The eXtensible Markup Language: An Introduction to XML Documents & Databases

Preliminary Issues
What is a document?

- Content: the components (words, images etc.) which make up a document
- Structure: the organization and inter-relationship of the components
- Presentation: how a document looks and what processes are applied to it

Separating these things means...

- The content can be re-used
  - for printing
  - for querying
  - for exchanging
- The structure can be formally validated
- The presentation can be customized for
  - different media
  - different audiences
- … in short, the information can be uncoupled from its processing
Documents vs Databases

- **Document world**
  - plenty of small documents
    - usually static
  - implicit structure
    - section, paragraph, toc,
  - tagging
    - human friendly
  - content
    - form/layout, annotation
  - paradigms
    - “Save as”, WYSIWYG
  - metadata
    - author name, date, subject

- **Database world**
  - a few large databases
    - usually dynamic
  - explicit structure
    - types
  - records
    - machine friendly
  - content
    - data, methods
  - paradigms
    - Data Independence, Transaction Management, Query Languages
  - metadata
    - schema description

---

DBMS ANSI/SPARC Architecture

- **EXTERNAL LEVEL**
  - VIEW 1
  - VIEW 2
  - VIEW 3

- **CONCEPTUAL LEVEL**
  - INTEGRATION

- **LOGICAL SCHEMA**

- **INTERNAL LEVEL**
What to do with them

Documents

- editing
- spell-checking
- counting words
- retrieving (IR)
- printing

Database

- updating
- cleaning
- querying
- composing/transforming

Query Languages

Document Retrieval

Claude Monet and San Diego Museum of Art

```sql
select p from Artists a, a.artwork p
where a.first = "Claude"
    and a.last = "Monet"
    and p.located =
    "San Diego Museum of Art"
```
What’s Wrong with HTML

- If written properly, normal HTML may reflect document presentation, but it cannot adequately represent the semantics & structure of data.

```html
<B>MONET, Claude</B><BR>
Haystacks at Chaillly at Sunrise<BR>
1865<BR>
Oil on canvas<BR>
30 x 60 cm (11 7/8 x 23 3/4 in.)<BR>
San Diego Museum of Art<BR>
</P>
</IMG SRC="http://192.41.13.240/archive/m/monet/hayricks.jpg">
A possible XML markup of the same information will retain the structure (and the semantics) of the various data objects.

```xml
<ARTIST>
  <NAME><FIRST>Claude</FIRST><LAST>Monet</LAST></NAME>
<ARTWORK>
  <ARTIFACT>
    <TITLE>Haystacks at Chailly at Sunrise</TITLE>
    <DATE>1865</DATE>
    <MATERIAL>Oil on canvas</MATERIAL>
    <DIM Metric='cm'>
      <HEIGHT>30</HEIGHT><WIDTH>60</WIDTH></DIM>
    <DIM Metric='in'>
      <HEIGHT>11 7/8</HEIGHT><WIDTH>23 3/4</WIDTH></DIM>
    <LOCATION>San Diego Museum of Art</LOCATION>
    <IMAGE File='http://192.41.13.240/archive/m/monet/hayricks.jpg'/>
  </ARTIFACT>
</ARTWORK>
</ARTIST>
```
XML can be Published as normal Web Data

What is XML?

- **Markup Meta-Language** for domain or application specific structured documentation
  - Mathematical, chemical, musical, publishing, etc.
- Developed by the SGML Editorial Board formed under the auspices of the [World Wide Web Consortium (W3C)](https://www.w3.org)
  - Founded in 1996 by Jon Bosac (Sun) and various Web/SGML vendors: Textuality, Netscape, Microsoft, INSO, HP, Highland, NCSA, ArbortText, GRIF, SoftQuand
- **Subset of SGML** optimized for use in the Inter/Intranet
  - SGML is proving difficult to implement for Web/Intranet applications
  - SGML has been hard to cost-justify to management
- Opens the way for a new generation of Web applications
  - Improve precision during searching and retrieval
  - Enable multiple usage of the same data
  - Facilitate distributed processing with more versatile ways to manipulate data
Why XML?

- XML provides key features for a new generation of Web applications:
  - Structuring: unlike HTML it preserves the structure of the data
  - Extensibility: not a fixed format like HTML but user-oriented tagging
  - Validation: provides the means to consuming applications to check data for structural validity on importation
  - Presentation Late Binding: describes data, not visual presentation
  - Human Readable: similar to HTML
  - Interchange: good for transmission of data from server to browser, and from application to application, or machine to machine
  - Open standard: non proprietary format

- XML becomes an integral part of the Web infrastructure
  - Microsoft Explorer (V5.0) already offers XML browsing
  - Ongoing XML implementation by Netscape
  - Various XML middleware and manipulation tools

The XML Language Family

- XML (Extensible Markup Language)
  - A subset of SGML (ISO 8879) designed for easy implementation

- XLink (Extensible Linking Language)
  - A set of standard hypertext mechanisms based on HyTime (ISO/IEC 10744) and the Text Encoding Initiative (TEI)

- XSL (Extensible Stylesheet Language)
  - A standard stylesheet language for structured information derived from DSSSL (ISO/IEC 10179) and key CSS concepts
Interrelationships Among the Various W3C Efforts

XML Syntax and Semantics
An Example of XML Markup

```
<ARTIST>
  <NAME>
    <FIRST>Claude</FIRST> <LAST>Monet</LAST>
  </NAME>
<ARTWORK>
  <ARTIFACT>
    <TITLE>Haystacks at Chailly at Sunrise</TITLE>
    <DATE>1865</DATE>
    <MATERIAL>Oil on canvas</MATERIAL>
    <DIM Metric='cm'>
      <HEIGHT>30</HEIGHT>
      <WIDTH>60</WIDTH>
    </DIM>
    <DIM Metric='in'>
      <HEIGHT>11 7/8</HEIGHT>
      <WIDTH>23 3/4</WIDTH>
    </DIM>
    <LOCATION>San Diego Museum of Art</LOCATION>
    <IMAGE File='http://192.41.13.240/artchive/m/monet/hayricks.jpg'/>
  </ARTIFACT>
</ARTWORK>
</ARTIST>
```

The Logical Tree Structure of XML

```
ARTIST
  NAME
    FIRST Claude
    LAST MONET
  ARTWORK
    ARTIFACT
      TITLE Haystacks
      DATE 1865
      DIM H 30
      DIM W 60
      DIM H 11 7/8
      DIM W 23 3/4
      LOCATION San Diego Museum of Art
      MATERIAL Oil on canvas
      IMAGE ...hayricks.jpg
```

---

Christophides Vassilis
XML Document Type Definitions

```xml
<!DOCTYPE artist [ 
  <!ELEMENT artist (name, born, death, artwork, nationality?, influences)> 
  <!ATTLIST artist id ID #REQUIRED xml:lang NM_TOKEN #IMPLIED> 
  <!ELEMENT name (first, last)> 
  <!ELEMENT first (#PCDATA)> 
  <!ELEMENT last (#PCDATA)> ... 
  <!ELEMENT artwork (artifact+)> 
  <!ELEMENT artifact (title, date, material, dim*, location, image)> 
  <!ELEMENT title (#PCDATA)> ... 
  <!ELEMENT dim (height, width)> 
  <!ATTLIST dim metric (cm|in) 'cm'> 
  <!ELEMENT location (#PCDATA)> 
  <!ELEMENT image EMPTY> 
  <!ATTLIST image file ENTITY #REQUIRED> 
  <!ELEMENT influences (PCDATA | aref)*> 
  <!ELEMENT aref EMPTY> 
  <!ATTLIST aref xml:link CDATA #FIXED 'simple' href CDATA #REQUIRED> 
  <!NOTATION jpeg PUBLIC "-/local//NOTATION Gpeg Images//EN"> 
  <!ENTITY fig1 SYSTEM ".../monet/hayricks.jpg' NDATA jpeg"> ]>
```

XML Core Markup Features

- **Elements**: Components of the tree logical structure defined by a DTD
  - identified in a document instance by *descriptive markup*, usually a start-tag and end-tag
- **Attributes**: Characteristics associated to the elements (other than their content and type)
  - may be applied to one specific instance of a given element
- **Entities**: Named fragments of information that can be stored separately from a document (or a DTD)
  - can be included in the document (or the DTD) one or more times by reference to their names
### Definition of Element’s Content

<table>
<thead>
<tr>
<th>EMPTY</th>
<th>ANY</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;!ELEMENT image EMPTY&gt;</code></td>
<td><code>&lt;!ELEMENT object ANY&gt;</code></td>
</tr>
<tr>
<td>end-tag is omitted implicit content or generated automatically by an application</td>
<td>element content can consist of a free mixture of parsed character data and any of the elements in the DTD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MIXED TEXT</th>
<th>GROUP</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;!ELEMENT title (#PCDATA)&gt;</code></td>
<td>`&lt;!ELEMENT media (image</td>
</tr>
<tr>
<td>element content can contain characters, entity references and tags allowed by the element model</td>
<td><code>&lt;!ELEMENT name (first,last)&gt;</code></td>
</tr>
</tbody>
</table>
| `<!ELEMENT name (first,last)>` | element content with connectors (’,’, ’|’) and occurrence indicators (’,’, ’?’, ’*’)

- Mixed models must be optional repeatable OR-groups, with #PCDATA first

### What XML can express?

- **Sequence** « , »
  
  `<!ELEMENT name (first,last)>`

- **Choice** « | »
  
  `<!ELEMENT media (image | video)>`

- **Option** ( 1 or 0 ) « ? »
  
  `<!ELEMENT artist (... , nationality?, ...)`

- **Repetition** (1 or more ) « + »
  
  `<!ELEMENT artwork (artifact+)>`

- **Option and Repetition** ( 0, 1 or more ) « * »
  
  `<!ELEMENT artwork (... , dim*, ...)>`
XML Content Models and Regular Expressions

- Each element content model is defined by a regular expression
- Example: name, addr*, email
- Each regular expression determines a corresponding finite state automaton
- This suggests a simple parsing program

Content Models should be defined by unambiguous regular expressions

XML Regular Expressions: Another Example

- Adding in the optional greet further complicates things
- Example: name,address*,(tel | fax)*,email*
Definition of Attribut’s Content

| (value1 | value2 | .. | valuen) | ![ATTLIST dim metric (cm | in)>  
|<dim metric = ‘in’> |
| CDATA | string of valid XML character data | ![ATTLIST ref href CDATA >  
|<ref href= ‘http://192.41.13.240/  
|artchive/m/monet.xml’> |
| ID | symbolic identifier of an element | ![ATTLIST artist oid ID>  
|<artist oid = Monet> |
| IDREF(S) | reference (or list of) to element’s identifiers | ![ATTLIST artifact creator IDREF>  
|<artifact creator = Monet> |
| ENTITYIES | reference (or list of) to entity names | ![ATTLIST image file ENTITY>  
|<image file = fig1> |
| NMTOKEN(S) | valid XML string preceded by numbers, hyphen, period | ![ATTLIST artist xml:lang NMTOKEN>  
|<artist xml:lang = “_english”> |

- More types (e.g., DATE) may soon be part of the standard

Attribute Default Values

- Value ‘vi’
  - a given value from an enumeration of values
- #FIXED value
  - the value is the only possible instance for the attribute
- #REQUIRED
  - the value must be supplied
- #IMPLIED
  - the value can be optionally supplied
XML Entities

- Entities allow the definition of short strings to stand for more complex information, which can reside inside or outside the document or its DTD.
- Used for substitutions of data or markup:
  - DTD level e.g., markup declaration (Parameter entity)
  - Document level e.g., data and markup instances (General entity)
- Used for references to external data or markup sources:
  - the content of the entity can be found using an XML system-specific storage location (Specific entity)
  - the content of the entity can be found by mapping a public identifier to a system-specific storage location (Public entity)

XML Parameter Entities

- Parameter entities are used for extensible declarations (e.g., macros) of complex content models or attributes in a DTD
  
  ```xml
  <!ENTITY % style "impressionism | cubism | surrealism"/>
  ```
- Parameter entities can be nested
  
  ```xml
  <!ENTITY % bibelem2 "%bibelem; | expressionism | dada">
  ```
  
  *but we must avoid infinite loops*
  
  ```xml
  <!ENTITY % bibelem "%bibelem; | expressionism | dada"/>
  ```
- Replacement entity text can be found outside the DTD
  
  ```xml
  <!ENTITY % ISOlat2 PUBLIC "ISO 8879-1986//ENTITIES Added Latin 2//EN"/>
  ```
XML General Entities

- General entities are used for substitution of textual or not textual objects (e.g., constants) occurring many times or are volatiles in the document instances
  
  ```xml
  <!ENTITY xml "Extensible Markup Language”>
  ```

- Replacement text of general entities can contain tags, character references or other entities
  
  ```xml
  <!ENTITY www “W3C Recommendation 10-February-1998”>
  <!ENTITY xml "<TITLE>Extensible Markup Language &www; <TITLE>”>
  ```

- but also we must avoid infinite loops

- The content of a general entity can be found outside the DTD and it may have a particular format

XML Specific Entities

- Specific entities can be viewed as “abstract storage objects” (e.g., data stream) that are mapped onto real ones by using a system-specific storage location

- Sub-documents encoded in XML with a different DTD
  
  ```xml
  <!ENTITY biography SYSTEM “... /monet.xml”>
  ```

- Textual data encoded with a particular format
  
  ```xml
  <!ENTITY bibliography SYSTEM “... /monet.bib”>
  ```

- Non-SGML data
  
  ```xml
  <!ENTITY fig1 SYSTEM “... /monet/hayricks.jpg” NDATA jpeg>
  ```
Well-Formed XML

- A textual object is said to be a well-formed XML document if it meets all the well-formedness constraints (WFCs) of the XML syntax:
  - tags (etc.) are syntactically correct
  - every tag has an end-tag
  - tags are properly nested
  - there exists a root

- By definition if a document is not well-formed, it is not XML
  - This means that there is no an XML document which is not well-formed, and XML processors are not required to do anything with such documents
Valid XML

- A well-formed document is valid only if it contains a proper DTD and if the document obeys the constraints of that DTD and therefore the XML Validity Constraints (VCs)
  - only declared tags are used
  - all tag occurrences conform to specified content models

- Examples:
  - The following XML Document is well-formed but not valid
    ```xml
    <artist> MONET, Claude </artist>
    ```
  - The following XML Document is not even well-formed
    ```xml
    <first> Claude </first> <last> MONET </last>
    ```

When do we need a DTD?

- At document preparation time (definitely)
  - validation, checking, consistency
- At document processing time (probably)
  - simplifies generic/specific processing
  - may clarify intended semantics
- At document delivery time (possibly)
  - strictly unnecessary for well-formed docs
  - but reduces processing effort
Where is the behaviour of XML defined?

- In a stylesheet
  - using XSL or CSS
- Possibly embedded in a program
  - applet, or script, or JAVA bean
  - defined for that particular DTD, set of tags, or tag
- By reference to pre-existing mutual agreement amongst user communities
  - aka “namespaces”
- By reference to a Document Object Model

Comparing XML and Programming Languages

- validation
- entity reference
- entity parameter
- ANY
- IDREF
- DTD
- conditional section
- key entities
- namespace

- type-checking
- constants
- macros
- void*
- void*
- header file
- #ifdef
- standard library
- namespace

- But no type inference, polymorphism, modules, etc.
XML DTDs vs. Database Schemas

- By database standards, DTDs are rather weak specifications
  - Only one base type i.e., PCDATA
  - Only two element constructors i.e., sequence and alternative
  - No useful “abstractions” e.g., bulk types, inheritance
  - IDREFs are untyped
    - You point to something, but you don’t know what!
  - No integrity constraints e.g., child is inverse of parent
  - No methods
  - Tag definitions are global

- Recent XML extensions impose something like a schema or type on an XML data (XML Schema)

XML vs. ODMG ODL: Example

class Movie
(  extent Movies, key title )
{
  attribute string title;
  attribute string director;
  relationship set<Actor> casts
    inverse Actor::acted_In;
  attribute int budget;
} ;

class Actor
(  extent Actors, key name )
{
  attribute string name;
  relationship set<Movie> acted_In
    inverse Movie::casts;
  attribute int age;
  attribute set<string> directed;
} ;
XML vs. ODMG ODL: Example

```xml
<db>
  <movie id="m1">
    <title>Waking Ned Divine</title>
    <director>Kirk Jones III</director>
    <cast idrefs="a1 a3"/>
    <budget>100,000</budget>
  </movie>
  <actor id="a1">
    <name>David Kelly</name>
    <acted_In idrefs="m1 m3 m78"/>
  </actor>
  <movie id="m2">
    <title>Dragonheart</title>
    <director>Rob Cohen</director>
    <cast idrefs="a2 a9 a21"/>
    <budget>110,000</budget>
  </movie>
  <actor id="a2">
    <name>Sean Connery</name>
    <acted_In idrefs="m2 m9 m11"/>
    <age>68</age>
  </actor>
  <movie id="m3">
    <title>Moondance</title>
    <director>Dagmar Hintz</director>
    <cast idrefs="a1 a8"/>
    <budget>90,000</budget>
  </movie>
  <actor id="a3">
    <name>Ian Bannen</name>
    <acted_In idrefs="m1 m35"/>
  </actor>
</db>
```

```xml
<!DOCTYPE db [  
<!ELEMENT db (movie+, actor+)>
<!ELEMENT movie (title, director, cast, budget)>  
<!ATTLIST movie id ID #REQUIRED>  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT director (#PCDATA)>  
<!ELEMENT cast EMPTY>  
<!ATTLIST cast idrefs IDREFS #REQUIRED>  
<!ELEMENT budget (#PCDATA)>  
<!ELEMENT actor (name, acted_In, age?, directed*)>  
<!ATTLIST actor id ID #REQUIRED>  
<!ELEMENT name (#PCDATA)>  
<!ELEMENT acted_In EMPTY>  
<!ATTLIST acted_In idrefs IDREFS #REQUIRED>  
<!ELEMENT age (#PCDATA)>  
<!ELEMENT directed (#PCDATA)> ]>
```
Mapping Between XML and Objects

XML vs Relational DBMS

```
<!DOCTYPE db [  
  <!ELEMENT db [(projects, employees)>  
  <!ELEMENT projects (project*)>  
  <!ELEMENT employees (employee*)>  
  <!ELEMENT project (title, budget, managedBy)>  
  <!ELEMENT employee (name, ssn, age)>  
  ...  
]>

<!DOCTYPE db [  
  <!ELEMENT db (project | employee)*>  
  <!ELEMENT project (title, budget, managedBy)>  
  <!ELEMENT employee (name, ssn, age)>  
  ...  
]>
```
Recursive DTDs

<DOCTYPE genealogy [
  <!ELEMENT genealogy (person*)>
  <!ELEMENT person ( name,
                      dateOfBirth,
                      person, -- mother
                      person
                      person -- father )>
...
]>  

- What is the problem with this?

Recursive DTDs cont’d.

<DOCTYPE genealogy [
  <!ELEMENT genealogy (person*)>
  <!ELEMENT person ( name,
                      dateOfBirth,
                      person?, -- mother
                      person?
                      person? )> -- father
...  
]>  

- What is now the problem with this?
Some Things are Hard to Specify

- Each employee element is to contain name, age and ssn elements in some order

```xml
<!ELEMENT employee
  ( (name, age, ssn) | (age, ssn, name) | (ssn, name, age) | ... )>
```

- Suppose there were many more fields!

Specifying ID and IDREF Attributes

```xml
<!DOCTYPE family [

<!ELEMENT family  (person)*>
<!ELEMENT person  (name)>
<!ELEMENT name   (#PCDATA)>
<!ATTLIST person id        ID        #REQUIRED
                      mother  IDREF    #IMPLIED
                      father  IDREF    #IMPLIED
                      children IDREFS   #IMPLIED>

]>```
Some Conforming XML data

```xml
<html>
<head></head>
<body>
<p>Some Conforming XML data</p>
<p>
    <family>
        <person id="jane" mother="mary" father="john">
            <name> Jane Doe </name>
        </person>
        <person id="john" children="jane jack">
            <name> John Doe </name>
        </person>
        <person id="mary" children="jane jack">
            <name> Mary Doe </name>
        </person>
        <person id="jack" mother="mary" father="john">
            <name> Jack Doe </name>
        </person>
    </family>
</body>
</html>
```

An Alternative XML DTD Specification

```xml
<!DOCTYPE family [
<!ELEMENT family (person)*>>
<!ELEMENT person (mother?, father?, children, name)>
<!ATTLIST person id ID #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT mother EMPTY>
<!ATTLIST mother idref IDREF #REQUIRED>
<!ELEMENT father EMPTY>
<!ATTLIST father idref IDREF #REQUIRED>
<!ELEMENT children EMPTY>
<!ATTLIST children idrefs IDREFS #REQUIRED>
]>
```
The revised XML data

```xml
<family>
  <person id = "jane">
    <name> Jane Doe </name>
    <mother idref = "mary"> </mother>
    <father idref = "john"> </father>
  </person>
  <person id = "john">
    <name> John Doe </name>
    <children idrefs = "jane jack"> </children>
  </person>
  ...
</family>
```

Mapping between XML and Tables
Towards XML-enabled DBMS

- Xml-enabled database system
  - Store XML data/documents into the database server
  - Query and search valid and well-formed XML
  - Generate XML data from the database server
  - Add XML capabilities in supporting database facilities
- XML has the potential to impact four important markets
  - Web integration
  - Web publishing
  - Application integration
  - Electronic commerce
Storing XML Data

- Enhance XML storage facilities in the database:
  - Utilities to load XML data into the database
  - Provide more efficient database storage (componentized storage, compression, indexing,...)
  - XML export tools from the server
  - Allow server-to-server replication of XML data

Querying and Searching XML Data

- Fine-grained access to XML documents
- Search XML data efficiently
  - Special SQL queries over valid + well-formed XML
  - Content-based indexing (e.g. Text indexes) for searching XML data efficiently
  - Support for XML query languages (e.g. XQL) on XML data
Generating and Manipulating XML

- Generate XML from the database server
  - Map ODMG, SQL92, SQL3 and PL/SQL datatypes to XML
  - Provide mappings between java, SQL and XML types
- Script XML content from the database
  - Allow SQL queries to return XML results
  - Provide embedded XML in stored procedures
  - Java scripting: support embedded XML in java
  - Common APIs to access any XML content in databases
Epilogue

- 1960's: Data Centric
- 1970's: Process Centric
- 1980's: Object Oriented
- 1990's: Component Based
- 2000's: XML?

Data was our First Focus

- Record Layouts
- Printer Layouts
- System Flow Charts
- Decision Tables

Batch Jobs were a Series of small Programs
Then we focused on Logic

- GOTO-Less Programming
- Structured Programming
- Top-Down Design

Programs became very large

Object Oriented Programming
Focused on Runtime Behavior

- Common terms for analysis and design
- Tightly coupled code

Code reuse was the Holy Grail, rarely achieved
Component Programming
Shifted the Focus to Interfaces

- Code Reuse
- IDE-Based Composition
- Limited Acceptance

Serialization Tied to Code

---

XML Returns the Focus to Data

- XML Wrappers for Incompatible Systems
- Industry-Specific Markup Languages
- XML for Persistent Data and Composition

XML Enables Middleware for Application-Specific Data
BIBLIOGRAPHY

- Sean McGrath. "XML by Example: Building E-Commerce Applications". Printice Hall, 1998