     <name>Joey Baron</name>
     <instrument>drums</instrument>
     <nrofrecordings>1</nrofrecordings>
  </musician>
  <musician>
     <name>Bill Frisell</name>
     <instrument>guitar</instrument>
     < nrofrecordings>3</nrofrecordings>
  </musician
</musicianlist>
```

Prolog – the XML declaration

- The first line in the document - the XML declaration
- Defines the XML version and the character encoding used in the document.
- In this case the document conforms to the 1.0 specification of XML and uses the ISO-8859-1 (Latin-1/West European) character set.

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

Stand-alone=no, if...

- The standalone document declaration must have the value "no" if any external markup declarations contain declarations of:
  - attributes with default values, if elements to which these attributes apply appear in the document without specifications of values for these attributes, or
  - entities (other than amp, lt, gt, apos, quot), if references to those entities appear in the document, or
  - attributes with values subject to normalization, where the attribute appears in the document with a value which will change as a result of normalization, or
  - element types with element content, if white space occurs directly within any instance of those types.
- I. e.: If an external DTD [subset] exists, always set standalone="no"!

Processing Instructions

- Processing instructions (PIs, not mandatory) allow documents to contain instructions for applications.

```
<?xml-stylesheet href= "musicians.xsl" type="text/xsl"?>
```

- In the above mentioned example you see the invocation of a XSL style sheet.

The root element

- The next line describes the root element of the document (like it was saying: "this document gives information about musicians"):

```
<musicianlist>
```

A child element

- After the root element, the next line describes a child elements of the root (**musicianlist**):

```
<musicianlist>
<musician>
```

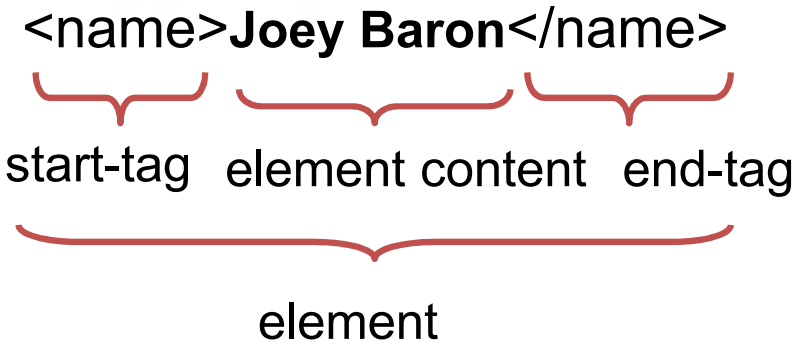
- The next 3 lines describe 3 child elements of a musician (name, instrument, and nrofrecordings):

```
<name>Joey Baron</name>
<instrument>drums</instrument>
<nrofrecordings>1</ nrofrecordings>
```

- Each element consists of a pair of tags.

Tags

- All of the information from the start of a start-tag to the end of an end-tag, and including everything in between, is called an element. So:



- The text between the start-tag and end-tag of an element is called the element content. The content between our tags will be data or elements.
- In this case, the element content is referred to as Parsed Character data, which is almost always referred to using its acronym, PCDATA.

Rules for element names

- Names can start with letters (including non-Latin characters) or the "_" character, but not numbers or other punctuation characters.
- After the first character, numbers are allowed, as are the characters "-" and ".".
- Names can't contain spaces.
- ":" not allowed -> reserved for using namespaces (we will see this at XSL!).
- Names can't start with the letters "xml", in uppercase, lowercase, or mixed.
- There can't be a space after the opening "<" character; the name of the element must come immediately after it. However, there can be space before the closing ">"character, if desired.

Naming conventions

- Give your elements distinct names, for your sanity, and for the sanity of those, who will read your XML documents.
- To help combat these kinds of problems, it's a good idea to pick a naming style and stick to it. Some examples of common styles are:

  <firstname> <first_name> <firstName>
  <first-name> <FirstName>

- Element names: as long as you like, but don't exaggerate. Names should be short and simple, like this: <book_title> not like this: <the_title_of_the_book>.
- Which style you choose isn't important; what is important is that you stick to it.
- A naming convention only helps when it's used consistently!

Naming conventions (cont.)

- XML documents often have a corresponding database, in which fields exist corresponding to elements in the XML document. A good practice is to use the naming rules of your database for the elements in the XML documents.
- Non-English letters like éòá are perfectly legal in XML element names, but watch out for problems if any other software you use doesn't support them!

Rules for tags

- In HTML documents you sometimes will find paragraphs like this:

```
<p>This is a paragraph
<p>This is another paragraph
```

- But XML elements must have a closing tag:

```
<p>This is a paragraph</p>
<p>This is another paragraph</p>
```

Rules for tags: Tags are case-sensitive

- Unlike HTML, XML tags are case sensitive.
- With XML, the tag <myTag> is different from the tag <MyTag>.
- Opening and closing tags must therefore be written with the same case:

```
<Message>This is incorrect</message>
<message>This is correct</message>
```

Rules for tags: Tag nesting

- All XML elements must be properly nested:

```
<b>This text is bold and <i>italic.</i></b>
```

- You have to avoid improper nesting:

~~This text is bold and italic~~

All XML documents must have *one* root element

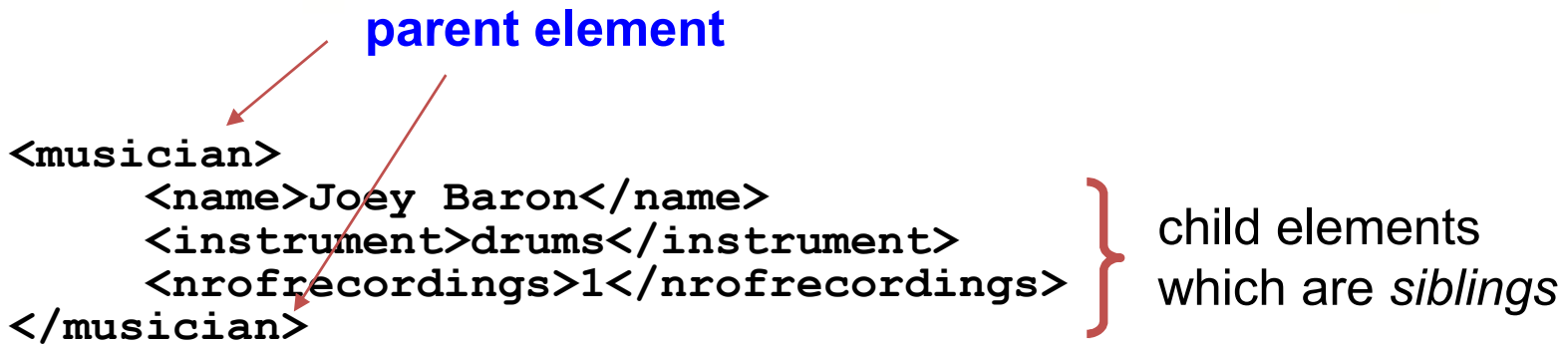
- All XML documents must contain a single tag pair to define a root element.
- The root element forms the first and the last tag of the document.

```
<?xml version="1.0" encoding="ISO-8859-
1"?>
<musicianlist>
  <musician>
     <name>Joey Baron</name>
     <instrument>drums</instrument>
     <nrofrecordings>1</nrofrecordings>
  </musician>
  <musician>
     <name>Bill Frisell</name>
     <instrument>guitar</instrument>
     < nrofrecordings>3</nrofrecordings>
  </musician>
</musicianlist>
```

All other elements are embedded within the root element

A parent can have children

- All elements can have sub elements (child elements). Sub elements must be correctly nested within their parent element:



Elements can have attributes

- XML elements can have attributes in name/value pairs just like in HTML or in query string of an URL.
- Attributes are used to provide additional information about elements.
- However, in XML the attribute value must always be quoted (single or double).
- Attributes are always part of the *starting tag*!

```
<text x="20" y="30" style="fill:#000; font-size: 24px"/>
```

Attribute *x* Attribute *y* Attribute *style*

Use of Elements vs. Attributes

- Data can be stored in child elements or in attributes.

```
<person>
<sex>female</sex>
<firstname>Anna</firstname>
<lastname>Smith</lastname>
</person>
```

```
<person sex="female">
<firstname>Anna</firstname>
<lastname>Smith</lastname>
</person>
```

Avoid using attributes?

- Here are some of the problems using attributes:
  - Attributes cannot contain multiple values (child elements can)
  - Attributes are not easily expandable (for future changes)
  - Attributes cannot describe structures (child elements can)
  - Attributes are more difficult to manipulate by program code
  - Attribute values are not easy to test against a DTD
- Try to use elements!
- Exception: The ID attribute (often used in SVG):

```
<note ID="501">...
```

White space is preserved

- White space includes things like the space character, new lines, and tabs. White space is used to separate words, as well as to make text more readable.
- With XML, the white space in element content is not truncated.
- This is unlike HTML. With HTML, a sentence like this:

```
<p>Hello,          my name is Ali Baba.</p>
```

- will be displayed like this:

Hello, my name is Ali Baba.

- …because HTML strips off the white space, but not XML!
- However: White spaces *outside* elements are not relevant!

Comments

- The syntax for writing comments in XML is similar to that of HTML:

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

XML Elements are Extensible

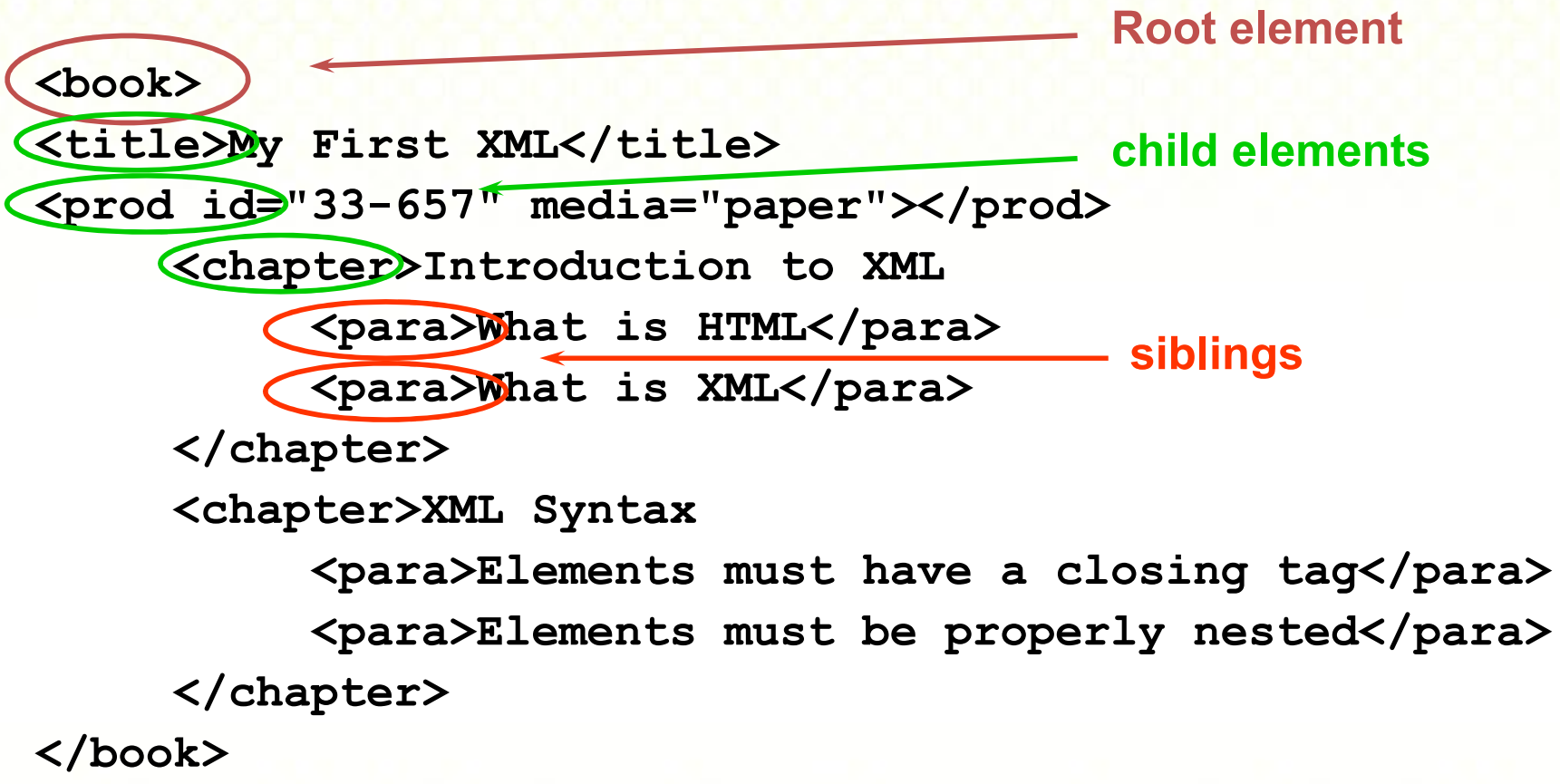
- XML documents can be extended to carry more information.

```
<musician>
  <name>Joey Baron</name>
  <instrument>drums</instrument>
  <nrofrecordings>1</nrofrecordings>
</musician>
```

```
<musician>
   <name>Joey Baron</name>
   <instrument>drums</instrument>
   <nrofrecordings>1</nrofrecordings>
   <age>37</age>
</musician>
```

An application will still be able to find the <name>, <instrument> and <nrofrecordings> elements in the XML document and process the file.

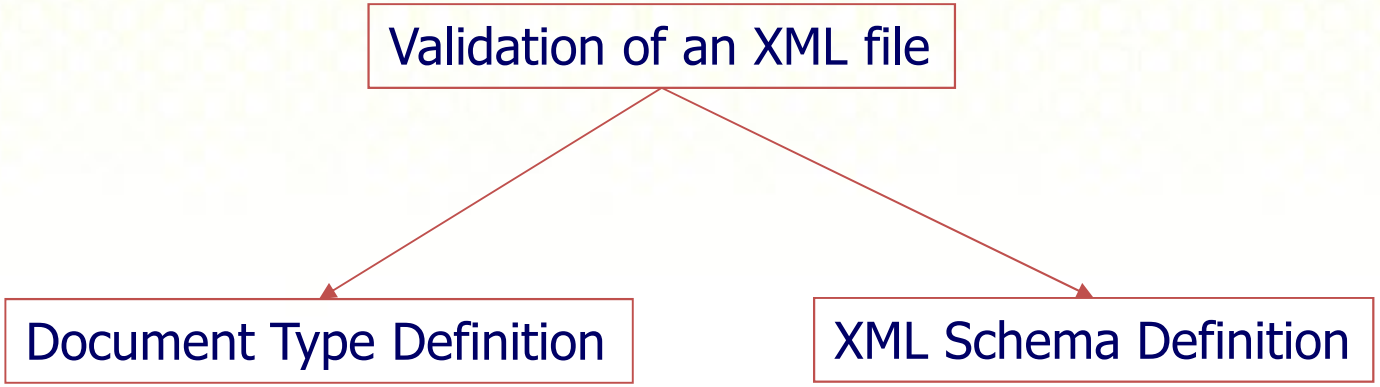
XML Elements have relations



Elements have Content

- element content,
- mixed content,
- simple content (text only), or
- empty content,
- an element can also have attributes.

Validation



- Easy to understand
- Validation limited
- detailed below

- More complex
- validation more powerful

Document Type Definition (DTD)

- The XML document type declaration contains or points to markup declarations that provide a grammar for a class of documents.
- It defines the document structure with a list of legal elements.
- A DTD can be declared inline in your XML document, or as an external reference.
- If the DTD is included in your XML source file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element [element-declarations]>
```

A first example

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE example [
<!ELEMENT example (#PCDATA)>
]>
<example>
Hello World
</example>
```

- The name in the document type declaration must match the element of the root element.
- There is only one element: example (the root element).
- example (#PCDATA)> defines the example element to be of the type "#PCDATA" (parsed character data).

Why use a DTD?

- With DTD, each XML files can carry a description of its own format with it.
- With a DTD, independent groups of people can agree to use a common DTD for interchanging data (i.e. spatial data).
- Your application can use a standard DTD to verify that the data you receive from other people or data vendors are valid.
- You can also use a DTD to verify your own data.

Our musicians with their DTD

```
<?xml version="1.0"?>
<!DOCTYPE musicianlist[
<!ELEMENT musicianlist(musician)+ >
<!ELEMENT musician (name, instrument, nrofrecordings)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT instrument (#PCDATA)>
<!ELEMENT nrofrecordings (#PCDATA)>
]>
< musicianlist >
  <musician>
     <name>Joey Baron</name>
     <instrument>drums</instrument>
     < nrofrecordings>1</ nrofrecordings>
  </musician>
  <musician>
     <name>Bill Frisell</name>
     <instrument>guitar</instrument>
     < nrofrecordings>3</ nrofrecordings>
  </musician>
</ musicianlist >
```

Our musicians with their DTD (cont.)

```
<!DOCTYPE musicianlist[
<!ELEMENT musicianlist(musician)+ >
<!ELEMENT musician (name, instrument, nrofrecordings)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT instrument (#PCDATA)>
<!ELEMENT nrofrecordings (#PCDATA)>
]>
```

- The name of the DOCTYPE is musicianlist.
- The root element consists of one or more elements musician.
- Each musician consists of the 3 child elements name, instrument, nrofrecordings.
- **name (#PCDATA)>** defines the example element to be of the type "#PCDATA" (parsed character data).

External DTD

```
<?xml version="1.0"?>
  <!DOCTYPE greeting SYSTEM "hello.dtd">
  <greeting>Hello World!</greeting>
```

- The external DTD file (in this case hello.dtd):
  - is ASCII
  - has *no* DOCTYPE element (this is part of the XML document)!
- External DTD file has to be embeded in the XML document using relative or absolute path names.

Form of DTDs

- Two forms:
  - Your own "system" DTD:

```
<!DOCTYPE brief SYSTEM "letter.dtd">
```

  - public DTD (for widely used DTDs)

```
<!DOCTYPE book PUBLIC "-//FirmaABC//DTD book//EN
 "http://www.firma.com/dtds/book.dtd">
```

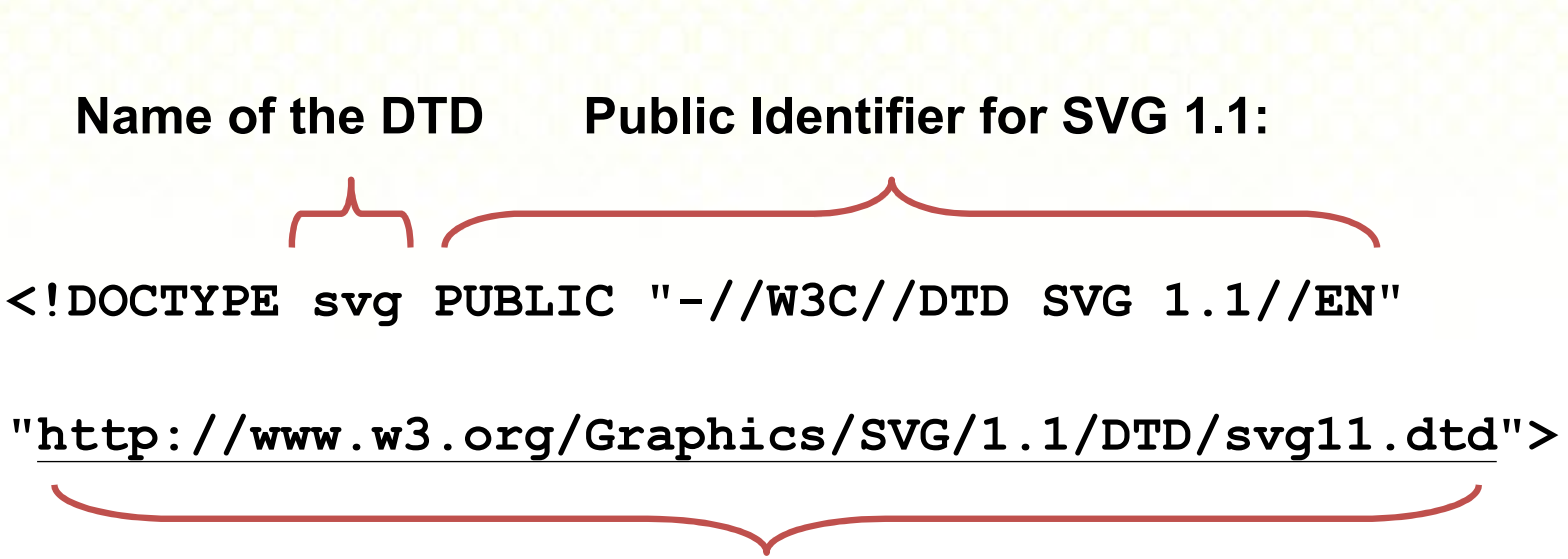
Parts of a “PUBLIC”-DTD

```
<!DOCTYPE book PUBLIC "-//FirmaABC//DTD book//EN
 “http://www.firma.com/dtds/book.dtd”>
```

- After keyword “PUBLIC”:
  - If DTD is a ISO-Standard, the declaration starts with "ISO",
  - If DTD is acknowledged by another standardization organisation, DTD starts with "+", otherwise with "-".
- After two "//" the institution is mentioned, to which the DTD belongs, after "//DTD" the name of the DTD, the language and the System Identifier are defined.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD DTD
XHTML 1.0 Strict//EN “DTD/xhtml1-strict.dtd”>
```

SVG DTD



Mixed DTD

- You can mix external and internal DTDs.
- If you use external and internal DTD, the internal DTD should be declared before the external DTD.
- **à** Entity and attribute declarations of the internal DTD have higher priority!

Declaring an Element

- In the DTD, XML elements are declared with an element declaration. An element declaration has the following syntax:

```
<!ELEMENT element-name category>
```

or

```
<!ELEMENT element-name (element-content)>
```

- There are several options for element content…

Element declarations

- An element must be declared only once.
- Example for element declarations:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE letter [
<!ELEMENT letter (subject, date, salutation, text, greeting)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT salutation(#PCDATA)>
<!ELEMENT text(#PCDATA)>
<!ELEMENT greeting(#PCDATA)>
<!ENTITY sender "Ali Baba">
]>
<letter>
...
```

Element content: Character data

- Keyword: #PCDATA
- PCDATA means parsed character data.
- … the text found between the start tag and the end tag
- PCDATA is text that will be parsed by a parser. Tags inside the text will be treated as markup and entities will be expanded.

Element content: Children

- Elements with one or more children are defined with the name of the children elements inside parentheses:
- When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document.

```
<!ELEMENT letter (subject, date, salutation, text, greeting)>
```

Element content: Elements and data

- This element is declared to have a child element and to have PCDATA.

```
<!ELEMENT letter (address,#PCDATA)>
<!ELEMENT address (#PCDATA)>
```

Empty elements

- Empty elements are declared with the category keyword EMPTY:

```
<!ELEMENT hr EMPTY>
```

- In your XML document, insert empty elements like in HTML:

<hr></hr>

or

<hr />

Entities

- Most of you will know the HTML entity reference: " ". This "non-breaking-space" entity is used in HTML to insert an extra space in a document. In XML:
- Entities are variables used to define common text.
- Entity references are references to entities.
- Entities are expanded when a document is parsed by an XML parser.
- The following 5 entities are predefined in XML:

&lt;	<
&gt;	>
&amp;	&
&quot;	"
&apos;	'

Entity example

- You are able to define your own entities.
- Definition of an entity:

```
<!ENTITY sender "Ali Baba">
```

- Reference of the entity in the XML document:

```
Yours sincerly &sender;
```

Declaring either/or content

- The following example declares that the "note" element must contain
  - a "to" element,
  - a "from" element,
  - a "header" element,
  - and either a "message" or a "body" element.

```
<!ELEMENT note (to,from,header,(message|body))>
```

Number of occurences

- *empty* Declaring only one occurrence of the same element
- ? Declaring zero or one occurrences of the same element.
- + Declaring minimum one occurrence of the same element.
- * Declaring zero or more occurrences of the same element.

Examples

- `<!ELEMENT adresslist (adress)+>`
- The root element contains at least one element *adress*.
- `<!ELEMENT adress (name,firstname?, street, zipcode, city, tel*,email*)>`
- The element *einzeladresse* contains the elements *name*, firstname (optional), *street*, zipcode, *city*, *tel* (zero or more occurrences) und email (zero or more occurrences).

Parameter entities

- Definition:

```
<!ENTITY % integer "#PCDATA">
```

- Reference of the entity in the XML document:

```
<!ENTITY year (%integer;)>
```

à integer is nothing else as PCDATA!

Declaring Attributes

- Usage: precise spezification of elements.
- Declaration: <!ATTLIST:

```
<!ATTLIST element-name attribute-name attribute-type
default-value >
```

Attribute-type

CDATA	The value is character data
(en1|en2|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

Default-Values

- The default-value can have the following values:

value	**The default value of the attribute**
#REQUIRED	**The attribute value must be included in the element**
#IMPLIED	**The attribute does not have to be included**
#FIXED value	**The attribute value is fixed**

```
DTD:
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">

Valid XML:
<square width="100" />
```

Specifying a Default attribute value

- The following (simplified) example shows the declaration of an SVG rect element:

```
<!ATTLIST rect;
    x %Coordinate.datatype; 0
    y %Coordinate.datatype; 0
    width %Length.datatype; #REQUIRED
    height %Length.datatype; #REQUIRED
    rx %Length.datatype; #IMPLIED
    ry %Length.datatype; #IMPLIED
>
```

Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

Source: http://www.w3schools.com/dtd/dtd_attributes.asp

Attribut type: character data

```
<!ATTLIST book title CDATA #REQUIRED
author CDATA #REQUIRED >
```

```
<book title="Strategies in GIS Management"
author="Franz-Josef Behr">
</book>
```

Attribut type: enumeration

- Use enumerated attribute values when you want the attribute values to be one of a fixed set of legal values.

```
<!ATTLIST tel num (private|office) #IMPLIED>
```

```
<tel num="office">121-5693</tel>
```

CDATA

- CDATA section can be used everywhere where character data are allowed.
- They are used to escape blocks of text containing characters which would otherwise be recognized as markup (i.e. <, >, &).
- Everything inside a CDATA section is ignored by the parser.
- A CDATA section starts with <![CDATA[and ends with]]>
- -> CDATA section often used in SVG documents!

```
<![CDATA[<greeting>Hello World!</greeting>]]
```

Drawbacks of DTD

- DTDs are not XML
- Attributes can not be validated.

Two important terms

- To sum up, two important terms (properties of XML objects):
  - Well formed
  - Valid

Well formed

- A textual object is a well-formed XML document if it satisfies at least the following conditions:
- Every start-tag must have a matching end-tag
- Tags do not overlap
- XML documents can have only one root element
- Element names must obey XML naming conventions
- XML is case-sensitive
- XML will keep white space in your text

A well-formed XML document

- ... A really short one:

```
<?xml version="1.0"?> <greeting>Hello, world!</greeting>
```

… and another one

- This is well formed:

```
<chapter>
<title> Test in XML </title>
<paragraph>XML is a transportation layer for
metadata.<bullets>Any geospatial data need to have
metadata theme.</bullets>
</paragraph>
</chapter>
```

Not well formed...

```
<chapter>
<title>Test in XML</title>
<paragraph>XML is a transportation layer for
metadata.
<bullets>Any geospatial data need to have
metadata theme.</paragraph></bullets>
</chapter >
```

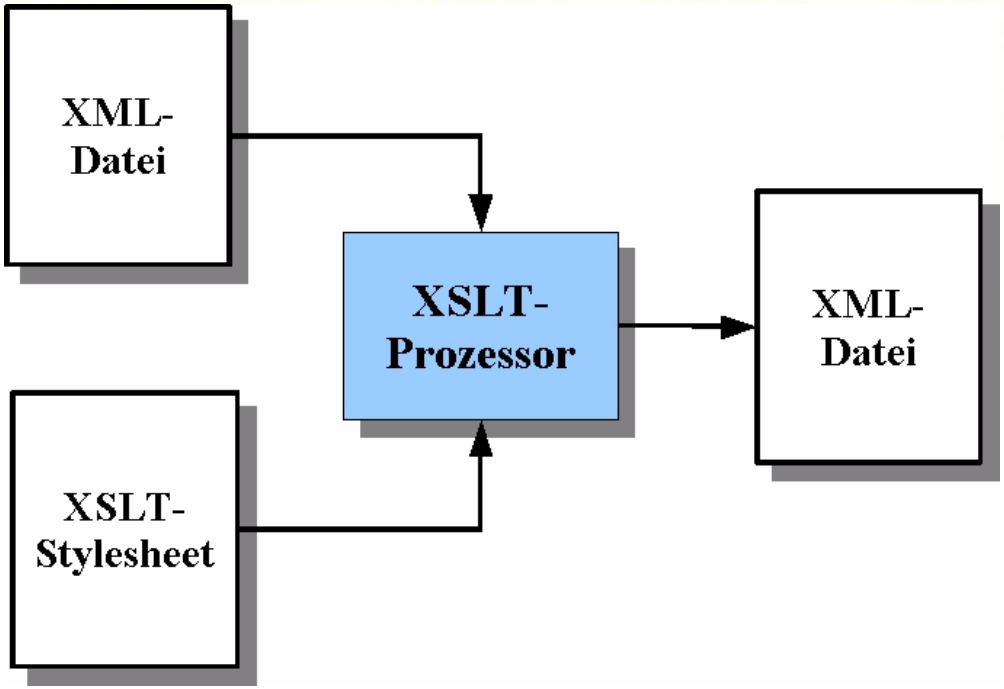
Valid

- Definition:

  If a well-formed XML document contains a Document Type Definition (DTD), and if the content is conformant with the DTD, the document is valid.

XSLT – eXtensible Style Language Transformation

Extensible Style Language (XSL)



XSL

- XSL is a language for expressing stylesheets. It consists of three parts:
  - XSL Transformations (XSLT): a language for transforming XML documents.
  - XML Path Language (XPath), an expression language used by XSLT to access or refer to parts of an XML document.
  - XSL Formatting Objects: an XML vocabulary for specifying formatting semantics.

XSL

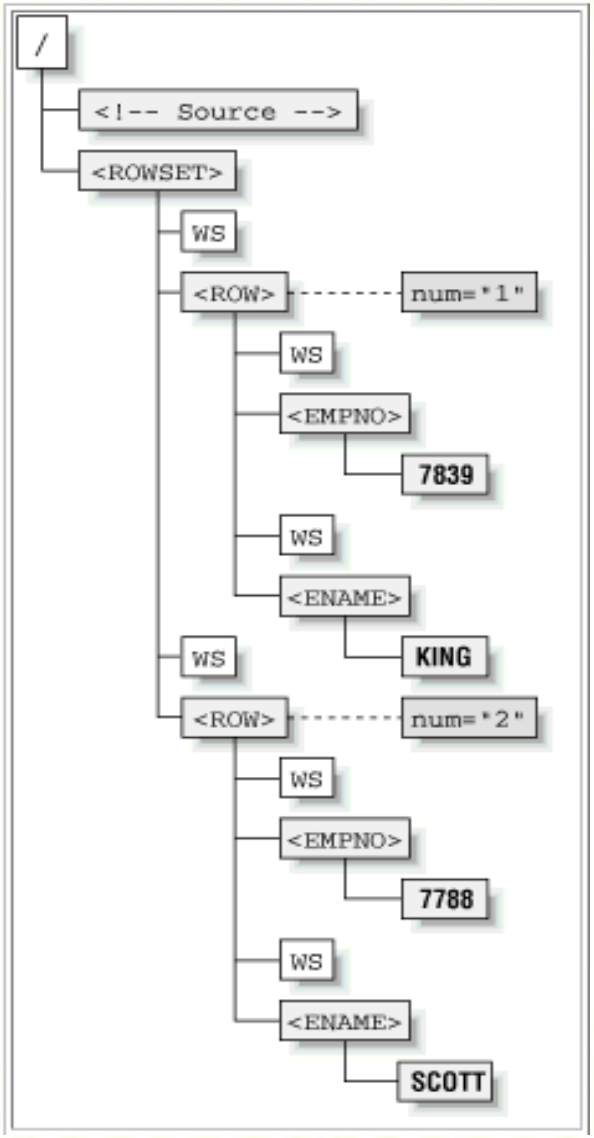
- XSLT: a language to transform XML.
- Xpath: a language to define XML parts or patterns.
- XSL Formatting Objects: a language to define XML display.

XPATH

- XPath is a separate recommendation from the W3C that uses a simple path language to address parts of an XML document.
- Generally speaking, XSLT provides a series of operations and manipulators, while XPath provides precision of selection and addressing.
- XPath is also used by the XML Linking specification).

Tree Structure of a XML Document

```
<!-- Emp.xml -->
     <ROWSET>
       <ROW num="1">
         <EMPNO>7839</EMPNO>
         <ENAME>KING</ENAME>
       </ROW>
       <ROW num="2">
         <EMPNO>7788</EMPNO>
         <ENAME>SCOTT</ENAME>
       </ROW>
     </ROWSET>
```



Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
     <?xml-stylesheet href="selfxml.xsl" type="text/xsl" ?>
     <Mitglieder>
     <Verein> Demo-Verein </Verein>
          <Mitglied Id="1">
               <Vorname> Thomas </Vorname>
               <Nachname> Maier </Nachname>
               <Strasse> Hauptstrasse 12 </Strasse>
               <PLZ> 76228 </PLZ> <Stadt> Karlsruhe </Stadt>
            </Mitglied>
          <Mitglied Id="2">
               <Vorname> Michael </Vorname>
               <Nachname> Schneider </Nachname>
               <Strasse> Bahnhofsstrasse 21 </Strasse>
               <PLZ> 54321 </PLZ> <Stadt> Stuttgart </Stadt>
            </Mitglied>
     </Mitglieder>
```

Example: Root-Element of the XSL-file

- The character "/" is equivalent with the root element.

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl">

          <xsl:template match="/">
             <html>
               <head>
                 <Title>Test Page</Title>
               </head>
               <body>
                 <H1>This is a test!</H1>
               </body>
             </html>
          </xsl:template>

     </xsl:stylesheet>
```

Example: Root-Element of the XSL-file

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html> …
```

- Since the style sheet is an XML document itself, the document begins with an xml declaration: <?xml version='1.0'?>
- The xsl:stylesheet tag in the second line defines the start of the stylesheet.
- The xsl:template tag in the third line defines the start of a template. The template attribute match="/" associates (matches) the template to the root (/) of the XML source document.
- The rest of the document contains the template itself, except for the last two lines that defines the end of the template and the end of the style sheet.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
          <xsl:template match="/">
             <HTML>
               <Head>
                 <Title> <xsl:value-of select="Staff"/> </Title>
               </Head>
               <Body>
                 <H1>Test XML</H1>
                   <table border="1">
                     <xsl:for-each select="Staff">
                       <tr>
                         <td> <xsl:value-of select="FirstName"/> </td>
                         <td> <xsl:value-of select="LastName"/> </td>
                       </tr>
                     </xsl:for-each>
                   </table> </Body> </HTML>
          </xsl:template>
     </xsl:stylesheet>
```

loop

Output is generated

... with sorting

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html><Head>
<title> <xsl:value-of select=" Staff "/> </Title>
</Head>
<body>
<H1> Ein Sortiertest </H1>
<table border="1">
<xsl:for-each select=" Staff " order-by=" FirstName
LastName " >
<tr>
<td> <xsl:value-of select=" FirstName "/> </td>
<td> <xsl:value-of select=" LastName "/> </td>
</tr>
</xsl:for-each>
</table>
</body></html>
</xsl:template>
</xsl:stylesheet>
```

Example with formating the output

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="beispiel.xsl" type="text/xsl" ?>
<Buecher>
   <Buch>
      <Name> Mein erstes Buch </Name>
      <Autor> Jochen Mueller </Autor>
      <Hinweis> Das Buch: <fett>sehr</fett> empfehlenswert!
      </Hinweis>
   </Buch>
   <Buch>
      <Name> XML-Buch </Name>
      <Autor> Andreas Schmidt </Autor>
      <Hinweis> Mehr Informationen
          <link href="http://www.selfxml.de"> hier </link> .
      </Hinweis>
   </Buch>
</Buecher>
```

Example with formatting the output: XSL-Document

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl" >
     <xsl:template match="/">
        <html><Head> <Title> Eine Buecherseite </Title></head>
          <body> <h3> eine Uebersicht </h3>
            <xsl:apply-templates/>
          </Body></HTML>
     </xsl:template>
     <xsl:template match="Buecher">
        <table border="2">
        <xsl:apply-templates/>
        </table>
     </xsl:template>

     <xsl:template match="Buch">
        <tr>
          <xsl:apply-templates/>
        </tr>
     </xsl:template>
```

Example with formatting the output: XSL-Document II

```
<xsl:template match="Name">
   <td>
     <xsl:apply-templates/>
   </td>
</xsl:template>

<xsl:template match="Autor">
   <td>
     <xsl:apply-templates/>
   </td>
</xsl:template>

<xsl:template match="Hinweis">
   <td>
     <xsl:apply-templates/>
   </td>
</xsl:template>
```

Example with formatting the output: XSL-Document III

```
<xsl:template match="fett">
        <b>
          <xsl:apply-templates/>
        </b>
    </xsl:template>

    <xsl:template match="link">
        <a>
          <xsl:attribute name="href">
            <xsl:value-of select="@href"/>
          </xsl:attribute>
          <xsl:apply-templates/>
        </a>
    </xsl:template>

    <xsl:template match="text()">
        <xsl:value-of/>
    </xsl:template>

    </xsl:stylesheet>
```

Example with formatting the output: Result

eine Uebersicht

Mein erstes Buch	Jochen Mueller	Das Buch ist **sehr** empfehlenswert!
XML-Buch	Andreas Schmidt	Mehr Informationen hier .

XSL uses Templates

- XSL uses one or more templates to define how to output XML elements. A match attribute is used to associate the template with an XML element.
- The match attribute can also be used to define a template for a whole branch of the XML document.

XSL uses Templates

- Since the style sheet is an XML document itself, the document begins with an xml declaration: <?xml version='1.0'?>
- The xsl:stylesheet tag in the second line defines the start of the stylesheet.
- The xsl:template tag in the third line defines the start of a template. The template attribute match="/" associates (matches) the template to the root (/) of the XML source document.
- The rest of the document contains the template itself, except for the last two lines that defines the end of the template and the end of the style sheet.

The <xsl:value-of> Element

- The XSL <xsl:value-of> element can be used to select XML elements into the output stream of the XSL transformation:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <xsl:template match="/">
      <html><body>
      <table border="1">
         <tr><th>Title</th><th>Artist</th></tr>
         <tr>
            <td><xsl:value-of select="CATALOG/CD/TITLE"/></td>
            <td><xsl:value-of select="CATALOG/CD/ARTIST"/></td>
         </tr>
      </table>
      </body></html>
   </xsl:template>
</xsl:stylesheet>
```

The <xsl:for-each> Element

- The XSL <xsl:for-each> element can be used to select every XML element into the output stream of the XSL transformation:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html><body>
<table border="1">
<tr><th>Title</th><th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/CD">
<tr>
<td><xsl:value-of select="TITLE"/></td>
<td><xsl:value-of select="ARTIST"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

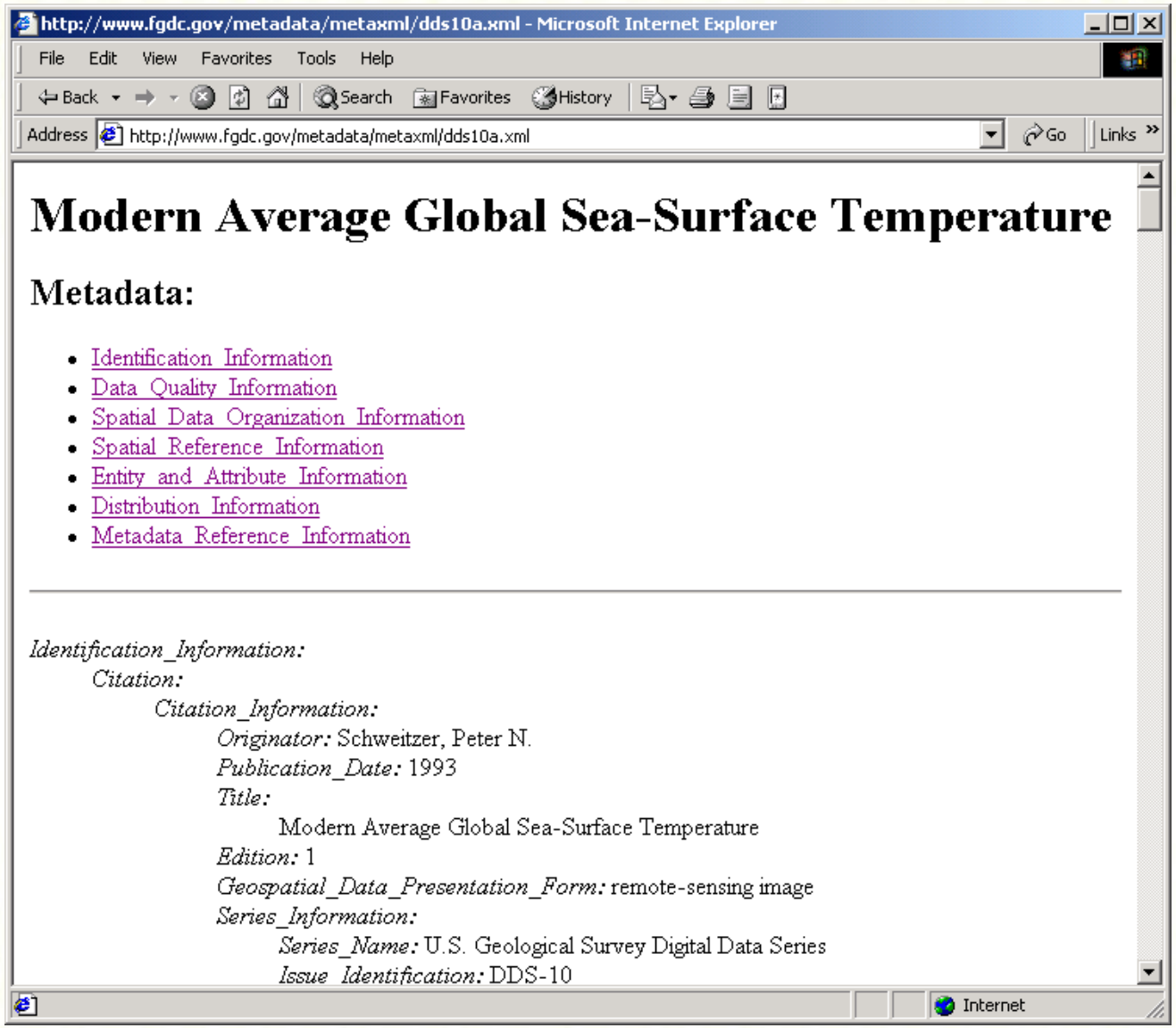
Example: Metadata, XML, XSL

dds10a[1] - Notepad

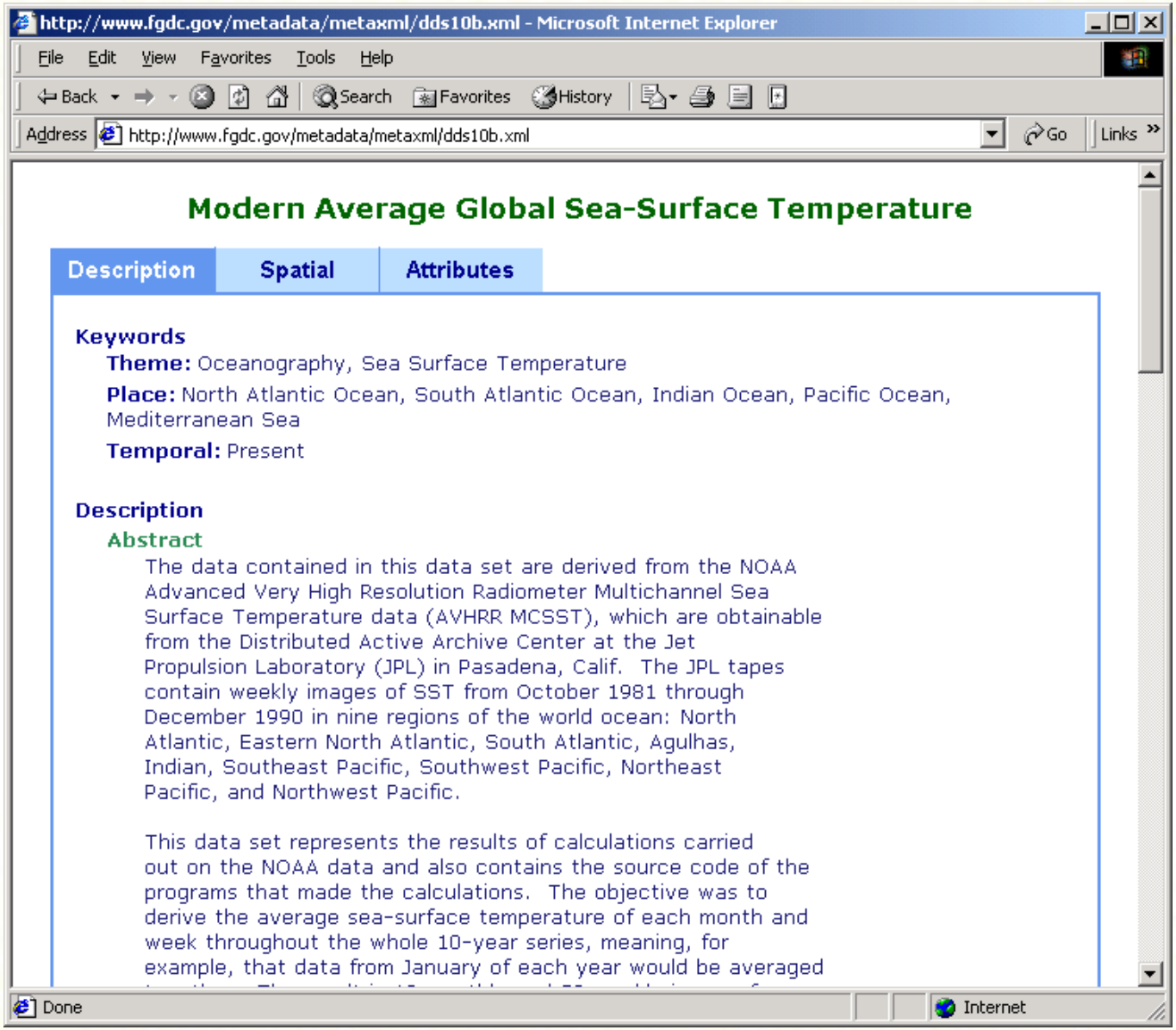
File Edit Format Help

```
<metadata>
<idinfo>
<citation>
<citeinfo>
<origin>Schweitzer, Peter N.</origin>
<pubdate>1993</pubdate>
<title>Modern Average Global Sea-Surface Temperature</title>
<edition>1</edition>
<geoform>remote-sensing image</geoform>
<serinfo>
<sername>U.S. Geological Survey Digital Data Series</sername>
<issue>DDS-10</issue>
</serinfo>
<pubinfo>
<pubplace>Reston, Virginia</pubplace>
<publish>U.S. Geological Survey</publish>
</pubinfo>
<onlink>http://geochange.er.usgs.gov/pub/magsst/magsst.html</onlink>
</citeinfo>
</citation>
<descript>
<abstract>
The data contained in this data set are derived from the NOAA
Advanced Very High Resolution Radiometer Multichannel Sea
Surface Temperature data (AVHRR MCSST), which are obtainable
from the Distributed Active Archive Center at the Jet
Propulsion Laboratory (JPL) in Pasadena, Calif.  The JPL tapes
contain weekly images of SST from October 1981 through
December 1990 in nine regions of the world ocean: North
Atlantic, Eastern North Atlantic, South Atlantic, Agulhas,
Indian, Southeast Pacific, Southwest Pacific, Northeast
Pacific, and Northwest Pacific.

This data set represents the results of calculations carried
out on the NOAA data and also contains the source code of the
programs that made the calculations.  The objective was to
```

Example: Metadata, XML, XSL



Example: Metadata, XML, XSL



Example: Metadata, XML, XSL