XML Schema (XSD) Reference

Purpose:	A quick reference for using the leading XML Schema language, XSD
Document group:	Software Development Approach for .Net

XSD is for ...

... defining the structure of data.

Principally, this allows people to come to an agreement about the structure of data so that the data may be exchanged. This could be at the organisation level, between organisations, or across an industry or other group. For example, geologists might define a standard way of describing fossils, that there should be a date of find, a location, in degrees longitude and latitude, the era the fossil belongs to, etc.

Technically, the structure that is defined is the allowable XML documents. XML instance documents are "validated" as conforming to the XSD schema(s).

XSD Reference Example

This schema ...

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:annotation>
  <xsd:documentation xml:lang="en">
   Purchase order schema for Example.com.
   Copyright 2000 Example.com. All rights reserved.
  </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

 <xsd:element name="comment" type="xsd:string"/>

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

 <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>

 <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="comment"   minOccurs="0"/>
      <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
     </xsd:sequence>
     <xsd:attribute name="partNum" type="SKU" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>

 <!-- Stock Keeping Unit, a code for identifying products -->
 <xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
   <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
 </xsd:simpleType>

</xsd:schema>
```

... can be used to ensure that this XML instance document is valid:

```
<?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>
```

W3C 2001 > Recommendation XML Schema Part 0: Primer

XSD Template

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema id="TemplateSchema"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://tempuri.org/TemplateSchema.xsd"
    xmlns:tns="http://tempuri.org/TemplateSchema.xsd"
    elementFormDefault="qualified"
    >
</xsd:schema>
```

XSD Concepts

Declarations Versus Definitions

Definitions create new types (simple and complex types). Definitions have representation in the Schema only.

Declarations describe the content models of elements and attributes. Declarations are an association between a name and the set of constraints for the specific declaration. Declarations have representation in an Instance document only.

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XML Schemas [derived]

Complex Type Versus Simple Type

A simple type defines values only.

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

A complex type defines an element which: contains other elements; and/or has attributes.

```
<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>
```

Global Versus Local/Inline

Global declarations and definitions are those that are the immediate children of the <xsd:schema> root element.

Local declarations and definitions have a parent other than <xsd:schema>.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <!-- Global Declarations -->
 <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

 <xsd:element name="comment" type="xsd:string"/>

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

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

 <!-- snip -->

 <!-- Another Global Definition -->
 <xsd:complexType name="Items">
  <xsd:sequence>

   <!-- A Local/Inline Element Declaration -->
   <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="productName" type="xsd:string"/>

      <!-- A Local/Inline Element Declaration -->
      <xsd:element name="quantity">

       <!-- A Local/Inline Definition of the Element's Content -->
       <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="comment"   minOccurs="0"/>
      <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
     </xsd:sequence>

     <!-- A local attribute declaration -->
     <xsd:attribute name="partNum" type="SKU" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

Place global declarations and definitions before the reference to it.

Named Versus Unnamed (or "Anonymous")

A named declaration or definition is one which has a name attribute.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <!-- A Named Declaration -->
 <xsd:element name="comment" type="xsd:string"/>

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

 <!-- A Named Definition -->
 <xsd:complexType name="Items">
  <xsd:sequence>

   <!-- A Named Declaration -->
   <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="productName" type="xsd:string"/>

      <!-- A Named Declaration -->
      <xsd:element name="quantity">

       <!-- An Unnamed Definition of the Element's Content -->
       <xsd:simpleType>
        <xsd:restriction base="xsd:positiveInteger">
         <xsd:maxExclusive value="100"/>
        </xsd:restriction>
       </xsd:simpleType>
      </xsd:element>
      <xsd:element name="USPrice"  type="xsd:decimal"/>

      <!-- An Unnamed Declaration -->
      <xsd:element ref="comment"   minOccurs="0"/>
      <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
     </xsd:sequence>

     <!-- A Named Declaration -->
     <xsd:attribute name="partNum" type="SKU" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

XSD Rules

XSD and other Schema languages

Use XSD, not Document Type Definitions (DTD's) nor XDR to validate your XML files.

DTD have too many limitations (eg it's not in XML form and doesn't work well with namespaces).

XML-Data Reduced (XDR) schema was interim schema language offered to developers while the W3C worked on a draft implementation of XSD.

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XML Schemas

I have heard "RelaxNG" is a popular, easy to use, but not written in XML, schema language.

Validating an XML Instance Document against a Schema

There are three ways to validate an XML instance document against a XSD Schema: By XML Processor initiation: Sharing a common namespace; Processor Hints.

Processor Initiation

To validate an XML instance against a schema you can just programmatically, or through a user interface, point he XML processor to the two (or more) documents and just tell it to validate.

Common Namespace

Steps:

1. In the Schema create a targetNamespace

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://example.org/employee/">
```

2. In the instance document declare a namespace that corresponds to the target namespace in the schema.

```
<tns:employee xmlns:tns="http://example.org/employee/">
```

Processor Hints

In Addition to a common namespace, Using schemaLocation

In the instance document provide hints as to the location of the schema. Use the schemaLocation attribute from the XMLSchema-instance namespace.

```
<!-- XML instance document -->
<p:Person
   xmlns:p="http://contoso.com/People"
   xmlns:v="http://contoso.com /Vehicles"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation=
     "http://contoso.com/People
      http://contoso.com/schemas/people.xsd
      http://contoso.com/schemas/Vehicles
      http://contoso.com/schemas/vehicles.xsd
      http://contoso.com/schemas/People
      http://contoso.com/schemas/people.xsd">
   <name>John</name>
   <age>28</age>
   <height>59</height>
   <v:Vehicle>
      <color>Red</color>
      <wheels>4</wheels>
      <seats>2</seats>
   </v:Vehicle>
</p:Person>
```

Note that schemaLocation is only useable when there is a common namespace (a target namespace in the schema which is declared in the XML instance document).

The value of the schemaLocation attribute is a list of namespace and schema pairs. The first URI reference in each pair is a namespace name, and the second is the location of a schema that describes that namespace.

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XML Schemas [Example]

When no namespace is used, use noNamespaceSchemaLocation

In the instance document provide hints as to the location of the schema. Use the noNamespaceschemaLocation attribute from the XMLSchema-instance namespace.

```
<!-- Instance Document -->
<catalog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <book xsi:noNamespaceSchemaLocation="http://www.example.com/MyData.xsd"
         id="bk101">
      <title>Presenting XML</title>
      <author>Richard Light</author>
   </book>
</catalog>
```

The referenced .xsd file will be used to validate the `<book>` node and any of its children nodes that are not namespace prefixed.

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XML Schemas

Local Files

Referencing Local Files

If you want to reference schema locally, whether you're using noNamespaceSchemaLocation or schemaLocation, escape it properly. That is, use a "file://" prefix and every backslash, "\" becomes a double forward slash, "//".

```
C:\Documents and Settings\All Users\Application Data\My
Application\MyData.xsd

xsi:noNamespaceSchemaLocation="file://C://Documents and Settings//All
Users//Application Data//My Application//MyData.xsd"
```

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XML Schemas

Referencing Files in the same directory

Reference files in the same directory, noNamespaceSchemaLocation or schemaLocation, by just specifying the file name.

```
xsi:noNamespaceSchemaLocation="MyData.xsd"
```

Namespaces in XSD Schema and Instance Documents

In Schema it is the **global** declarations and definitions that either belong to no namespace or the targetnamespace.

In Schema the targetNamespace applies only to the **global** Element declarations and All definitions. Local declarations, element and attributes, are associated with the global elements. So local declarations only indirectly belong to the targetNamespace.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:po="http://www.example.com/PO1"
        targetNamespace="http://www.example.com/PO1"
        elementFormDefault="unqualified"
        attributeFormDefault="unqualified">

 <!-- purchaseOrder in targetNamespace -->
 <element name="purchaseOrder" type="po:PurchaseOrderType"/>

 <!-- comment in targetNamespace -->
 <element name="comment" type="string"/>

 <!-- PurchaseOrderType in targetNamespace -->
 <complexType name="PurchaseOrderType">
  <sequence>

   <!-- shipTo not in targetNamespace -->
   <element name="shipTo"    type="po:USAddress"/>
   <element name="billTo"    type="po:USAddress"/>
   <element ref="po:comment" minOccurs="0"/>
   <!-- etc. -->
  </sequence>
  <!-- etc. -->
 </complexType>

 <!-- USAddress in targetNamespace -->
 <complexType name="USAddress">
  <sequence>
   <!-- name not in targetNamespace -->
   <element name="name"   type="string"/>
   <element name="street" type="string"/>
   <!-- etc. -->
  </sequence>
 </complexType>
 <!-- etc. -->
</schema>
```

This is only a John Bentley way of understanding things.

To validate an XML instance document which does not use namespaces at all, you must provide a schema with no target namespace.

W3C 2001 > Recommendation XML Schema Part 0: Primer

Type definitions (complexType and simpleType) are placed in one symbol space. Element declarations are placed in a second symbol space and attribute declarations are placed in a third symbol space. Hence, you can have a type and an element and an attribute all with the same name!

Costello 2002 > XML Schema Tutorial

Illegal : Same name and same Symbol Space but different type

```
<xsd:element name="foo">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="bar" type="xsd:string"/>            ...
            <xsd:element name="bar" type="xsd:integer"/>
</xsd:sequence>
    </xsd:complexType
</xsd:element>
```

Costello 2002 > XML Schema Tutorial

Legal: Same name and same Symbol Space and same type

```
<xsd:element name="foo">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="bar" type="xsd:string"/>            ...
            <xsd:element name="bar" type="xsd:string"/>
       </xsd:sequence>
    </xsd:complexType
</xsd:element>
```

Costello 2002 > XML Schema Tutorial

Locating a Schema Document on the Web

There may or may not be a schema retrievable via the namespace name.

W3C 2001 > Recommendation XML Schema Part 1: Structures Section 4.3.2

Schema Structure

Declare and Define Elements Inline

With Simple Content Only

The basic form is with a type association.

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

When declaring an element with an inline definition, if it just contains content (and no child elements), use the simpleType:

```
<xsd:element name="quantity">
  <xsd:simpleType>
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:maxExclusive value="100"/>
    </xsd:restriction>
  </xsd:simpleType>
```

```
</xsd:element>
```

With Child Elements Only

When declaring an element with an inline definitions, if it contains other elements, use complexType then wrap the child element declarations in one of: xsd:sequence, xsd:choice, or xsd:all

Compositor	Definition
xsd:sequence	An ordered sequence of contained items
xsd:choice	A choice of the contained items
xsd:all	All of the contained items in any order

```
<xsd:element name="item">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="productName" type="xsd:string" />
      <xsd:element name="quantity" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Skonnard 2002 > The XML Files A Quick Guide to XML Schema

Containing Elements and having attributes

When defining a complex type the attribute declarations must follow the element declarations. That is, the attribute declarations must come after the compositor.

```
<!-- XML Instance we want -->
<cloud cover="broken">
  <height>1500</height>
</cloud>
```

```
<!-- XSD Schema -->
<xsd:element name="cloud">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="height" type="xsd:nonNegativeInteger" />
    </xsd:sequence>
    <xsd:attribute name="cover" type="xsd:string" />
  </xsd:complexType>
</xsd:element>
```

Empty Content

To **allow** empty element content use the anyType type. As anyType is the default it can be omitted.

```
<!-- XML Instances we want -->

<coolAndCalm />
<coolAndCalm></coolAndCalm>

<!-- and -->
<coolAndCalm>any old shit</coolAndCalm>
```

```
<!-- Schema -->

<xsd:element name="coolAndCalm" type="xsd:anyType"/>

<!-- Equivalent -->
<xsd:element name="coolAndCalm" />
```

To **Force** empty content define a complex Type, omitting the elements.

```
<!-- In XSD Schema -->
<xsd:element name="airport">
  <xsd:complexType>
    <xsd:attribute name="icao" type="xsd:string" />
  </xsd:complexType>
</xsd:element>

<!-- In XML Instance -->
<airport icao="YSSY" />
```

Or:

```
<!-- In XSD Schema -->
<xsd:element name="airport">
  <xsd:complexType>
  </xsd:complexType>
</xsd:element>

<!-- In XML Instance -->
<airport />
```

To **allow** the Instance Author to flag empty content explicitly use the nillable attribute.

```
<!-- In XSD Schema -->
<xsd:element name="coolAndCalm" nillable="true"/>
```

Forces

```
<!-- In XML Instance -->
<occurrenceConstraintsExample
xmlns="http://tempuri.org/OccurrenceConstraints.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
..
<coolAndCalm xsi:nill="true" ></coolAndCalm>

<!-- Or -->
<coolAndCalm xsi:nill="true" />
...
```

Containing only content (no child elements) and having attributes

```
<aircraft engines="2">Boeing 737</aircraft>
```

```
<xsd:element name="aircraft">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string" >
        <xsd:attribute name="engines" type="xsd:integer" />
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

With Child elements in any order and with many occurences

Use choice nested in a sequence.

```
<xsd:element name="html">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="head">
        <xsd:complexType>
          <xsd:sequence minOccurs="0" maxOccurs="unbounded">
            <xsd:choice>
              <xsd:element name="meta"  />
              <xsd:element name="script" />
            </xsd:choice>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="body" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Validates:

```
<html xmlns="http://tempuri.org/xhtmlplay.xsd">
  <head>
    <meta />
    <script />
    <meta />
    <script />
  </head>
  <body>
  </body>
</html>
```

Declare and Define Attributes Inline

When declaring an attribute with an inline definition, use the **simpleType** element.

```
<xs:attribute name="myHolidayLocationTemperature">
  <xs:simpleType>
   <xs:restriction base="xs:integer">
    <xs:minInclusive value="60"/>
    <xs:maxInclusive value="95"/>
   </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

Declare and Define Attributes, More Clichés

Associating Declarations with other Declarations or Types

Association between declarations and a global named declaration

Associate a named, globally declared element or attribute by using the `ref` attribute within a declaration.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://example.org/employee/"
  targetNamespace="http://example.org/employee/">

  <xsd:element name="employee">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:name"/>
        <xsd:element ref="tns:hiredate"/>
        <xsd:element ref="tns:salary"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <!-- Named Global Elements -->
  <xsd:element name="name"/>
  <xsd:element name="hiredate"/>
  <xsd:element name="salary"/>
</xsd:schema>
```

```
<xs:element name="eyeColor" type="eyeColorType" />
<xs:complexType name="eyeColorType">
   <xs:attribute ref="blue" />
   <xs:attribute name="light" />
</xs:complexType>

<xs:attribute name="blue" />
```

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XML Schemas

Association between a declaration and a global named type definition

Associate a named, global complex or simple definition using the `type` attribute within a declaration.

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

 <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>
```

```
     ....
     <xsd:attribute name="partNum" type="SKU" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>

 <!-- Stock Keeping Unit, a code for identifying products -->
 <xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
   <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
 </xsd:simpleType>
```

Association between an Element declaration and an Element Group

Element groups can only contain elements.

Just use a "Ref" within the referencing declaration.]

```
<xsd:element name="CD" >
      <xsd:complexType>
                <xsd:sequence>
                         <xsd:group ref="PublicationElements"/>
<xsd:element name="RecordingStudio" type="string"/>
                </xsd:sequence>
      </xsd:complexType>
</xsd:element>
<xsd:group name="PublicationElements">
        <xsd:sequence>
            <xsd:element name="Title" type="xsd:string"/>
            <xsd:element name="Author" type="xsd:string"
maxOccurs="unbounded"/>
            <xsd:element name="Date" type="xsd:string"/>
        </xsd:sequence>
</xsd:group>
```

Costello 2002 > XML Schema Tutorial

Occurrence Constraints

For elements use the minOccurs and maxOccurs attributes.

```
<xsd:element name="freedomFighter">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="firstName" type="xsd:string"
        maxOccurs="1" minOccurs="1" />

      <!-- maxOccurs and minOccurs defaults to 1 so you don't need them -->
      <xsd:element name="lastName" type="xsd:string"/>

      <!-- weaponOfChoice is optional -->
      <xsd:element name="weaponOfChoice" minOccurs="0">
        <xsd:complexType>

          <!-- Make this choice a maximum of two times -->
          <xsd:choice maxOccurs="2">
            <xsd:element name="globalJustice" />
            <xsd:element name="cruiseMissile" />
            <xsd:element name="suicideBombing" />
          </xsd:choice>
        </xsd:complexType>
      </xsd:element>

    </xsd:sequence>

  </xsd:complexType>
</xsd:element>
```

Valid weaponOfChoice in Instance:

```
  <weaponOfChoice>
    <cruiseMissile />
    <cruiseMissile />
  </weaponOfChoice>
```

Invalid weaponOfChoice in instance:

```
  <weaponOfChoice>
    <cruiseMissile />
    <cruiseMissile />
    <cruiseMissile />
  </weaponOfChoice>
```

For attributes use the "use" attribute. It has possible values of optional, prohibited, and required. Optional is the default.

```
<xsd:element name="freedomFighter">
  <xsd:complexType>
    <xsd:sequence>
      <!-- Element declarations ommitted -->
    </xsd:sequence>
    <xsd:attribute name="ideology" type="xsd:string" use="optional" />
    <xsd:attribute name="killed" type="xsd:integer" use="required" />
  </xsd:complexType>
</xsd:element>
```

Valid freedom fighter element:

```
<freedomFighter killed="3">
...
</freedomFighter>
```

Qualifying Components

In an instance document global elements, by default, need to be qualified with the target namespace, and local elements must remain unqualified.

However, you can control whether a local element needs to be qualified with the form attribute.

```
<!-- Schema -->
```

```
<xsd:element name="employee">
  <xsd:complexType>
    <xsd:sequence>
      <!-- local element declarations -->
      <xsd:element name="name" form="qualified"/>
      <xsd:element name="hiredate"/>
      <xsd:element name="salary" form="qualified"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- Instance document -->
<tns:employee xmlns:tns="http://example.org/employee/">
  <tns:name>Monica</tns:name>
  <hiredate>1997-12-02</hiredate>
  <tns:salary>42000.00</tns:salary>
</tns:employee>
```

Skonnard 2002 > The XML Files A Quick Guide to XML Schema

In an instance document attributes, by default, will not be qualified.

```
<!-- Valid -->
<tns:employee xmlns:tns="http://example.org/employee/"
            id='555-12-3434'>
    <name>Monica</name>
</tns:employee>
```

However, as with elements, you can change this behaviour through the form or attributeFormDefault attributes if you would rather make local attributes qualified. Unlike elements you can't implicitly qualify attributes with a namespace - attributes must be explicitly qualified or belong to no namespace.

Skonnard 2002 > The XML Files A Quick Guide to XML Schema

There is also file level control for whether local elements and attributes need to be qualified for not.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://example.org/employee/"
  elementFormDefault="qualified">
 •••
</xsd:schema>
```

Skonnard 2002 > The XML Files A Quick Guide to XML Schema

Casing

XSD schemas validate instance documents in a case sensitive way.

```
<!-- An element declaration and definition like this ... -->
<xsd:element name="extent" >
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Few" />
      <xsd:enumeration value="Scattered" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<!-- ... Will not allow this to be valid ... -->
```

```
<extent>few</extent>

<!-- ... But WILL allow this... -->
<extent>Few</extent>
```

Multiple Schema Documents

Not only can you reuse named types within a single schema, you can also reuse named types across schemas through the `include` and `import` elements.

Skonnard 2002 > The XML Files A Quick Guide to XML Schema

Include, for schemas in the same targetNamespace.

Import, for schemas in a different targetNamespace.

The <redefine> element does the same thing as an <include> element (i.e., it allows you to access components in other schemas, provided they have the same namespace), plus it enables you to redefine one or more components (simpleType, complexType, attributeGroup, or group)

Costello 2002 > XML Schema Tutorial

Inheritance

Use complextContent for inheriting complex types.

By Extension has the effect of appending.

```
<xsd:complexType name="Publication">
     <xsd:sequence>
        <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="Author" type="xsd:string"
maxOccurs="unbounded"/>
        <xsd:element name="Date" type="xsd:gYear"/>
     </xsd:sequence>
</xsd:complexType >

<xsd:complexType name="BookPublication">
    <xsd:complexContent>
        <xsd:extension base="Publication">
            <xsd:sequence>
                <xsd:element name="ISBN" type="xsd:string"/>
                <xsd:element name="Publisher" type="xsd:string"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType >
```

By Restriction narrows the possiblilities.

```
<xsd:complexType name="Publication">
        <xsd:sequence>
                <xsd:element name="Title" type="xsd:string"
maxOccurs="unbounded"/>
```

```
                <xsd:element name="Author" type="xsd:string"
maxOccurs="unbounded"/>
                <xsd:element name="Date" type="xsd:gYear"/>
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name= "SingleAuthorPublication">
    <xsd:complexContent>
        <xsd:restriction base="Publication">
            <xsd:sequence>
                <xsd:element name="Title" type="xsd:string"
maxOccurs="unbounded"/>
                <xsd:element name="Author" type="xsd:string"/>
                <xsd:element name="Date" type="xsd:gYear"/>
            </xsd:sequence>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

Element Substitution

Substitution groups must be global, and have the same type as the head, or a derived type as the head.

```
<xsd:element name="Publication" type="PublicationType"/>
<xsd:element name="Book" substitutionGroup="Publication" type="BookType"/>
<xsd:element name="Magazine" substitutionGroup="Publication"
type="MagazineType"/>
<xsd:element name="BookStore">
    <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Publication" maxOccurs="unbounded"/>
            </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Costello 2002 > XML Schema Tutorial

Type Susbitution

A base type can be substituted by any derived type.

In the instance document to indicate that the content is not the *source type*, but rather a *derived type*, we need to specify the derived type that is being used. The attribute 'type' comes from the XML Schema Instance (xsi) namespace.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                     targetNamespace="http://www.books.org"
                     xmlns="http://www.books.org"
                     elementFormDefault="unqualified">    <xsd:complexType
name="PublicationType">
        <xsd:sequence>
            <xsd:element name="Title" type="xsd:string"/>
            <xsd:element name="Author" type="xsd:string"
maxOccurs="unbounded"/>
            <xsd:element name="Date" type="xsd:year"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="BookType">
        <xsd:complexContent>
            <xsd:extension base="PublicationType">
                <xsd:sequence>
                    <xsd:element name="ISBN" type="xsd:string"/>
                    <xsd:element name="Publisher" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Publication"  maxOccurs="unbounded"
type="PublicationType"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

```
<?xml version="1.0"?>
<bk:BookStore xmlns:bk="http://www.books.org"
                         xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
                         xsi:schemaLocation=
                                    "http://www.books.org
                                     BookStore.xsd">
        <Publication>
                <Title>Staying Young Forever</Title>
                <Author>Karin Granstrom Jordan, M.D.</Author>
                <Date>1999</Date>
        </Publication>
        <Publication xsi:type="bk:BookType">
                <Title>Illusions The Adventures of a Reluctant
Messiah</Title>
                <Author>Richard Bach</Author>
                <Date>1977</Date>
                <ISBN>0-440-34319-4</ISBN>
                <Publisher>Dell Publishing Co.</Publisher>
        </Publication>
        <Publication xsi:type="bk:BookType">
                <Title>The First and Last Freedom</Title>
                <Author>J. Krishnamurti</Author>
                <Date>1954</Date>
                <ISBN>0-06-064831-7</ISBN>
                <Publisher>Harper &amp; Row</Publisher>
        </Publication>
</bk:BookStore>
```

Costello 2002 > XML Schema Tutorial

Keys and Uniqueness

Use a key to validate the instance document as: having a unique field; that exits; is not nillable.

```
<xsd:key name="PK">
    <xsd:selector xpath="bk:Book"/>
    <xsd:field xpath="bk:ISBN"/>
</xsd:key >
```

ISBN acts as a key.

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
                     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                     xsi:schemaLocation=
                                           "http://www.books.org
                                            BookStore.xsd">
        <Book>
                <Title>My Life and Times</Title>
                <Author>Paul McCartney</Author>
                <Date>1998</Date>
                <ISBN>1-56592-235-2</ISBN>
<Publisher>McMillin Publishing</Publisher>
        </Book>
        <Book>
                <Title>Illusions The Adventures of a Reluctant
Messiah</Title>
                <Author>Richard Bach</Author>
                <Date>1977</Date>
                <ISBN>0-440-34319-4</ISBN>                <Publisher>Dell
Publishing Co.</Publisher>
        </Book>
        <Book>
                <Title>The First and Last Freedom</Title>
                <Author>J. Krishnamurti</Author>
                <Date>1954</Date>
                <ISBN>0-06-064831-7</ISBN>                <Publisher>Harper
&amp; Row</Publisher>
        </Book>
</BookStore>
```

Costello 2002 > XML Schema Tutorial

The <unique> element is used exactly like the <key> element is used. It has a <selector> and one or more <field> elements, just like <key> has. The only difference is that the schema validator will simply validate that, whenever present, the values are unique.

Costello 2002 > XML Schema Tutorial

Keyref which asserts: "the value of this element must match the value of an element referred to by this".

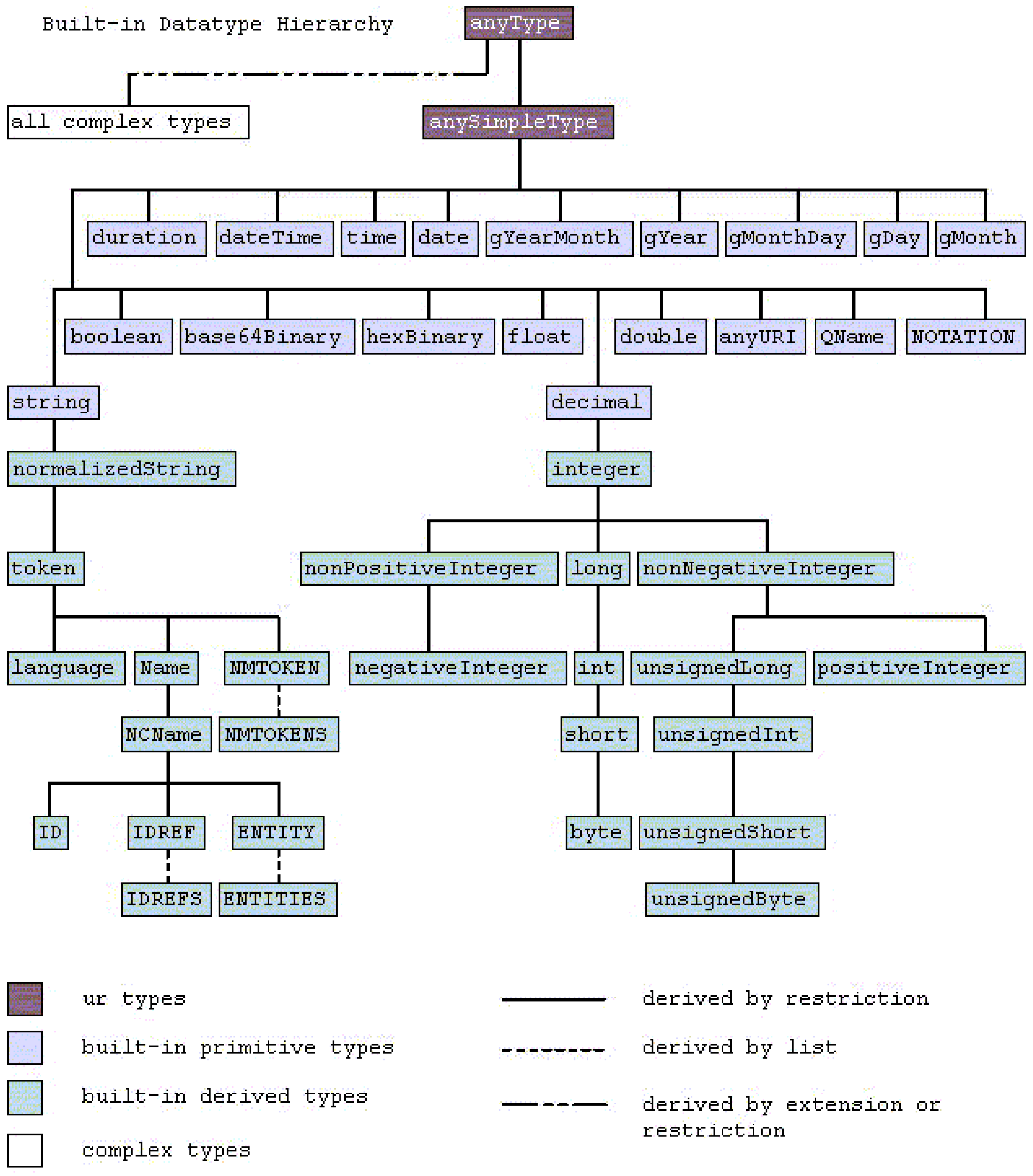
```
<?xml version="1.0"?>
<Library xmlns="http://www.library.org"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation=
                              "http://www.library.org
                               AuthorSigningAtLibrary.xsd">
    <Books>
        <Book>
                <Title>Illusions The Adventures of a Reluctant
Messiah</Title>
                <Author>Richard Bach</Author>
                <Date>1977</Date>
                <ISBN>0-440-34319-4</ISBN>
                <Publisher>Dell Publishing Co.</Publisher>
        </Book>
       ...
    </Books>
    <GuestAuthors>
```

```
        <Author>
            <Name>Richard Bach</Name>
            <BookForSigning>
                <Title>Jonathon Livingston Seagull</Title>

<!-- Ensure that the ISBN for the GuestAuthor matches one of the ISBNs in
the BookStore. -->

                <ISBN>0-440-34319-4</ISBN>
            </BookForSigning>
        </Author>
    </GuestAuthors>
</Library>
```

```

<xsd:element name="Library">
        <xsd:complexType>
             <xsd:sequence>
                 <xsd:element name="Books">
                     <xsd:complexType>
                         <xsd:sequence>
                             <xsd:element ref="Book"
maxOccurs="unbounded"/>
                         </xsd:sequence>
                     </xsd:complexType>
                 </xsd:element>
                 <xsd:element ref="GuestAuthors"/>
             </xsd:sequence>
        </xsd:complexType>
        <xsd:key name="PK">
            <xsd:selector xpath="bk:Books/bk:Book"/>
            <xsd:field xpath="bk:ISBN"/>
        </xsd:key>
        <xsd:keyref name="isbnRef" refer="PK">
            <xsd:selector
xpath="bk:GuestAuthors/bk:Author/bk:BookForSigning"/>
            <xsd:field xpath="bk:ISBN"/>
        </xsd:keyref>
    </xsd:element>
```

Schema Datatypes

Datatype Hierarchy



W3C 2001 > Recommendation XML Schema Part 1: Structures

DataType Inventory

Built-in Primitive
Built-inDerived

This Datatype inventory is derived from *W3C 2001 > Recommendation XML Schema Part 1: Structures* and *Dan Vit > XSD Quick Reference*

Note that many of the examples bellow use list definitions, as in:

```
<xsd:element name="myLongList">
 <xsd:simpleType>
   <xsd:list itemType="xsd:long" />
 </xsd:simpleType>
</xsd:element>
```

Logic Type

Name	**boolean**
Description	Binary-valued logic legal literals.
Range	true, false, 1, 0
Example	`<myBoolean>true</myBoolean>`

Binary Data Types

Name	**base64Binary**
Description	Base64-encoded arbitrary binary data.
Range	
Example	`<myBase64Binary>0F9G</myBase64Binary>`

Name	**hexBinary**
Description	Hex-encoded binary data.
Range	

Example	`<myHexBinary> FB7</myHexBinary>`
	`"0FB7" is a`
	`hex encoding for 16-bit int 4023 (binary 111110110111).`

Numeric Types - Theoretical

Name	**decimal**
Description	Arbitrary precision decimal numbers, i × 10^-n, where i and n are integers such that n >= 0. That is, the point floats by base 10 scaling. Sign omitted, "+" is assumed. Leading and trailing zeroes are optional. If the fractional part is zero, the period and following zero(es) can be omitted. Scientific notation not allowed.
Range	-infinity, 0, +infinity

Example	`<xsd:element name=" myDecimalList" maxOccurs="unbounded" >`
	`  <xsd:simpleType>`
	`    <xsd:list itemType="xsd:decimal" />`
	`  </xsd:simpleType>`
	`</xsd:element>`
	
	`<myDecimalList>-1.23 12678967.543233 +100000.00 210</myDecimalList>`

Name	**integer**
Derived From	decimal
Description	Arbitrary Integer or whole numbers .Sign omitted, "+" is assumed. Leading and trailing zeroes are optional.
Range	-infinity to 0 to + infinity

Example	`<myIntegerList>-12323 +324 0 13432566 458</myIntegerList>`

Name	**nonNegativeInteger**
Derived From	decimal > integer
Description	Arbitrary whole numbers from zero, inclusive, to infinity. Leading and trailing zeroes are optional. Sign omitted, "+" is assumed.
Range	0,1,2, ... + infinity
Example	`<myNonNegativeIntegerList>0 2 02 010 234324 2343555 999934399 </myNonNegativeIntegerList>`

Name	**nonPositiveInteger**
Derived From	decimal > integer
Description	Arbitrary whole numbers from zero, inclusive, to minus infinity. Leading and trailing zeroes are optional.
Range	-inifinity, ...-2,-1,0
Example	`<myNonPositiveIntegerList>-12 -23323 0 -10 -05 </myNonPositiveIntegerList>`

Name	**positiveInteger**
Derived From	decimal > integer > nonNegativeInteger
Description	Arbitrary whole numbers from one, inclusive, to infinity. Leading and trailing zeroes are optional.
Range	1,2,3, ... +inifinity
Example	`<myPositiveIntegerList>1 23 +3435 13439588 </myPositiveIntegerList>`

Name	**negativeInteger**
Derived From	decimal > integer > nonPositiveInteger
Description	Arbitrary whole numbers from minus one, inclusive, to minus infinity. Leading and trailing zeroes are optional.
Range	-infinity, ... -2.-1
Example	`<myNegativeIntegerList>-098568 -0000234 -234 -23 -2 -1 </myNegativeIntegerList>`

Numeric Types - Machine Constrained

Nonintegrals

Name	**float**
Description	IEEE single-precision 32-bit floating point type. Can use scientific notation. Includes extra literals. The point floats by base 2 scaling.
Range	{-INF,0,-0,+INF, `NaN` } plus { -3.4028235E+38 through -1.401298E-45 for negative values; 1.401298E-45 through 3.4028235E+38 for positive values.}
Example	`<myFloatList>-3.4028235E+38 -123.23435 432.345e20 -00034.234 +0098.344 -INF NaN -0 +0 0</myFloatList>`

Name	**double**
Description	IEEE double-precision64-bit floating point type. Can use scientific

	notation. Includes extra literals. The point floats by base 2 scaling.
Range	{-INF,0,-0,+INF, `NaN` } plus -1.79769313486231570E+308 through -4.94065645841246544E-324 for negative values; 4.94065645841246544E-324 through 1.79769313486231570E+308 for positive values.
Example	`<myDoubleList>-1.79769313486231570E+308 -123.23435 432.345e20  234E-10 -00034.234 +0098.344 -INF NaN -0 +0 0</myDoubleList>`

Integrals - Signed

Name	**long**
Derived From	decimal > integer
Description	Whole numbers representable with 8 bytes, signed. If sign omitted "+" assumed. Leading zeros optional.
Range	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Example	`<myLongList>-9223372036854775808 0 13 +123 +000123 -0034 </myLongList>`

Name	**int**
Derived From	decimal > integer > long
Description	Whole numbers representable with 4 bytes, signed. If sign omitted "+" assumed. Leading zeros optional.
Range	-2,147,483,648 to 2,147,483,647
Example	`<myIntList> -2147483648 2147483647 0 +00234 </myIntList>`

Name	**short**
Derived From	decimal > integer > long > int
Description	Whole numbers representable with 2 bytes, signed. If sign omitted "+" assumed. Leading zeros optional.
Range	-32,768 to 32,767
Example	`<myShortList> -32768 0 32767 -00234 -234 09898 +23</myShortList>`

Name	**byte**
Derived From	decimal > integer > long > int > short
Description	Whole numbers representable with 1 byte, signed. If sign omitted "+" assumed. Leading zeros optional.
Range	-128 to 127
Example	`<myByteList> -128 0 +127 -23 -000045 89</myByteList>`

Integrals - unsigned

Name	**unsignedLong**
Derived From	decimal > integer > nonNegativeInteger
Description	Whole numbers representable with 8 bytes, unsigned. Leading zeros optional. No sign permitted (not even "+").
Range	0 to 18446744073709551615
Example	`<myUnsignedLongList>18446744073709551615 0 234 00234</myUnsignedLongList>`

Name	**unsignedInt**
Derived From	decimal > integer > nonNegativeInteger > unsignedLong
Description	Whole numbers representable with 4 bytes, unsigned. Leading zeros optional. No sign permitted (not even "+").
Range	0 to 4294967295

Example	`<myUnsignedIntList>0 3254 003243 4294967295`
	`</myUnsignedIntList>`

Name	**unsignedShort**
Derived From	decimal > integer > nonNegativeInteger > unsignedLong > unsignedShort
Description	Whole numbers representable with 2 bytes, unsigned. Leading zeros optional. No sign permitted (not even "+").
Range	0 to 65535

Example	`<myUnsignedShortList>0 342 0000032546`
	`65535</myUnsignedShortList>`

Name	**unsignedByte**
Derived From	decimal > integer > nonNegativeInteger > unsignedLong > unsignedShort > unsignedByte
Description	Whole numbers representable with 1 byte, unsigned. Leading zeros optional. No sign permitted (not even "+").
Range	0 to 255
Example	`<myUnsignedByteList>0 234 0044 255 </myUnsignedByteList>`

Text Types

Name	**string**
Description	A sequence of Unicode characters.

Example	`<myStringType>The fight        against cliché can`
	`be كجدؤعض and`
	`take €100 </myStringType> [Has carriage returns]`

Name	**anyURI**
Description	A Uniform Resource Identifier Reference (URI).

Example	`<myURI>http://www.nowhere.com</myURI>`
	`<myURI>urn:john-bentley-schema:nevertochange</myURI>`
	`<myURI>www.nowhere.com</myURI>`
	`<myURI>dfgd</myURI>`

Name	**normalizedString**
Derived From	string
Description	A string that may contain spaces but not tabs, linefeeds or carriage returns.

Example	`<myNormalizedString>cliché can be كجدؤعض and    take €100`
	`</myNormalizedString> [Second line due to wrap around in`
	`word]`

Name	**token**

Derived From	string > normalizedString
Description	Set of strings that do not contain the line feed nor tab characters, that have no leading or trailing spaces and that have no internal sequences of two or more spaces.
Example	`<myToken>Cliché can be كجدؤعض and take €100</myToken>`

Name	**language**
Derived From	string > normalizedString > token
Description	Represents natural language identifiers as defined by [RFC 1766], "language identification".

Example	`<myLanguageType>en-NZ</myLanguageType>`
	`<myLanguageList>de-CH de-AT de-LU de-LI el-GR en en-US en-GB en-AU en-CA en-NZ</myLanguageList>`

Text Types - Defined in XML 1.0

Name	**QName**
Description	XML Qualified Names. QName ::= (NamespacePrefix ':')? LocalPart NamespacePrefix ::= NCName LocalPart ::= NCName

Example	`<root   xmlns:dummyns="http://tempuri.org/dummy">`
	`...`
	`   <myQNameType>dummyns:sdf</myQNameType>`
	`</root>`

Name	**Name**
Derived From	string > normalizedString > token
Description	A Name is a token beginning with a letter or one of a few punctuation characters, and continuing with letters, digits, hyphens, underscores, colons, or full stops. No white space allowed.
Example	`<myNameType>nowThis:Is64_.Cool</myNameType>`

Name	**NCName**
Derived From	string > normalizedString > token > Name
Description	A non colonized XML Name. No white space allowed. No colon allowed.
Example	`<myNCNameType>nowThisIsStill64_.Cool</myNCNameType>`

Name	**NMTOKEN**
Derived From	string > normalizedString > token
Description	An Nmtoken (name token) is any mixture of name characters. No white space allowed. Should only be used on attributes.
Example	`<myNMTOKENType myNMTOKENValue="ThisIsMost_.:Cool" />`

Name	**NMTOKENS**
Derived From	string > normalizedString > token > NMTOKEN
Description	A List of NMTOKEN types. Should only be used for attributes.
Example	`<myNMTOKENSType myNMTOKENSValue="ThisIsMost_.:Cool SoisThis And.this Will:YOu:..Like" />`

ID, IDREF, IDREFS omitted.

ENTITY and ENTITIES omitted.

NOTATION

Date and Time Types

Name	**dateTime**
Description	An instance of nonrecurring time in form CCYY-MM-DDThh:mm:ss. Optional Sign. 24 hour time form only. Optional fractional seconds. Optional trailing "Z" for Zulu (Universal Time Coordinated) or TimeZone offset, eg -05:30. Date and time parts mandatory.
Example	`<myDateTime>2003-10-09T15:30:23</myDateTime>` `<myDateTime>2003-10-09T15:30:23Z</myDateTime>` `<myDateTime>2003-10-09T15:30:23.3245+10:00</myDateTime>` `<myDateTime>2003-10-09T15:30:23.3245Z</myDateTime>`

Name	**date**
Description	A non recurring Gregorian calendar date in the form CCYY-MM-DD. Optional Sign. Optional trailing "Z" for Zulu (Universal Time Coordinated) or TimeZone offset, eg -05:30.
Example	`<myDate>2003-10-09</myDate>` `<myDate>2003-10-09Z</myDate>` `<myDate>2003-10-09+10:30</myDate>`

Name	**dateTime**
Description	An instant of time that recurs every day in form hh:mm:ss. 24 hour time form only. Optional Sign. Optional fractional seconds. Optional trailing "Z" for Zulu (Universal Time Coordinated) or TimeZone offset, eg -05:30.
Example	`<myTime>15:30:23</myTime>` `<myTime>15:30:23Z</myTime>` `<myTime>15:30:23.3245+10:00</myTime>` `<myTime>15:30:23.3245Z</myTime>`

Name	**duration**
Description	A duration of time. In the form PnYn MnDTnH nMnS. Optional leading sign. Seconds may have a fractional part. If zero you can omit a term. "T" is the Time designator and may be omitted if no time part.
Example	`<myDuration>P2Y3M17D</myDuration>` `<myDuration>P217DT3.3S</myDuration>` `<myDuration>P0217DT003.3S</myDuration>` `<myDuration>-P217DT3.3S</myDuration>`

Name	**gYear**
Description	An unrecurring Gregorian year. Has form CCYY. Optional sign. Optional Time Zone offset or "Z" for Zulu time (UTC)
Example	`<myGYear>2003</myGYear>` `<myGYear>0023</myGYear>` `<myGYear>0023+10:00</myGYear>` `<myGYear>2003Z</myGYear>`

	`<!-- Not Allowed in Visual Studio .Net 2002,_but OK by W3C`
	`<myGYear>-1239</myGYear>`
	`<myGYear>13094</myGYear> -->`

Name	**gYearMonth**
Description	A Gregorian year. Has form CCYY-MM. Optional sign. Optional Time Zone offset or "Z" for Zulu time (UTC)

Example	`<myGYearMonth>2003-12</myGYearMonth>`
	`<myGYearMonth>2003-12Z</myGYearMonth>`
	`<myGYearMonth>2003-12-05:30</myGYearMonth>`

Name	**gMonth**
Description	A gregorian month that recurs every year. Has form `--MM--`. Optional Time Zone offset or "Z" for Zulu time (UTC). No sign.

Example	`<myGMonth>--02--</myGMonth>`
	`<myGMonth>--02--Z</myGMonth>`
	`<myGMonth>--02--+10:30</myGMonth>`
	`<myGMonth>--02---05:30</myGMonth>`

Name	**gMonthDay**
Description	A gregorian date that recurs, specifically a day of the year such as the third of May. Has form `--MM-DD`. Optional Time Zone offset or "Z" for Zulu time (UTC). No sign.

Example	`<myGMonthDay>--04-03</myGMonthDay>`
	`<myGMonthDay>--04-03Z</myGMonthDay>`
	`<myGMonthDay>--12-20-05:30</myGMonthDay>`

Name	**gDay**
Description	A Gregorian day that recurs, specifically a day of the month such as the 5th of the month. Has form `---DD`. Optional Time Zone offset or "Z" for Zulu time (UTC). No sign.

Example	`<myGDay>---05</myGDay>`
	`<myGDay>---05Z</myGDay>`
	`<myGDay>---05-05:30</myGDay>`

Simple Typing

Built In

To constrain the text that's used within an element or attribute, simply choose the data type with the appropriate value/lexical space and use it in the declaration's type attribute, as shown in the following code:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://example.org/employee/"
  targetNamespace="http://example.org/employee/">
  <xsd:complexType name="EmployeeType">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="hiredate" type="xsd:date"/>
      <xsd:element name="salary" type="xsd:double"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string"/>
  </xsd:complexType>
  <xsd:element name="employee" type="tns:EmployeeType"/>
</xsd:schema>
```

Custom Simple Datatypes

Use the simpleType element to build a custom Simple datatype. This can be named or anonymous.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://example.org/employee/"
  targetNamespace="http://example.org/employee/">
  <xsd:simpleType name="socialSecurityNumber">
     <!-- define characteristics of new simple type here -->
  </xsd:simpleType>
  <xsd:attribute name="id" type="tns:socialSecurityNumber"/>
</xsd:schema>
```

You can base new simple types on existing types using three techniques: restricting a base type, creating a list of a given type, or defining a union of types. There is an element that represents each of these different derivation techniques, which can be nested directly within the xsd:simpleType element. These elements are xsd:restriction, xsd:list, and xsd:union

Derivation Element	Description
xsd:restriction	The new type created will be a restriction of the existing type, which means it will have a narrower set of legal values.
xsd:list	The new type created will be a whitespace-delimited list of another simple type.
xsd:union	The new type created will be a union of two or more other simple types.

Skonnard 2002 > The XML Files A Quick Guide to XML Schema-Part 2

Simple Type Restriction

Use facets to further restrict a base type.

```
<xsd:simpleType name="age">
    <xsd:restriction base="xsd:unsignedShort">
      <!-- restrict string's value space here -->
    </xsd:restriction>
</xsd:simpleType>
```

Simple Type List

The following schema fragment defines a new simple type that is a list of recordAge values:

```
<xsd:simpleType name="listOfAges">
  <xsd:list itemType="tns:recordAge" />
</xsd:simpleType>

<xsd:element name="ages" type="tns:listOfAges"/>
```

A valid instance of this new type must contain a whitespace-delimited list of valid recordAge values, as shown here:

```
<tns:ages xmlns:tns="...">112 122 119</ages>
```

Skonnard 2002 > The XML Files A Quick Guide to XML Schema-Part 2

Simple Type Union

```
<xsd:simpleType name="extremeAge">
    <xsd:union memberTypes="tns:infantAge tns:recordAge" />
</xsd:simpleType>

<xsd:element name="extremeAgeRanges" type="tns:extremeAge"/>
```

This means that an instance of extremeAge must contain a value from within either the infantAge or recordAge value spaces. For example, the following extremeAgeRanges element is valid because it contains the value of 2, which is within the infantAge value space defined as up to three years old:

```
<tns:extremeAgeRanges xmlns:tns="...">2</extremeAgeRanges>
```

Skonnard 2002 > The XML Files A Quick Guide to XML Schema-Part 2

Facets

Facet Element	What It Specifies
xsd:enumeration	A fixed value that the type must match
xsd:fractionDigits	The maximum number of decimal digits to the right of the decimal point
xsd:length	The number of characters in a string-based type, the number of octets in a binary-based type, or the number of items in a list-based type
xsd:maxExclusive	The exclusive upper-bound on the value space of the type
xsd:maxInclusive	The inclusive upper-bound on the value space of the type
xsd:maxLength	The maximum number of characters in a string-based type, the maximum number of octets in a binary-based type, or the maximum number of items in a list-based type
xsd:minExclusive	The exclusive lower-bound on the value space of the type
xsd:minInclusive	The inclusive lower-bound on the value space of the type
xsd:minLength	The minimum number of characters in a string-based type, the minimum number of octets in a binary-based type, or the minimum number of items in a list-based type
xsd:pattern	A pattern, based on a regular expression, that the type must match
xsd:totalDigits	The maximum number of decimal digits for types derived from number
xsd:whiteSpace	Rules for whitespace normalization

Patterns, enumerations => "or" them together: All other facets => "and" them together

```
<!-- Or together -->
<xsd:simpleType name="TelephoneNumber">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d{2}-\d{5}"/>
        <xsd:pattern value="\d{3}-\d{4}"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="shape">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="circle"/>
      <xsd:enumeration value="triangle"/>
      <xsd:enumeration value="square"/>
    </xsd:restriction>
</xsd:simpleType>
```

```
<!-- And together -->
```

Type checking in an Instance document without Validating the entire doc

```
<age xmlns:tns="http://example.org/ages/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="tns:infantAge">2
</age>
```

Then use a validator to check instance document against type definitions (in a XSD)

Open Content (Wildcards)

Your schema should have open content. That is, a liberal use of the "any" anyAttribute".

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema id="OpenContent"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://tempuri.org/OpenContent.xsd"
    xmlns:tns="http://tempuri.org/OpenContent.xsd"
    elementFormDefault="qualified"
    >
  <xsd:element name="Book">
    <xsd:complexType>
      <xsd:sequence>

        <xsd:element name="Title">
          <xsd:complexType>
            <xsd:simpleContent>
              <xsd:extension base="xsd:string">
                <xsd:anyAttribute/>
              </xsd:extension>
            </xsd:simpleContent>
          </xsd:complexType>
        </xsd:element>

        <xsd:any minOccurs="0" maxOccurs="unbounded"/>

        <xsd:element name="author" >
          <xsd:complexType>
            <xsd:simpleContent>
              <xsd:extension base="xsd:string">
                <xsd:anyAttribute/>
              </xsd:extension>
            </xsd:simpleContent>
          </xsd:complexType>
        </xsd:element>

        <xsd:any minOccurs="0" maxOccurs="unbounded"/>

        </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

An open content schema is one that allows instance documents to contain additional elements beyond what is declared in the schema.

This is a good idea as it allows the schema to evolve.

Costello 2002 > XML Schema Tutorial [Derived]

You can use the namespace attribute of "any" or "anyAttribute" for finer control.

Namespace Attribute in Any	
Value of Namespace Attribute	Allowable Element Content
##any	Any well-formed XML from any namespace (default)
##local	Any well-formed XML that is not qualified, i.e. not declared to be in a namespace
##other	Any well-formed XML that is not from the target namespace of the type being defined
"http://www.w3.org/1999/xhtml ##targetNamespace"	Any well-formed XML belonging to any namespace in the (whitespace separated) list; ##targetNamespace is shorthand for the target namespace of the type being defined

This examples allows you to have XHTML in an XML instance document:

```
<element name="htmlExample">
 <complexType>
  <sequence>
   <any namespace="http://www.w3.org/1999/xhtml"
        minOccurs="1" maxOccurs="unbounded"
        processContents="skip"/>
  </sequence>
  <anyAttribute namespace="http://www.w3.org/1999/xhtml"/>
 </complexType>
</element>
```

W3C 2001 > Recommendation XML Schema Part 0: Primer

XSD Regular Expressions

Basic

Use | for Or.

Use (and) to group expressions.

Repetition Operators

* 0 or more,
? 0 or 1,

+ 1 or more

Interval Operators

{x,y} range x to y,

{x,} at least x,

{x} exactly x, i.e. {4,8} 4 to 8

{0,x} at most x

{0,0} empty string

Character Set Expressions

Character Set by Range Expressions

[a-zA-Z] = character a to z upper and lower case

[0-9] = digits 0 to 9

Character Set negations in Range Expressions

Use ^

Character Set by Special Character Sequences

\n	newline
\r	return
\t	tab
. (dot)	all characters except newline and return
\s	space characters (space, tab, newline, return)
\S	non-Space characters
\i	initial XML name characters (letter _ ;)
\I	not initial XML name characters
\c	XML NameChar characters
\C	not XML NameChar characters
\d	decimal digits
\D	not decimal digits
\w	XML Letter or Digit characters
\W	not XML Letter or Digit characters

Character Set by \p{ Special Character Sequences }

Category	Property	Meaning
Letters	L	All Letters
	Lu	uppercase
	Ll	lowercase
	Lt	titlecase
	Lm	modifier
	Lo	other
Marks	M	All Marks
	Mn	nonspacing
	Mc	spacing combining
	Me	enclosing
Numbers	N	All Numbers

	Nd	decimal digit
	Nl	letter
	No	other
Punctuation	P	All Punctuation
	Pc	connector
	Pd	dash
	Ps	open
	Pe	close
	Pi	initial quote (may behave like Ps or Pe depending on usage)
	Pf	final quote (may behave like Ps or Pe depending on usage)
	Po	other
Separators	Z	All Separators
	Zs	space
	Zl	line
	Zp	paragraph
Symbols	S	All Symbols
	Sm	math
	Sc	currency
	Sk	modifier
	So	other
Other	C	All Others
	Cc	control
	Cf	format
	Co	private use
	Cn	not assigned

Character Set by \p{IsCategory}

\p{IsBasicLatin} block escape identifying ASCII characters.
Similar IsGreek, IsHebrew, IsThai for these ranges of Unicode blocks.
\P{} not the block or category, eg, \P{IsGreek} = not Greek block

Special Characters needing to be escaped with a '\'

\ | . - ^ ? * + { } ( ) [ ]

Examples

Expression	Match(s)
Chapter \d	Chapter 0, Chapter 1, Chapter 2 ....
Chapter\s\d	Chapter followed by a single whitespace character (space, tab, newline, etc.), followed by a single digit
Chapter\s\w	Chapter followed by a single whitespace character (space, tab, newline,

	etc.), followed by a word character (XML 1.0 Letter or Digit)
Espan&#xF1;ola	Española
\p{Lu}	any uppercase character, the value of \p{} (e.g. "Lu") is defined by Unicode
\p{IsGreek}	any Greek character, the 'Is' construction may be applied to any block name (e.g. "Greek") as defined by Unicode
\P{IsGreek}	any non-Greek character, the 'Is' construction may be applied to any block name (e.g. "Greek") as defined by Unicode
a*x	x, ax, aax, aaax ....
a?x	ax, x
a+x	ax, aax, aaax ....
(a|b)+x	ax, bx, aax, abx, bax, bbx, aaax, aabx, abax, abbx, baax, babx, bbax, bbbx, aaaax ....
[abcde]x	ax, bx, cx, dx, ex
[a-e]x	ax, bx, cx, dx, ex
[-ae]x	-x, ax, ex
[ae-]x	ax, ex, -x
[^0-9]x	any non-digit character followed by the character x
\Dx	any non-digit character followed by the character x
.x	any character followed by the character x
.*abc.*	1x2abc, abc1x2, z3456abchooray ....
ab{2}x	abbx
ab{2,4}x	abbx, abbbx, abbbbx
ab{2,}x	abbx, abbbx, abbbbx ....
(ab){2}x	ababx

Dan Vit > XSD Quick Reference

W3C 2001 > Recommendation XML Schema Part 2: Datatypes

Miscellaneous

Questions

Visual Studio .NET 2002/ msxml parser 3 or 4 ? bugs

- Doesn't work to well with substitution groups. When you validate an instance document it returns "No validation errors where found" but lists those in the substitution as unsupported by the active schema.
- Doesn't work with the any element wildcard.
- Sometimes the XML instance document will not recognize the XSD Schema. It reports

  "Visual Studio could not locate a schema for this document. Validation can only ensure this is a well formed XML document and cannot validate the data against a schema. "

  Work around:

In the XML instance document we need to have a default namespace that corresponds to a schemas target namespace.

- `<xsd:group ref="myGroup" />` has to be used instead of `<xsd:element ref="myGroup" />`

Sources

Skonnard 2002 > The XML Files A Quick Guide to XML Schema
Microsoft/DevelopMentor Aaron Skonnard
MSDN Magazine The XML Files A Quick Guide to XML Schema Last Dated: April 2002
ms-help://MS.MSDNQTR.2003FEB.1033/dnmag02/html/xml0204.htm

Skonnard 2002 > The XML Files A Quick Guide to XML Schema-Part 2
Microsoft/DevelopMentor Aaron Skonnard
MSDN Magazine The XML Files A Quick Guide to XML Schema-Part 2 Last Dated: July 2002
ms-help://MS.MSDNQTR.2003FEB.1033/dnmag02/html/xml0207.htm

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XML Schemas
Microsoft
Microsoft XML Core Services (MSXML) 4.0 - XML Schemas Last Dated: 2003
ms-help://MS.MSDNQTR.2003FEB.1033/xmlsdk/htm/xmlschemas_overview_2u9f.htm

W3C 2001 > Recommendation XML Schema Part 0: Primer
World Wide Web Consortium (W3C)
Recommendation XML Schema Part 0: Primer Last Dated: 2 May 2001
http://www.w3.org/TR/xmlschema-0/

W3C 2001 > Recommendation XML Schema Part 1: Structures
World Wide Web Consortium (W3C)
Recommendation XML Schema Part 1: Structures Last Dated: 2 May 2001
http://www.w3.org/TR/xmlschema-1/

W3C 2001 > Recommendation XML Schema Part 2: Datatypes
World Wide Web Consortium (W3C)
Recommendation XML Schema Part 2: Datatypes Last Dated: 2 May 2001

http://www.w3.org/TR/xmlschema-2/

Costello 2002 > XML Schema Tutorial
Roger Costello
XML Schemas Tutorial Last Dated: 2 Jun 2002

Dan Vit > XSD Quick Reference
Dan Vit
XSD Quick Reference Last Dated: 2003
xmlhelp@dvint.com
http://www.xml.dvint.com

Document Licence