XML Schema

Sang Shin
Michèle Garoche
www.javapassion.com
"Learning is fun!"

Agenda

- Motivation
- Simple types
- Complex types
- Element vs. Attribute
- Occurrences
- List type
- Union type
- Explicit vs. Implicit
- Element content
- Annotation
- Choices and Group
- Namespaces

Motivations for XML Schema

XML Schema Status

- W3C recommendation
- http://www.w3.org/XML/Schema.html
- XML schema aware tools
  - Several free and commercial versions available (Check the above site)
  - NetBeans 5.5 and after (Free)
  - XMLSpy (Commercial, Not Free)
  - DTD to XML Schema conversion tools

Motivations of XML Schema (over DTD)

- Provides more powerful and flexible schema language than DTD
  - DTD is designed for publishing industry
- Represents XML document syntax in XML language
  - XML tools can be readily used
- Support non-textual data types, i.e. Integer
  - Important to B2B, e-Commerce
- Handle complex syntax
  - With DTD, you cannot specify “a value of an element is in the range of 10 to 100”

Valid vs. Schema-valid

- XML schema was not part of XML 1.0
- When DTD is more popular than XML Schema in the past
  - XML document that is validated with DTD is “valid”
  - XML document that conforms to XML schema is “schema-valid”
- Now given that DTD is rarely used these days, “valid” usually means “schema-valid”
- XML document that conforms to a particular XML schema is called “instance document” of that schema

Definitions vs. Declarations

Definition and Declaration

- Definition
  - Creation of a type (both simple and complex types)
- Declaration
  - Usage of a defined type

Example: Definition and Declaration

```
<!-- Definition: Creation of a type -->
<xsd:simpleType name="zipUnion">
    <xsd:union memberTypes="USState listOfMyIntType"/>
</xsd:simpleType>

<!-- Declaration: Usage of a type -->
<xsd:element name="zips" type="zipUnion"/>
```

Schema Data Types: Simple Types & Complex Types

Schema Data Types

- Simple type definition
  - Do not have sub-elements (child elements)
    - Do not have "*element*" sub-elements
    - Do not have "*attribute*" sub-elements
  - Can be either predefined simple type or derived from predefined simple type
- Complex type definition
  - Have either "*element*" sub-elements or "*attribute*" sub-elements or both

Simple Types

Built-in Primitive Simple Types included in the XML Schema specification

- duration, dateTime, time, date, gYearMonth, gYear, gMonthDay, gDay, gMonth
- boolean, base64Binary, hexBinary, float, decimal, double, anyURI, Qname, NOTATION
- string

Built-in Derived Simple Types included in the XML Schema specification

- normalizedString, token, language, Name, NMTOKEN, NCName, NMTOKENS, ID, IDREF, ENTITY, IDREFS, ENTITIES
- integer, nonPositiveInteger, negativeInteger, long, int, short, byte, nonNegativeInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte, positiveInteger

Usage Examples of Predefined Simple types

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://xml.netbeans.org/schema/newXmlSchema"
    xmlns:tns="http://xml.netbeans.org/schema/newXmlSchema"
    elementFormDefault="qualified">
<xsd:element name="Title" type="xsd:string"/>
<xsd:element name="Heading" type="xsd:string"/>
<xsd:element name="Topic" type="xsd:string"/>
<xsd:element name="Price" type="xsd:decimal"/>
<xsd:attribute name="focus" type="xsd:string"/>
</xsd:schema>
```

Derived Simple Type

- Derived from existing (predefined or derived) simple types
- Typically restricting existing simple type
  - The legal range of values for a new type is subset of the ones of an existing type
  - Existing type is called *base* type
  - Use *restriction* element along with facets to restrict the range of values
  - Facets are rules of restriction
- May be a list of simple or derived simple types
- May be an union of simple or derived simple types

Example of Derived Simple Type 1 (Numeric range)

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

- Defining *myInteger* type whose range of value is between 10000 and 99999 inclusive
- *minInclusive* and *maxInclusive* are facets that are applied to *integer* base type

Example of Derived Simple Type 2 (Regular expression)

```
<xsd:simpleType name="SKU">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d{3}-[A-Z]{2}"/>
    </xsd:restriction>
</xsd:simpleType>
```

- Defining new derived simple type called *SKU*
- *pattern* is a facet that is applied to *string* base type
  - Regular expression
  - three digits followed by a hyphen followed by two upper-case ASCII letters

Example of Derived Simple Type 3 (Enumeration)

```
<xsd:simpleType name="USState">
    <xsd:restriction base="xsd:string">
            <xsd:enumeration value="AK"/>
            <xsd:enumeration value="AL"/>
            <xsd:enumeration value="AR"/>
            <!-- and so on ... -->
     </xsd:restriction>
</xsd:simpleType>
```

- *enumeration* facet limits a simple type to a set of distinct values
- *USState* type is enumeration of AK, AL, AR, ...

Complex Types

Complex Type

- Defined using “*complexType*” element
- Typically contain
  - *element* sub-elements
  - *attribute* sub-elements
  - element references

complexType Example 1

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

complexType Example 1

- Definition of *USAddress* type
- It contains 5 sub-element's (in sequence) and one attribute sub-element
- USAddress definition contains only declarations involving simple types: *string*, *decimal*, and *NMTOKEN*

complexType Example 2

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

complexType Example 2

- Definition of *PurchaseOrder* type
- Contains type declarations referencing complex types, e.g. *USAddress, Items*
- Contains type declaration referencing simple type, e.g. *comment*
- Contains type declaration referencing “pre-defined” simple types: *date*

Elements vs. Attributes

Element vs. Attribute

- Elements can reference both simple types or complex types
- Attributes can reference only simple types
  - Because they cannot contain other sub-elements

ref Attribute

- To use an existing element or attribute rather than declaring a new element or attribute
- Existing element must be global element - an element that is declared under root element

ref Example

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" ...>
   <xsd:element name="purchaseOrder" type="tns:PurchaseOrderType"/>
   <xsd:element name="comment" type="xsd:string"/>
   <xsd:element name="newElement"/>
   <xsd:complexType name="PurchaseOrderType">
      <xsd:sequence>
         <xsd:element name="shipTo" type="tns:USAddress"/>
         <xsd:element name="billTo" type="tns:USAddress"/>
         <xsd:element ref="tns:comment" minOccurs="0"/>
         <xsd:element name="items"  type="tns:Items"/>
      </xsd:sequence>
      <xsd:attribute name="orderDate" type="xsd:date"/>
   </xsd:complexType>
</xsd:schema>
```

Occurences

Occurrences of Elements

- *minOccurs*
- *maxOccurs*
- *fixed = "Hannah"*
  - If the element appears (optional), the value must be "Hannah" (if not it is "invalid")
  - Else the value is set to "Hannah" by the parser
- *default = "Hannah"*
  - If the element appears (optional), the value is set to what is specified
  - Else the value is set to "Hannah" by the parser

Example

```
<xsd:element name="test" type="tns:testall"/>
<xsd:complexType name="testall">
   <xsd:sequence>
      <xsd:element name="test1" type="xsd:string" minOccurs="1"
                                                  maxOccurs="1"/>
      <xsd:element name="test2" type="xsd:string" minOccurs="1"
                                                  maxOccurs="1" fixed="Hannah"/>
      <xsd:element name="test3" type="xsd:string" minOccurs="2"
                                                  maxOccurs="unbounded"/>
      <xsd:element name="test4" type="xsd:string" minOccurs="0" maxOccurs="1"
                                                  fixed="Hannah"/>
      <xsd:element name="test5" type="xsd:string" minOccurs="0" maxOccurs="1"
                                                  default="Hannah"/>
      <xsd:element name="test6" type="xsd:string" minOccurs="0" maxOccurs="2"
                                                  default="Hannah"/>
      <xsd:element name="test7" type="xsd:string" minOccurs="0" maxOccurs="0"/>
   </xsd:sequence>
</xsd:complexType>
```

Occurrences of Attributes

- Attributes can occur once or not at all
- "*use*" attribute
  - required
  - optional
  - prohibited
- "*fixed*", "*default*" or none attribute when "*use*" attribute is optional

Example: use usage

```
<xsd:element name="test" type="tns:testall"/>
<xsd:complexType name="testall">
   <xsd:sequence>
      <xsd:element name="t1" type="tns:testall1"/>
      <xsd:element name="t2" type="tns:testall2"/>
      <xsd:element name="t3" type="tns:testall3"/>
      <xsd:element name="t4" type="tns:testall4"/>
      <xsd:element name="t5" type="tns:testall5"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="testall1">
   <xsd:attribute name="test1" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="testall2">
  <xsd:attribute name="test2" type="xsd:string" use="optional"/>
</xsd:complexType>
<xsd:complexType name="testall3">
   <xsd:attribute name="test3" type="xsd:string" use="optional" fixed="37"/>
</xsd:complexType>
<xsd:complexType name="testall4">
   <xsd:attribute name="test4" type="xsd:string" use="optional"
default="37"/>
</xsd:complexType>
<xsd:complexType name="testall5">
   <xsd:attribute name="test5" type="xsd:string" use="prohibited"/>
</xsd:complexType>
```

Example: fixed usage

```
<xsd:complexType name="USAddress">
        <xsd:sequence>
                <xsd:element name="name"   type="xsd:string"/>
                <xsd:element name="street" type="xsd:string"/>
                ...
         </xsd:sequence>
         <xsd:attribute name="country" type="xsd:NMTOKEN"
                  use="optional" fixed="US"/>
</xsd:complexType>
```

- Appearance of a *country* attribute is optional
- Its value must be *US* if it does appear
- If it does not appear, parser will create a country attribute with value *US*

Example: default usage

```
<xsd:complexType name="USAddress">
        <xsd:sequence>
              <xsd:element name="name"   type="xsd:string"/>
              <xsd:element name="street" type="xsd:string"/>
              ...
         </xsd:sequence>
         <xsd:attribute name="country" type="xsd:NMTOKEN"
               use="optional" default="US"/>
</xsd:complexType>
```

- Appearance of a *country* attribute is optional
- Its value can be whatever if it does appear
- If it does not appear, parser will create a country attribute with value *US*

Attributes

- Enumeration
  - *simpleType* element with *base* attribute, except boolean
  - *base* attribute specifies the type

Example of Enumeration

```
<xsd:complexType name="ContentsType">
    <xsd:sequence>
        <xsd:element name="Chapter" maxOccurs="unbounded">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="Heading" type="xsd:string" minOccurs="0"/>
                    <xsd:element name="Topic" maxOccurs="unbounded">
                        <xsd:complexType>
                            <xsd:attribute name="subSections" type="xsd:integer"/>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="focus" default="Java">
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                            <xsd:enumeration value="XML" />
                            <xsd:enumeration value="Java" />
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:attribute>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

Complete Example

Complete Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.oreilly.com/catalog/javaxml/"
    xmlns:JavaXML="http://www.oreilly.com/catalog/javaxml/"
    elementFormDefault="qualified">
    <xsd:element name="Book" type="JavaXML:BookType"/>
    <xsd:complexType name="BookType">
        <xsd:sequence>
            <xsd:element name="Title" type="xsd:string"/>
            <xsd:element name="Contents" type="JavaXML:ContentsType"/>
            <xsd:element name="Copyright" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
```

Complete Example (continued)

```
    <xsd:complexType name="ContentsType">
        <xsd:sequence>
            <xsd:element name="Chapter" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="Heading" type="xsd:string" minOccurs="0"/>
                        <xsd:element name="Topic" maxOccurs="unbounded">
                            <xsd:complexType>
                                <xsd:attribute name="subSections" type="xsd:integer"/>
                            </xsd:complexType>
                        </xsd:element>
                    </xsd:sequence>
                    <xsd:attribute name="focus" default="Java">
                        <xsd:simpleType>
                            <xsd:restriction base="xsd:string">
                                <xsd:enumeration value="XML" />
                                <xsd:enumeration value="Java" />
                            </xsd:restriction>
                        </xsd:simpleType>
                    </xsd:attribute>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="SectionBreak" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType></xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

List Type

List Type

- *Simple type*
- Comprised of sequences of atomic simple types
  - <xsd:list itemType="myInteger"/>
- Three built-in list types
  - *NMTOKENS*, *IDREFS*, *ENTITIES*
- User defined List type
  - Derive from atomic types
- facets
  - *length, minLength, maxLength, enumeration*

Example of List Type

- Schema

```
<xsd:schema … xmlns:tns="
http://xml.netbeans.org/examples/PurchaseOrder" …>
<xsd:simpleType name="myInteger">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="10000"/>
      <xsd:maxInclusive value="99999"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="listOfMyIntType">
   <xsd:list itemType="tns:myInteger"/>
</xsd:simpleType>
</xsd:schema>
```

- Usage of the type in an instance Document

```
<listOfMyIntType>20003 15037 95977
95945</listOfMyIntType>
```

Example: List Type with Facet (1)

```
<xsd:schema ... xmlns:tns="http://xml.netbeans.org/examples/PurchaseOrder" ...>
    <xsd:simpleType name="USState">
     <xsd:restriction base="xsd:string">
       <xsd:enumeration value="AK"/>
       <xsd:enumeration value="AL"/>
       <xsd:enumeration value="AR"/>
       ...
     </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="USStateList">
        <xsd:list itemType="tns:USState"/>
    </xsd:simpleType>

    <xsd:simpleType name="SixUSStates">
        <xsd:restriction base="tns:USStateList">
            <xsd:length value="6"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:element name="sixStates" type="tns:SixUSStates"/>
</xsd:schema>
```

Example: List Type with Facet (2)

- In order to define a list of exactly six US states (*SixUSStates*):
  - We first define an enumeration of US states (*USState*)
  - Then a new list type called *USStateList* from *USState*
  - *We* derive *SixUSStates* by restricting *USStateList* to only six items
- Usage
  - <sixStates>PA NY CA NY LA AK</sixStates>

Union Type

Union Type

- *Simple type*
- Enables an element or attribute value to be one or more instances of one type drawn from the union of multiple atomic and list types
- facets: *pattern* and *enumeration*

Union Type for Zipcodes: Schema

```
<xsd:schema ... xmlns:tns="http://xml.netbeans.org/examples/PurchaseOrder" ...>
<xsd:element name="SomeLocations" type="tns:Locations"/>
<xsd:simpleType name="myInteger">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="10000"/>
      <xsd:maxInclusive value="99999"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="listOfMyIntType">
   <xsd:list itemType="tns:myInteger"/>
</xsd:simpleType>
<xsd:simpleType name="USState">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="AK"/>
    <xsd:enumeration value="AL"/>
    <xsd:enumeration value="AR"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="zipUnion">
    <xsd:union memberTypes="tns:USState tns:listOfMyIntType"/>
</xsd:simpleType>
<xsd:complexType name="Locations">
   <xsd:sequence>
      <xsd:element name="zips" maxOccurs="unbounded" type="tns:zipUnion"/>
   </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Union Type for Zipcodes: Usage

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:SomeLocations  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xmlns:ns0='http://xml.netbeans.org/examples/PurchaseOrder'
xsi:schemaLocation='http://xml.netbeans.org/examples/PurchaseOrder/newXmlSche
ma.xsd'>
    <ns0:zips>AK</ns0:zips>
    <ns0:zips>23000</ns0:zips>
    <ns0:zips>AL</ns0:zips>
</ns0:SomeLocations>
```

Explicit Type vs. Implicit Type

Explicit Type vs. Implicit Type

- *Explicit type*
  - One in which a name is given to the type
  - Element that uses the type is generally defined in a different section of the schema
  - Object-oriented in that same explicit type is used as the type for several different elements
- Implicit type (sometimes called as nameless type or anonymous type)
  - Use when the type is not used by multiple elements

Example of Explicit Type

```
<!-- Type has a name zipUnion -->
<xsd:simpleType name="zipUnion">
    <xsd:union memberTypes="tns:USState tns:listOfMyIntType"/>
</xsd:simpleType>

<!-- zipUnion type is used in other parts of Schema document -->
<element name="zips" type="tns:zipUnion">
...
<element name="theOtherZip"s type="tns:zipUnion">
...
<element name="theThirdZips" type="tns:zipUnion">
```

Example of Implicit Type

```
<xsd:element name="comment" type="xsd:string"/>
<xsd:complexType name="Items">
   <xsd:sequence>
      <xsd:element name="item" minOccurs="0"
                   maxOccurs="unbounded">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="productName"
                            type="xsd:string"/>
               <xsd:element name="quantity">
                  <xsd:simpleType>
                     <xsd:restriction base="xsd:positiveInteger">
                        <xsd:maxExclusive value="100"/>
                     </xsd:restriction>
                  </xsd:simpleType>
               </xsd:element>
               <xsd:element name="USPrice" type="xsd:decimal"/>
               <xsd:element ref="tns:comment" minOccurs="0"/>
               <xsd:element name="shipDate" type="xsd:date"
                            minOccurs="0"/>
            </xsd:sequence>
            <xsd:attribute name="partNum" type="tns:SKU"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="SKU">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-[A-Z]{2}"/>
   </xsd:restriction>
</xsd:simpleType>
```

Legend

Explicit complex type

Implicit complex type

Explicit simple type

Implicit simple type

anyType

anyType

- *Base type from which all simple and complex types are derived*
- Does not constrain its contents in any way
- Default type when no type is specified
  - *<xsd:element name="anything" type="xsd:anyType" />* is same as
  - *<xsd:element name="anything"/>*
- Best practice suggests "Use more constrained types whenever possible"

Annotation

Annotation

- *Appears at the beginning of most schema constructions*
- Can have two sub-elements
  - *documentation*
  - *appInfo*
- *documentation*
  - For human readable materials
- *appInfo*
  - For tools, stylesheets and other applications

Example of Annotation

```
<xsd:element name="internationalPrice">
  <xsd:annotation>
      <xsd:documentation> element declared with anonymous type
      </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
 <xsd:annotation>
      <xsd:documentation> empty anonymous type with 2 attributes
          </xsd:documentation>
 </xsd:annotation>
 <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
          <xsd:attribute name="currency" type="xsd:string" />
          <xsd:attribute name="value"    type="xsd:decimal" />
      </xsd:restriction>
 </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

Choices & Group

Choice and Group

- Choice
  - Only one of its children to appear in an instance
- Group
  - Grouping a group of elements
  - Further constraints
    - sequences
- All
  - Appears zero or once
  - In any order

Choice and Sequence Groups (1)

```
<xsd:element name="comment" type="xsd:string"/>
<xsd:complexType name="PurchaseOrderType">
   <xsd:sequence>
      <xsd:choice>
         <xsd:group ref="tns:shipAndBill" />
         <xsd:element name="singleUSAddress" type="tns:USAddress" />
      </xsd:choice>
      <xsd:element ref="tns:comment" minOccurs="0"/>
   </xsd:sequence>
   <xsd:attribute name="orderDate" type="xsd:date" />
</xsd:complexType>
```

Choice and Sequence Groups (2)

```
<xsd:group name="shipAndBill">
   <xsd:sequence>
      <xsd:element name="shipTo" type="tns:USAddress"
/>
      <xsd:element name="billTo"
type="tns:USAddress" />
   </xsd:sequence>
</xsd:group>
<xsd:complexType name="USAddress">
   <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="state" type="xsd:string"/>
      <xsd:element name="zip" type="xsd:decimal"/>
   </xsd:sequence>
   <xsd:attribute name="country" type="xsd:NMTOKEN"
```

Example of *all*

```
<xsd:complexType name="PurchaseOrderType">
    <xsd:all>
        <xsd:element name="shipTo" type="tns:USAddress"/>
        <xsd:element name="billTo" type="tns:USAddress"/>
        <xsd:element ref="tns:comment" minOccurs="0"/>
        <xsd:element name="items"  type="tns:Items" />
    </xsd:all>
    <xsd:attribute name="orderDate" type="xsd:date" />
</xsd:complexType>
...
```

Schema Namespaces

Types of Namespaces

- target Namespace
  - Namespace for the XML schema document itself
- source Namespaces
  - Namespaces external to the XML schema document

targetNamespace

- It is the namespace that is going to be assigned to the schema you are creating
  - The names defined in a schema are said to belong to its target namespace
- It is the namespace a document instance uses to access the types it declares

targetNamespace

- Each schema has one target namespace and possibly many source namespaces

Example 1: Sample Schema

```
<xsd:schema targetNamespace="http://www.SampleStore.com/Account"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:ACC="http://www.SampleStore.com/Account">
  <xsd:element name="InvoiceNo" type="xsd:positiveInteger"/>
  <xsd:element name="Product" type="ACC:ProductCode"/>
  <xsd:simpleType name="ProductCode">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Z]{1}d{6}"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Example 1: Explanation

- The targetNamespace name is *http://www.SampleStore.com/Account*, which contains the *InvoiceNo*, *ProductID*, and *ProductCode* names in its namespace
- The *schema*, *element*, *simpleType*, *pattern*, *string*, and *positiveInteger* belong to source namespace *http://www.w3.org/2001/XMLSchema*
- The targetNamespace also happens to be one of the source namespaces because the name *ProductCode* is used in defining other names.

Importing a Schema with schemaLocation

Example 2: Sample Schema with no schema location

```
<!-- This is the same schema you saw in Example1 -->
<xsd:schema targetNamespace="http://www.SampleStore.com/Account"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:ACC="http://www.SampleStore.com/Account">
 <xsd:element name="InvoiceNo" type="xsd:positiveInteger"/>
 <xsd:element name="Product" type="ACC:ProductCode"/>
 <xsd:simpleType name="ProductCode">
  <xsd:restriction base="xsd:string">
   <xsd:pattern value="[A-Z]{1}d{6}"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:schema>
```

Why schemaLocation?

- The schema file uses several source namespaces but how does it know where to get the schema files of the source namespaces?

Example 2: Explanation

- Example 2 (same schema as Example 1) does not need to specify locations of source schema files
  - For the overall "schema of schemas," *http://www.w3.org/2001XMLSchema*, you need not specify a location because it is well known
  - For the source namespace *http://www.SampleStore.com/Account,* you do not need to specify a location since it also happens to be the name of the target namespace that is being defined in this file.

Example 3: Schema with schema location

```
<xsd:schema targetNamespace="http://www.SampleStore.com/Account"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:ACC="http://www.SampleStore.com/Account"
      xmlns:PART="http://www.PartnerStore.com/PartsCatalog">
<xsd:import namespace="http://www.PartnerStore.com/PartsCatalog"
schemaLocation="http://www.ProductsStandards.org/repository/alpha.xsd"/>
  <xsd:element name="InvoiceNo" type="xsd:positiveInteger"/>
  <xsd:element name="Product" type="ACC:ProductCode"/>
  <xsd:simpleType name="ProductCode">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Z]{1}d{6}"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="stickyGlue" type="PART:SuperGlueType"/>
</xsd:schema>
```

Example 3: Explanation

- The PART namespace needs to be imported using the *import* declaration element whose *schemaLocation* attribute specifies the location of the file that contains the schema because
  - Is not a well-known namespace
  - Is not a targetNamespace

Summary

Summary

- Status
- Motivation
- Namespaces
- Vocabularies
  - element
  - complexType
  - attribute
  - simpleType
  - Enumeration

References

- XML Schema Primer on W3C Recommendation 28 October 2004, Edited by David Fallside, Priscilla Walmsley http://www.w3.org/TR/xmlschema-0/
- “Java and XML” written by Brett McLaughlin, O’Reilly, June 2000 (First edition), Chapter 4 “Constraining XML”, XML Schema section, page 108-123

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"