What is XML Schema?

- The origin of schema
  - XML Schema documents are used to define and validate the content and structure of XML data
  - XML Schema was originally proposed by Microsoft, but became an official W3C recommendation in May 2001
    - [http://www.w3.org/XML/Schema](http://www.w3.org/XML/Schema)
Why Schema?

Separating Information from Structure and Format

Traditional Document: Everything is clumped together

“Fashionable” Document: A document is broken into discrete parts, which can be treated separately

Why Schema?

Schema Workflow
DTD vs. Schema

Limitations of DTD
- No constraints on character data
- Not using XML syntax
- No support for namespace
- Very limited for reusability and extensibility

Advantages of Schema
- Syntax in XML Style
- Supporting Namespace and import/include
- More data types
- Able to create complex data type by inheritance
- Inheritance by extension or restriction
- More …

Problems of XML Schema

- General Problem
  - Several-hundred-page spec in a very technical language

- Practical Limitations of expressibility
  - content and attribute declarations cannot depend on attributes or element context

- Technical Problem
  - The notion of “type” adds an extra layer of confusing complexity

…
XML.org Registry

- The XML.org Registry offers a central clearinghouse for developers and standards bodies to publicly submit, publish and exchange XML schemas, vocabularies and related documents.

An XML Document Example

```xml
<book isbn="0836217462">
  <title> ... </title>
  <author> ... </author>
  <qualification> ... </qualification>
</book>
```
The Example’s Schema

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
        <xs:element name="qualification" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

book.xsd

Referring to a Schema

- To refer to a DTD in an XML document, the reference goes before the root element:
  ```xml
  <?xml version="1.0"?>
  <!DOCTYPE rootElement SYSTEM "url">
  <rootElement> ... </rootElement>
  ```

- To refer to an XML Schema in an XML document, the reference goes in the root element:
  ```xml
  <?xml version="1.0"?>
  <rootElement
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="url.xsd">
    ...
  </rootElement>
  ```
The XSD Document

- Since the XSD is written in XML, it can get confusing which we are talking about
- The file extension is .xsd
- The root element is `<schema>`
- The XSD starts like this:
  ```xml
  <?xml version="1.0"?>
  <xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  </schema>
  ```

The `<schema>` element may have attributes:

- xmlns:xs="http://www.w3.org/2001/XMLSchema"
  - This is necessary to specify where all our XSD tags are defined
- elementFormDefault="qualified"
  - This means that all XML elements must be qualified
“Simple” and “Complex” Elements

- A “simple” element is one that contains text and nothing else
  - A simple element cannot have attributes
  - A simple element cannot contain other elements
  - A simple element cannot be empty
  - However, the text can be of many different types, and may have various restrictions applied to it

- If an element isn’t simple, it’s “complex”
  - A complex element may have attributes
  - A complex element may be empty, or it may contain text, other elements, or both text and other elements

Defining a Simple Element

- A simple element is defined as
  \[
  \text{\textless} \text{xs:element} \text{ name="name" type="type"} \text{\textgreater}
  \]
  where:
  - \text{name} is the name of the element
  - the most common values for \text{type} are \text{xs:boolean} \text{xs:integer} \text{xs:date} \text{xs:string} \text{xs:decimal} \text{xs:time}

- Other attributes a simple element may have:
  - \text{default="default value"} if no other value is specified
  - \text{fixed="value"} no other value may be specified
Defining an Attribute

- Attributes themselves are always declared as simple types

- An attribute is defined as
  \[
  \text{<xs:attribute name="name" type="type" />}
  \]
  where:
  - \text{name} and \text{type} are the same as for \text{xs:element}

- Other attributes a simple element may have:
  - \text{default="default value"} if no other value is specified
  - \text{fixed="value"} no other value may be specified
  - \text{use="optional"} the attribute is not required (default)
  - \text{use="required"} the attribute must be present

Restrictions, or “Facets”

- The general form for putting a restriction on a text value is:
  \[
  \text{<xs:element name="name">}
  \text{<xs:simpleType>}
  \text{<xs:restriction base="type">}
  \text{... the restrictions ...}
  \text{</xs:restriction>}
  \text{</xs:simpleType>}
  \text{</xs:element>}
  \]

- For example:
  \[
  \text{<xs:element name="age">}
  \text{<xs:simpleType>}
  \text{<xs:restriction base="xs:integer">}
  \text{<xs:minInclusive value="20"/>}
  \text{<xs:maxInclusive value="100"/>}
  \text{</xs:restriction>}
  \text{</xs:simpleType>}
  \text{</xs:element>}
  \]
Restrictions, or “Facets”

- The “age” element is a simple type with a restriction. The acceptable values are: 20 to 100

- The example above could also have been written like this:

  ```xml
  <xs:element name="age" type="ageType"/>
  <xs:simpleType name="ageType">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="20"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
  ```

Restrictions on numbers

- **minInclusive** number must be ≥ the given *value*
- **minExclusive** number must be > the given *value*
- **maxInclusive** number must be ≤ the given *value*
- **maxExclusive** number must be < the given *value*
- **totalDigits** number must have exactly *value* digits
- **fractionDigits** number must have no more than *value* digits after the decimal point
Restrictions on strings

- **length**: the string must contain exactly \textit{value} characters
- **minLength**: the string must contain at least \textit{value} characters
- **maxLength**: the string must contain no more than \textit{value} characters
- **pattern**: the \textit{value} is a regular expression that the string must match
- **whiteSpace**: not really a "restriction" - tells what to do with whitespace
  - \textit{value}="preserve": Keep all whitespace
  - \textit{value}="replace": Change all whitespace characters to spaces
  - \textit{value}="collapse": Remove leading and trailing whitespace, and replace all sequences of whitespace with a single space

Enumeration

- An enumeration restricts the value to be one of a fixed set of values

Example:

```xml
<xs:element name="season">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Spring"/>
      <xs:enumeration value="Summer"/>
      <xs:enumeration value="Autumn"/>
      <xs:enumeration value="Fall"/>
      <xs:enumeration value="Winter"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```
Complex Elements

A complex element is defined as

```
<xs:element name="name">
  <xs:complexType>
    ... information about the complex type...
  </xs:complexType>
</xs:element>
```

Example:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstName" type="xs:string"/>
      <xs:element name="lastName" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Another example – using a type attribute

```
<xs:element name="employee" type="personinfo"/>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstName" type="xs:string"/>
    <xs:element name="lastName" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```
**xs:sequence**

- We've already seen an example of a complex type whose elements must occur in a specific order:

  ```xml
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstName" type="xs:string" />  
        <xs:element name="lastName" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  ```

**xs:all**

- `xs:all` allows elements to appear in any order.

- Despite the name, the members of an `xs:all` group can occur once or not at all.

- You can use `minOccurs="n"` and `maxOccurs="n"` to specify how many times an element may occur (default value is 1):
  - In this context, `n` may only be 0 or 1.
Text Element with Attributes

- If a text element has attributes, it is no longer a simple type

```xml
<xs:element name="population">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="year" type="xs:integer"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Empty Elements

- Empty elements are (ridiculously) complex

```xml
<xs:complexType name="counter">
  <xs:complexContent>
    <xs:extension base="xs:integer"/>
    <xs:attribute name="count" type="xs:integer"/>
  </xs:complexContent>
</xs:complexType>
```
Mixed Elements

- Mixed elements may contain both text and elements
- We add `mixed="true"` to the `xs:complexType` element
- The text itself is not mentioned in the element, and may go anywhere (it is basically ignored)

```xml
<xs:complexType name="paragraph" mixed="true">
  <xs:sequence>
    <xs:element name="someName" type="xs:anyType"/>
  </xs:sequence>
</xs:complexType>
```

Example

```xml
<letter>
  Dear Mr.<name>John Smith</name>.
  Your order <orderid>1032</orderid> will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>

<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:integer"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```
Extensions

- You can base a complex type on another complex type

```xml
<xs:complexType name="newType">
  <xs:complexContent>
    <xs:extension base="otherType">
      ...new stuff...
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Predefined String Types

- Recall that a simple element is defined as:

```xml
<xs:element name="name" type="type"/>
```

- Here are a few of the possible string types:
  - `xs:string` - a string
  - `xs:normalizedString` - a string that doesn't contain tabs, newlines, or carriage returns
  - `xs:token` - a string that doesn't contain any whitespace other than single spaces

- Allowable restrictions on strings:
  - `enumeration`, `length`, `maxLength`, `minLength`, `pattern`, `whiteSpace`
Predefined Date and Time Types

- **xs:date** - A date in the format `CCYY-MM-DD`, for example, `2003-11-05`

- **xs:time** - A date in the format `hh:mm:ss` (hours, minutes, seconds)

- **xs:dateTime** - Format is `CCYY-MM-DDThh:mm:ss`

- **Allowable restrictions on dates and times:**
  - enumeration, minInclusive, maxExclusive, maxInclusive, pattern, whiteSpace

Predefined Numeric Types

- Here are some of the predefined numeric types:
  - `xs:decimal`
  - `xs:positiveInteger`
  - `xs:byte`
  - `xs:negativeInteger`
  - `xs:short`
  - `xs:nonPositiveInteger`
  - `xs:int`
  - `xs:nonNegativeInteger`
  - `xs:long`

- **Allowable restrictions on numeric types:**
  - enumeration, minInclusive, maxExclusive, maxInclusive, fractionDigits, totalDigits, pattern, whiteSpace
XML Parsers

- Every XML application is based on a parser

- Two types of XML documents:
  - Well-formed: if it obeys the syntax of XML
  - Valid: if it conforms to a proper definition of legal structure and elements of an XML document

- Two types of XML Parsers:
  - Non-validating
  - Validating

Interfacing XML Documents with XML Applications

- Two Ways
  - Object-based: DOM
  - Event-based: SAX
Available XML Schema-supported Parsers

- Apache® Xerces 2 Java/C++ free
  - Validating/Non-validating
  - DOM and SAX
- Microsoft® XML Parser 4.0 free
  - DOM and SAX
- TIBCO® XML Validate commercial
  - SAX-based implementation
  - Suitable in a streaming runtime environment
- SourceForge.net® JBind 1.0 free
  - A data binding framework linking Java and XML
  - The runtime environment is used to read/write XML documents for validation, accessing and manipulating XML data
- And many many more…

Schema Features

- Object-Oriented Features
  - Distinction between types and instances. Schema type definitions are independent of instance declarations
  - Inheritance
- Relational information Features
  - Like tree structure; having parents and children
  - Strongly-typed: strong typing available in the specification
Xml schema enable translations from XML documents to databases.

What is XML Software Development process?

1. Begin with developing content model using XML Schema or DTD
2. Edit and validate XML documents according to the content model
3. Finally, the XML document is ready to be used or processed by an XML enabled framework
What is XML Software Development process?

References

- W3School XSD Tutorial
  - http://www.w3schools.com/schema/default.asp
- MSXML 4.0 SDK
- Several online presentations
Reading List

- W3School XSD Tutorial
  - http://www.w3schools.com/schema/default.asp