

# CountryData Technologies for Data Exchange



---

Introduction to XML



# What is XML?

---

- **EX**tensible **M**arkup **L**anguage
  - Format is similar to HTML, but XML deals with data structures, while HTML is about presentation
- Open standard
  - Recommended by World Wide Web Consortium (W3C)



# XML document: an example

---

```
<?xml version="1.0" ?>
<books>
  <book bookID="BU1032" price="19.99">
    <title>The Busy Executive's Database Guide</title>
    <authors>
      <author authorID="A213-46-8915">Blue, Marjorie</author>
      <author authorID="A409-56-7008">Bennet, Abraham</author>
    </authors>
  </book>
  <book bookID="BU2075" price="2.99">
    <title>You Can Combat Computer Stress!</title>
    <authors>
      <author authorID="A213-46-8915">Blue, Marjorie</author>
    </authors>
  </book>
</books>
```



# Markup languages

---

- Text-based languages, where instructions and/or descriptions are inserted into data
- Both machine and human-readable



# XML, HTML, and SGML

---

- SGML (Standard Generalized Markup Language) is a broad specification for creating markup documents – essentially, a set of rules
- Has been in use since early 1980s
- Serves as a base for several other markup languages



# HTML

---

- HyperText Markup Language, designed around 1990 to support networked hypertext documents and became the foundation of World Wide Web
- An implementation of SGML, i.e. a defined set of markup instructions (tags) that browsers must understand to display Web pages



# XML

---

- Extensible Markup Language – extensible because, like SGML, allows users to define their own tags
- A simplified sub-set of SGML
- Stricter rules make it easier for humans to read and machines to process
- XML is infrastructure, not a solution!



# HTML vs XML

---

- HTML defines a set of tags that are used for presentation of hypertext documents. You use `<B>` to display text in bold font, `<IMG>` to insert an image, etc.
- XML has no predefined tags. You create your own tags and give them your own meaning.





# HTML vs XML

---

<HTML>

HTML tags

<H1>Hypertext Markup Language</H2>

<P>

You <B>must<B> use HTML <I>tags</I> for presentation of text.

</P>

</HTML>

---

<?xml version= "1.0" ?>

XML tags

<MyXmlDocument>

<MyStatement>

In XML, you use your own tags and give them your own meaning.

</MyStatement>

</MyXmlDocument>



# Multi-language compatibility

---

- XML documents are based on Unicode, which supports all of the world's written alphabets



# Basis for other standards

---

- Concrete implementations of XML
  - RSS (Really Simple Syndication)
  - MathML (Mathematical Markup Language)
  - CML (Chemical Markup Language)
  - SDMX (Statistical Data and Metadata Exchange)



# XML tags

---

- Must start with a letter (a-z, A-Z) or an underscore (\_)
- Must contain at least one letter
- Can contain letters, digits, hyphens, underscores, and periods



# Commonly used naming conventions

---

- **TimeSeries**
- **timeSeries**
- **time-series**
- **time\_series**
- It is not mandatory, but highly recommended, to use a consistent naming convention



# XML components

```
<?xml version="1.0" ?>
<books>
  <book bookID="BU1032" price="19.99">
    <title>The Busy Executive&quot;s Database Guide</title>
    <authors>
      <author authorID="A213-46-8915">Blue, Marjorie</author>
      <author authorID="A409-56-7008">Bennet, Abraham</author>
    </authors>
  </book>
  <book bookID="PS1372" price="21.59">
    <title>Computer Phobic AND Non-Phobic Individuals: Behavior Variations</title>
    <authors>
      <author authorID="A724-80-9391">MacFeather, Stearns</author>
      <author authorID="A756-30-7391">Karsen, Livia</author>
    </authors>
  </book>
</books>
```

**Declaration** points to `<?xml version="1.0" ?>`

**Root** points to `<books>`

**Entity** points to `<book bookID="BU1032" price="19.99">`

**Element** points to `<title>The Busy Executive&quot;s Database Guide</title>`

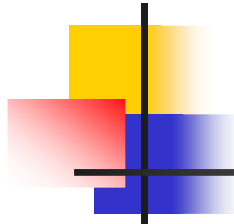
**Attribute** points to `price="21.59"`



# Declaration

---

- Optional but recommended
- Specifies XML version (currently 1.0) and encoding
- Encoding may vary but **utf-8** is most commonly used



# Root

---

- Also known as **root element** and **document element**
- Valid XML document must contain a single root, which encloses all other elements
- Should, but does not have to, have a unique name





# Element

---

- “A unit of XML data, delimited by tags”
- Can contain text
- Can contain other elements
- Can contain attributes
- Can be empty



# Attribute

---

- “A qualifier on an XML tag”
- Provides additional information about elements
- Must be unique in every element: cannot be repeated
- Single or double quote can be used to enclose value



# Entity

---

- An “escape sequence”, i.e. a sequence of characters used to represent characters that are illegal in content
  - **&lt;** - less than (<)
  - **&gt;** - greater than (>)
  - **&amp;** - ampersand (&)
  - **&quot;** - quotation mark (“ ”)
  - **&apos;** - apostrophe (’)



# Element content

---

```
<book>  
  <title>XML Databases</title>  
  <price>49.99</price>  
</book>
```

- The **book** element only contains other elements



# Simple content

---

```
<title>XML Databases</title>
```

```
<price>49.99</price>
```

- The **title** element contains text
- The **price** element contains a number



# Mixed content

---

`<authors>`

The list of authors includes

`<author>Ann Dull</author>`

and `<author>Michael O'Leary</author>`

`</authors>`

- The **Authors** element contains both other elements and text
- Rarely used in data structures



# Empty content

---

`<page-break/>`

`<page-break></page-break>`

`<book price="49.99"/>`

- These elements are empty: they carry no content between the opening and closing tags.



# Elements vs Attributes

---

```
<book title="XML Databases"  
price="49.99"/>
```

---

```
<book>  
  <title>XML Databases</title>  
  <price>49.99</price>  
</book>
```





## Elements vs Attributes (2)

---

- Data structures can be element-centric or attribute-centric.
- The choice of data model rests with the designer.
- Typically, elements and attributes are both used in the same document, although it is possible to only use elements.



## Elements vs Attributes (3)

---

- Elements are best used for information that
  - Can be repeated
  - Can be grouped
  - Can be extended



## Elements vs Attributes (4)

---

- Attributes are best used for information that
  - Is a simple property of the object: consists of a single value, such as ID or price
  - Is unique for the object: cannot be repeated



# PCDATA

---

- Parseable Character Data: any text that is not markup, including entities
- Elements containing data are declared as PCDATA
- Numerical data are typically represented as text for compatibility and readability



# CDATA

---

- CDATA (character data) sections are not parsed and can therefore contain reserved XML characters

**<condition>x <= 5</condition>** illegal

**<condition>x &lt;= 5</condition>** OK

**<condition>**

**<![CDATA[x <= 5]]>**

**</condition>** OK



# Elements vs Attributes

---

- Element-based structures are more verbose and can be more difficult to read due to deeper nesting.
- Simple properties represented as attributes mitigate these issues.
- Complex properties, which themselves are objects, should (and often have to) be represented as elements.



# Well-formed XML

---

- A “well-formed” XML document is a document that conforms to syntax rules:
  - XML documents must have a root element
  - Elements and attributes must have valid names
  - Elements must be properly terminated
  - Tags are case sensitive
  - Elements must be properly nested
  - Attribute values must always be quoted



## Well-formed XML (2)

---

- The rules are stricter than HTML
  - A closing tag is always required
- Strict rules mean faster validation and processing
- Generally, XML documents that are not well-formed cannot be processed in any way





# Name conflicts

---

- When processing XML documents, especially combining different documents, name conflicts may occur
- The same element or attribute may have one meaning in one document, but a different meaning in the other document



# Name conflict: an example

---

```
<books>
  <book price="19.99">
    <title>The Busy Executive's Database Guide</title>
    <authors>
      <author authorID="A213-46-8915">Blue, Marjorie</author>
      <author authorID="A409-56-7008">Bennet, Abraham</author>
    </authors>
  </book>
</books>
```

---

```
<employee>
  <title>Programmer Analyst</title>
  <name>Smith, John</name>
</employee>
```



# XML namespaces

---

- Provide a mechanism to avoid name conflicts
- Let you group elements and attributes by giving them a unique group identifier
- Elements and attributes must be unique within a namespace



# Namespace identifiers

---

- XML namespace is identified by a **URI** (Uniform Resource Identifier)
- Namespace declarations often resemble Web addresses because Web addresses are unique
- However, they do not have to be either valid addresses or follow the Web address syntax



# Namespace declarations

---

- Namespaces are declared using the **xmlns** attribute.
- Used on an element, this declaration applies to the element and its children.
- A prefix may be specified as a shortcut to the namespaces.



# Namespaces: an example

---

```
<books xmlns="http://www.mypublisher.com/book_namespace">
  <book price="19.99">
    <title>The Busy Executive's Database Guide</title>
    <authors>
      <author authorID="A213-46-8915">Blue, Marjorie</author>
      <author authorID="A409-56-7008">Bennet, Abraham</author>
    </authors>
  </book>
</books>
```

---

```
<emp:employee xmlns:emp="urn:mypublisher:employee_namespace">
  <emp:title>Programmer Analyst</emp:title>
  <emp:name>Smith, John</emp:name>
</emp:employee>
```



# Document Type Definition

---

- “A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes.”
- Originally designed for SGML.



# DTD: an example

---

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE books[
  <!ELEMENT books (book*)>
  <!ELEMENT book (title, authors)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT authors (author+)>
  <!ELEMENT author (#PCDATA)>
  <!ATTLIST book price CDATA #IMPLIED>
  <!ATTLIST author authorID CDATA #REQUIRED>]>
<books>
  <book price="19.99">
    <title>The Busy Executive's Database Guide</title>
    <authors>
      <author authorID="A213-46-8915">Blue, Marjorie</author>
      <author authorID="A409-56-7008">Bennet, Abraham</author>
    </authors>
  </book>
</books>
```





# Valid or well-formed?

---

- A **well-formed** XML document is said to be **valid** if it conforms to its DTD or *schema*.
- A well-formed document may not conform to its DTD, in which case it is invalid.



# Disadvantages of DTD

---

- Uses a custom non-XML syntax
- No support for certain XML features
  - Namespaces are not supported
- No data types
- Not extensible
- Difficult to build or read



# XML schema

---

- An XML-based alternative to DTD
- Like DTD, describes the structure of an XML document



# XML schema: an example

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.mypublisher.com/book_namespace"
  xmlns="http://www.mypublisher.com/book_namespace" elementFormDefault="qualified">
  <xsd:element name="books">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="book" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="title" minOccurs="1" maxOccurs="1" type="xsd:string"/>
              <xsd:element name="authors" minOccurs="1" maxOccurs="1">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="author" minOccurs="1" maxOccurs="unbounded">
                      <xsd:complexType>
                        <xsd:simpleContent>
                          <xsd:extension base="xsd:string">
                            <xsd:attribute name="authorID" type="xsd:string"/>
                          </xsd:extension>
                        </xsd:simpleContent>
                      </xsd:complexType>
                    </xsd:element>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
          <xsd:attribute name="price" type="xsd:int" use="optional"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```



# Simple element declaration

---

```
<xsd:element name="author" type="xsd:string" />
```

- Defines element that contains PCData, and the type of PCData.
- Most commonly used types include:
  - xsd:string
  - xsd:decimal
  - xsd:integer
  - xsd:boolean
  - xsd:date
  - xsd:time



# Attribute declaration

---

```
<xsd:attribute name="price" type="xsd:int" use="optional"/>
```

- Attributes are defined on elements
- **use** declares that the attribute:
  - **required** – must be present
  - **optional** – may or may not be present
  - **prohibited** – must not be present



# Facets

---

- A way to restrict legal values on a simple type
- A variety of restrictions are supported, e.g.
  - Enumeration
  - Range
  - Length
  - Pattern, etc



# Facets: enumeration

---

```
<xsd:element name="Gender">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Male"/>
      <xsd:enumeration value="Female"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

- This declares an element named **Gender**, with two possible values: Male and Female.





# Facets: range

---

```
<xsd:element name="Age">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="150"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

- This declares an element named **Age**, with valid integer values from 0 to 150. **minExclusive** and **maxExclusive** facets are also supported.



# Facets: length

---

```
<xsd:element name="LastName">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="2"/>
      <xsd:maxLength value="30"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

- This declares an element named **LastName**, with the minimum length of 2 characters and maximum length 30 characters.



# Facets: other constraints

---

- **whiteSpace** – specifies how white space is handled
- **totalDigits** – restricts the maximum number of digits in a value
- **fractionDigits** – restricts the maximum number of digits after the decimal point
- **pattern** – restricts the value based on regular expression matching



# Type declarations

---

- Instead of specifying types directly on the elements or attributes, it is possible to declare a type and then reuse it on multiple element/attribute declarations.



# Type declaration: example

---

```
<xsd:simpleType name="AgeType">  
  <xsd:restriction base="xsd:integer">  
    <xsd:minInclusive value="0"/>  
    <xsd:maxInclusive value="150"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

```
<xsd:element name="HusbandAge" type="AgeType"/>  
<xsd:element name="WifeAge" type="AgeType"/>
```



# Simple and complex types

---

- Simple types can be used to define:
  - Attributes
  - Simple elements, which cannot have attributes and cannot have element content
- Complex types define elements that may include attributes and/or other elements.



# Complex types: element content

---

- Three types of declarations can be used to define element content:
  - **xsd:choice** - defines a list of elements, one of which must appear
  - **xsd:all** - defines a list of elements, any of which may or may not appear once, in any order
  - **xsd:sequence** - defines a list of elements that must appear in the order specified



# xsd:choice

---

```
<xsd:complexType name="AddressType">
  <xsd:choice>
    <xsd:element name="LocalAddress" type="LocalAddressType"/>
    <xsd:element name="InternationalAddress"
      type="InternationalAddressType"/>
  </xsd:choice>
</xsd:complexType>
```

- Now, we can declare an element of type **AddressType**, which must include either an element named **LocalAddress**, or an element **InternationalAddress**.





# xsd:all

---

```
<xsd:complexType name="AddressListType">
  <xsd:all>
    <xsd:element name="HomeAddress" type="AddressType"
      minOccurs="1" maxOccurs="1"/>
    <xsd:element name="OfficeAddress" type="AddressType"
      minOccurs="0" maxOccurs="1"/>
  </xsd:all>
</xsd:complexType>
```

- Now, we can declare an element of type **AddressListType**, which must include an element **HomeAddress** and may include **OfficeAddress**.



# xsd:sequence

---

```
<xsd:complexType name="AuthorsListType">
  <xsd:sequence>
    <xsd:element name="author" minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

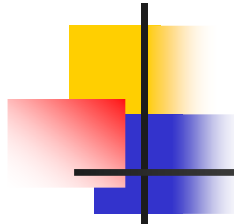
- Now, we can declare an element of type **AuthorsListType**, which must contain at least one element named **author**.



# XML Schema: with types

---

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.mypublisher.com/book_namespace"
  xmlns="http://www.mypublisher.com/book_namespace" elementFormDefault="qualified">
  <xsd:complexType name="bookListType">
    <xsd:sequence>
      <xsd:element name="book" type="BookType" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="BookType">
    <xsd:sequence>
      <xsd:element name="title" type="xsd:string" maxOccurs="1"/>
      <xsd:element name="authors" type="AuthorListType" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="price" type="xsd:float" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="AuthorListType">
    <xsd:sequence>
      <xsd:element name="author" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:string">
              <xsd:attribute name="authorID" type="xsd:string"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="books" type="bookListType"/>
</xsd:schema>
```

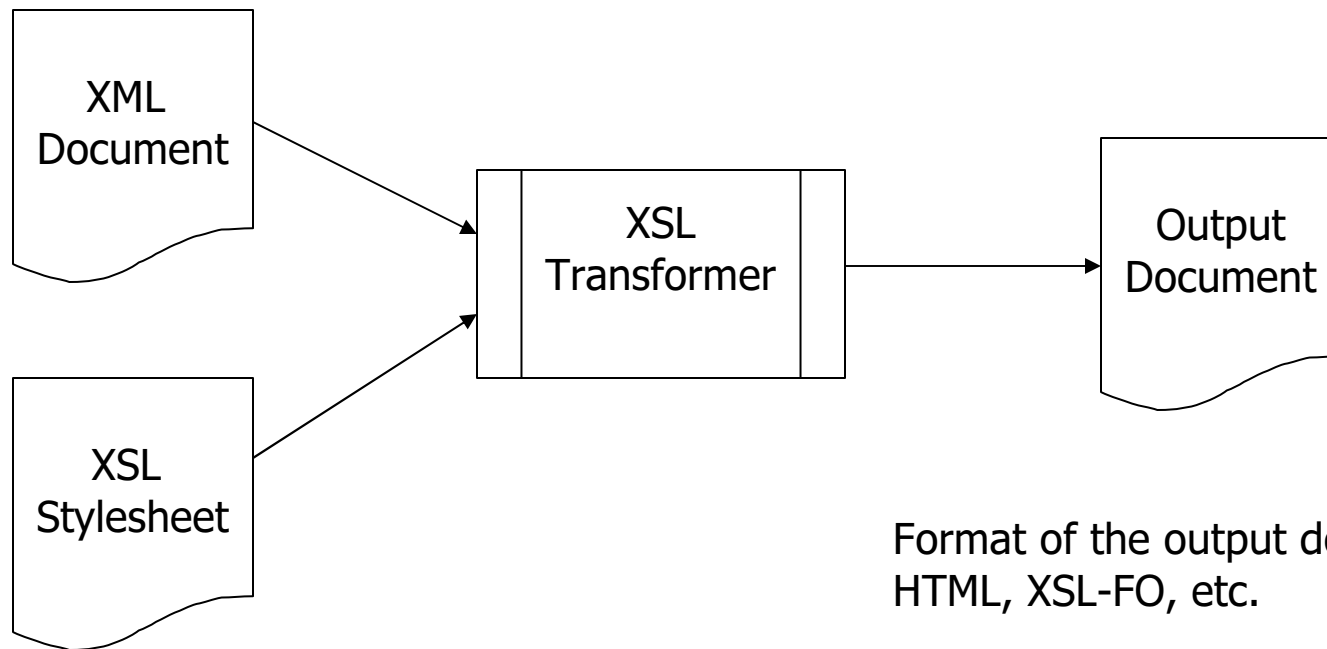


# XSL

---

- **EX**tensible **S**tylesheet **L**anguage
- An implementation of XML: a defined set of tags that is used to transform XML documents into other representations
- The most common mechanism for converting XML from one form to another, as well as into HTML and other representations.
- Conversions are carried out by software known as XSL transformer.

# XSL Transformations



Format of the output document: XML, HTML, XSL-FO, etc.



# XSL stylesheet: an example

---

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="books">
    <html>
      <body>
        <table border="1">
          <tr>
            <td><b>Title</b></td>
            <td><b>Authors</b></td>
            <td><b>Price</b></td>
          </tr>
          <xsl:for-each select="book">
            <tr>
              <td>
                <xsl:value-of select="title"/>
              </td>
              <td>
                <xsl:for-each select="authors/author">
                  <xsl:value-of select="."/>
                  <br/>
                </xsl:for-each>
              </td>
              <td>
                <xsl:value-of select="@price"/>
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```