

# SGML

---

- ❑ Stands for *Standard Generalized Markup Language*
- ❑ Provides rules for defining a markup language
- ❑ An SGML document has two parts
  - A description of the structure given by Document Type Declaration (DTD)
  - A conformant document
- ❑ SGML tags can be nested
- ❑ This induces a *tree structure* on the document

# SGML-Brief History

---

- ❑ In the late sixties, IBM asked a researcher named Charles Goldfarb to build a system for storing, finding, managing, and publishing legal documents
- ❑ Goldfarb found that there were many systems within IBM that could not interchange information
- ❑ Goldfarb and his colleagues at IBM, Ed Mosher and Ray Lorie, set out to solve this problem
- ❑ The result was vendor or application software independent GML (Generalized Markup Language)
- ❑ GML was later standardized and thus became SGML

## HTML vs XML

---

- HTML is based on SGML
- Primarily defines layout and formatting
- There is a fixed collection of markup tags with a fixed semantics
- No mechanism for extension for other situations/applications

## An HTML Document

---

```
<html>
<head>
  <title>
    This is a memo
  </title>
</head>
<body>
  <b> To: </b> Dr. Beck <br>
  <b> From: </b> Dr. Gehlot <br>
  <b> Re: </b> CSC3400 <br>
  <hr>
  <p> It is progressing well. </p>
</body>
</html>
```

## Same in XML

---

```
<memo>
  <to> Dr. Beck </to>
  <from> Dr. Gehlot </from>
  <re> CSC 3400 </re>
  <content> It is progressing
    well </content>
</memo>
```

## XML

---

- ❑ Stands for eXtensible Markup Language
- ❑ Simplification (subset) of SGML
- ❑ No fixed collection of markup tags
- ❑ Users can define their own *elements* tailored for their kind of information
- ❑ **Tags** markup XML **elements**
- ❑ A starting tag **<element\_name>** marks up the beginning of an element, and an ending tag **</element\_name>** marks up the end
- ❑ Elements may have *attributes*

## XML (cont.)

---

- ❑ All start tags must have end tags
- ❑ Elements must be properly nested
- ❑ Every document must contain a root element
- ❑ Fully separates semantic information and layout

## XML Document Type Definition (DTD)

---

- ❑ Specification of syntax of an XML document
- ❑ Two major components are:
  - Element declarations
  - Attribute declarations
- ❑ Can be defined separately and referenced by document
- ❑ An XML document is *valid* if its structure conforms to its associated DTD.
- ❑ An online validator:  
<http://www.stg.brown.edu/service/xmlvalid/>

## DTD for Memo

---

```
<!ELEMENT memo
  (to, from, re, content) >
<!ELEMENT to (#PCDATA) >
<!ELEMENT from (#PCDATA) >
<!ELEMENT re (#PCDATA) >
<!ELEMENT content (#PCDATA) >
```

## DTD for Memo (cont.)

---

- The DTD can be included in the memo file (say myMemo.xml) as follows:

```
<?xml version="1.0" ?>
<!DOCTYPE memo [...]>
```

This also declares **memo** to be the root element

- Alternatively, it can be defined in a file, say, memo.dtd and referenced as follows:

```
<?xml version="1.0" ?>
<!DOCTYPE memo SYSTEM "memo.dtd">
```

## DTD Operators

- More intricate structure can be defined by making use of the following operators:
  - choice: (...|...|...)
  - sequence: (...,,,...)
  - optional: ...?
  - zero or more: ...\*
  - one or more: ...+
- For example, **to+** means a valid memo may have one or more recipients.

## DTD Attributes

- General form:  
**<!ATTLIST *element-name attr-name attr-type attr-default ...*>**
- For example:  
**<!ATTLIST memo type (public | confidential) public>**
- Thus, **<memo>** or  
**<memo type="confidential">** or  
**<memo type="public">**
- DTD examples online:  
[http://www.w3schools.com/dtd/dtd\\_examples.asp](http://www.w3schools.com/dtd/dtd_examples.asp)

## DTD vs. Schema

---

- ❑ The term *schema* has been used in db context to describe the layout of data
- ❑ Recall a DTD describes the layout of an XML document
- ❑ In this sense, a DTD is a schema
- ❑ As people started to build DTDs for different applications, several drawbacks surfaced
- ❑ This led to development of XML Schema which is a replacement (refinement) for (of) DTDs

## DTD vs. Schema (cont.)

---

- ❑ Some limitations of DTDs are:
  - Very limited support for built-in types
  - Very limited support for modularity and reuse
  - Potential naming *conflicts* or *collisions* and no support for namespaces
  - Separate syntax for description
- ❑ Consider the example XML document `po.xml` from <http://www.w3c.org/TR/xmlschema-0/#PO>

```

<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    <city>Mill Valley</city>
    <state>CA</state>
    <zip>90952</zip>
  </shipTo>
  <billTo country="US">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    <city>Old Town</city>
    <state>PA</state>
    <zip>95819</zip>
  </billTo>
  <comment>Hurry, my lawn is going wild!</comment>
  <items>
    <item partNum="872-AA">
      <productName>Lawnmower</productName>
      <quantity>1</quantity>
      <USPrice>148.95</USPrice>
      <comment>Confirm this is electric</comment>
    </item>
    <item partNum="926-AA">
      <productName>Baby Monitor</productName>
      <quantity>1</quantity>
      <USPrice>39.98</USPrice>
      <shipDate>1999-05-21</shipDate>
    </item>
  </items>
</purchaseOrder>

```

## XML Schema

- ❑ Similar to many programming languages, XML Schema provides mechanisms to build (complex) types.
- ❑ For modularity and reuse, it provides mechanisms for extensions and restrictions similar to what is found in OO languages
- ❑ An XML document that conforms to a schema is termed an *instance document*.
- ❑ This is similar to saying that an expression has a certain type or that an object is an instance of a class.

## XML Schema (cont.)

---

- ❑ An element in an XML document is *valid* according to a given schema if the element has the (schema) specified *type*.
- ❑ A document is *valid* if all its elements are valid according to a given schema
- ❑ Unlike DTD, a specific root element is not required (actually can't be done)

## XML Schema (cont.)

---

- ❑ Typically, an XML Schema consists of:
  - Global element(s) declaration (if only one then it is the root)
  - Complex type definitions
  - Simple type definitions
- ❑ Elements that contain subelements or carry attributes are said to have complex types
- ❑ Elements that contain numbers, strings, etc. are said to have simple types
- ❑ Attributes always have simple types
- ❑ Described using XML syntax (no separate special syntax)
- ❑ Tags: <element>, <complexType>, <simpleType>, etc.

## XML Schema: complexType

- A typical way to describe a **<complexType>** is as a **<sequence>** of **<element>**s
- It may also contain **<attribute>**s
- Attribute's **use** may be **required** or **optional** and may have **fixed** or **default** value
- Elements are declared by giving a **name** and a **type**
- Global (top level) elements may be **refrenced** but cannot themselves contain **ref**
- Non-global element occurrences may be controlled by **minOccurs** and **maxOccurs**

## Example: Complex Types

- A possible type for the **shipTo** element of po.xml

```
<complexType name="USAddress" >
  <sequence>
    <element name="name" type="string"/>
    <element name="street" type="string"/>
    <element name="city" type="string"/>
    <element name="state" type="string"/>
    <element name="zip" type="decimal"/>
  </sequence>
  <attribute name="country" type="string"/>
</complexType>
```

- With this, several elements can share a type
- **string** and **decimal** are built-in (simple) types.

## Example: Complex Types (cont.)

- A type for purchase order

```
<element name="comment" type="string"/>
<complexType name="PurchaseOrderType">
  <sequence>
    <element name="shipTo" type="USAddress"/>
    <element name="billTo" type="USAddress"/>
    <element ref="comment" minOccurs="0"/>
    <element name="items" type="Items"/>
  </sequence>
  <attribute name="orderDate" type="date"/>
</complexType>
```

- Here **comment** is a global element and can be referenced

## Inlining

- Types definitions can be *inlined*
- However, these *anonymous types* cannot be shared
- Useful if using a type only once
- In particular, anonymous types lack **type=** in an element (or attribute) declaration
- The (simple or complex) type definition is included as part of element declaration:

```
<element name=...>
  ...type definition...
</element>
```

## Example: Inlining

- Instead of:  
`<element name="shipTo" type="USAddress"/>`
- Write:  

```
<element name="shipTo">
  <complexType>
    <sequence>
      <element name="name" type="string"/>
      <element name="street" type="string"/>
      <element name="city" type="string"/>
      <element name="state" type="string"/>
      <element name="zip" type="decimal"/>
    </sequence>
    <attribute name="country" type="string"/>
  </complexType>
</element>
```

## More on complexType

- Another way is to use `<choice>`. There are many more. A good readable reference is:  
<http://www.w3c.org/TR/xmlschema-0/>
- ```
<complexType name= ... >
  <choice>
    <element ... />
    <element ... />
    ...
    <element ... />
  </choice>
  <attribute ... />
</complexType>
```

## XML Schema: simpleType

- There are a number of built-in simple types:
  - string
  - integer
  - decimal
  - date
  - time
  - NMTOKEN (XML 1.0)
  - And many more

## XML Schema: simpleType (cont.)

- New simple types can be derived from existing ones:
  - As a **list** (of a given **itemType**)  
E.g. `<list itemType="someSimpleType" />`
  - As a **union** (of a given **memberTypes**)  
E.g. `<union memberTypes="sType1 sType2 sType3" />`
  - As a **restriction** which include:
    - **enumeration** (of values)
    - **pattern** (regular expression)
    - **minInclusive**, **maxInclusive**, etc. (bounds on numbers)

## Example: Derived Simple Types

```
<simpleType name="TriState">
  <restriction base="string">
    <enumeration value="DE"/>
    <enumeration value="NJ"/>
    <enumeration value="PA"/>
  </restriction>
</simpleType>
```

## More Examples

```
<simpleType name="myInteger">
  <restriction base="integer">
    <minInclusive value="10000"/>
    <maxInclusive value="99999"/>
  </restriction>
</simpleType>

<simpleType name="SSNum">
  <restriction base="string">
    <pattern value="\d{3}-\d{2}-\d{4}"/>
  </restriction>
</simpleType>
```

## XML: Namespaces

---

- Different people writing schemas may define elements/types with the same name
- One way to avoid such collisions is to use *qualified names*
- We may have two people with the name **John**
- We can qualify their names, say, with their addresses to disambiguate:  
**USA.PA.123 Some Street.John** vs.  
**USA.DE.567 Another Street.John**
- XML (and some other languages) follow a similar collision avoidance scheme

## XML: Namespaces

---

- In XML, a qualified name has two parts: a prefix and a local name
- These two are separated by a colon (:)
- The prefix identifies a namespace (or address)
- Address for electronic documents is their URIs
- By convention, the prefix associated with the XML Schema language is given name **xsd**  
`<xsd:schema  
xmlns:xsd="http://www.w3.org/2001/XMLSchema">`
- Better to write: **xsd:element**, **xsd:string**, **xsd:complexType**, etc

## Two Large Examples

- <http://www.w3.org/TR/xmlschema-0/#po.xsd>
- [LOG.xsd](#) (an XML Log Schema for Digital Library Logging Analysis)
- Online schema and document validator:  
<http://www.validome.org/xml/>

## XML in DoD

- Air Tasking Order (ATO), which assigns specific Air Force units to specific missions. An ATO, produced daily, may contain thousands of sorties.
- With XML-MTF, units could subscribe to only receive the information on those missions they are assigned to perform. In addition, the information would be in a format easily recognized by mission planning tools.
- The following fragment is from a Tactical Report, or TACREP, which contains aircraft mission information in MTF (Message Text Format) format:

```
MSGID/TACREP/CTF 124//  
MAROP/011800Z/1/US/SUB/CL:WASHINGTON/NAME:SEAROVER  
/LM:4040N01100E//  
OPSUP/ACTTYP:ASW//  
AIROP/020200Z/6/US/FTR/F15/TN:401/LM:4130N01000E/CRS:180/SPD:600KPH  
/ALT:12000FT//  
OPSUP/ACTTYP:DCA//
```

- The italicized portion of the fragment can be rewritten in XML as follows:

```
<air_operations_data_segment>  
<air_operations_data>  
<day-time> 020200Z </day-time>  
<quantity> 6 </quantity>  
<country> US </country>  
<subject_type> FTR </subject_type>  
<aircraft_type> F15 </aircraft_type>  
<track_number> 401 </track_number>  
</air_operations_data>  
<supplementary_operations_data>  
<activity_type> DCA </activity_type>  
</supplementary_operations_data>  
</air_operations_data_segment>
```