XML Schema Infoset Model, Part 2

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

Section 1. Before you start

About this tutorial

The second of a two-part series, this tutorial gives you the tools to create all of the different constructs within the XML Schema Infoset Model. In this tutorial, you will learn:

- How to create and work with Schema as a whole
- How to create and work with simple types
- How to create and work with complex types
- How to create and work with attribute group definitions
- How to create and work with model group definitions
- How to create and work with model groups
- How to create and work with attribute declarations

You should be familiar with the content in *Part 1* before you begin this tutorial. Part 1 takes you through the steps in setting up the development environment, creating and loading XML Schema models, working with namespaces, and reusing definitions across XML Schema models.

This tutorial series is for developers who are familiar with Java, XML, and XML Schema, and who are interested in combining these technologies using the XML Schema Infoset Model. You should therefore understand how to write Java code and understand how XML Schemas work. Some understanding of the Unified Modeling Language (UML) is helpful but not required. You can get an introduction to XML Schema fundamentals in the Resources on page 36 at the end of this tutorial, and an introduction to UML basics in *How to read UML diagrams* in *Part 1* of this tutorial series.

Setting up the development environment

To get the most out of this tutorial, download and install the Eclipse Modeling Framework and the XML Schema components as described in *Setting up the development environment* in *Part 1* of this tutorial series.

About the author

Dave Spriet is a software developer for the IBM Toronto Laboratory. He develops tools for the WebSphere Business Integration Message Broker (WBIMB) product. He specializes in object-oriented technologies, XML, XML Schema, middleware, and UML Modeling. He is also a committer for the *XML Schema Infoset Model* component. Dave has a Bachelor of Science degree (Honors) in Computer Science and Statistics from McMaster University. He welcomes any comments; please direct them to *spriet@ca.ibm.com*.

Section 2. Overview of XML Schema documents

Schema as a whole

The <schema> element is the root element of every XML Schema document:

```
<?xml version="1.0"?>
<xsd:schema>
...
...
</xsd:schema>
```

The <schema> element may contain some attributes, which are described in *XML Schema Part 1: Structures* of the standard. A typical schema declaration often looks something like this:

```
<?xml version="1.0"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.eclipse.org/xsd/examples"
xmlns="http://www.eclipse.org/xsd/examples"
elementFormDefault="qualified">
...
...
</xs:schema>
```

- **xmlns:xsd="http://www.w3.org/2001/XMLSchema"** - Indicates that the elements and data types used in the schema (schema, element, complexType, sequence, string, int, etc.) come from the "http://www.w3.org/2001/XMLSchema" namespace, which is also known as the *schema for schemas*. It also specifies that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with xsd in the example above.
- **xmlns="http://www.eclipse.org/xsd/examples"** - Indicates that the default namespace is "http://www.eclipse.org/xsd/examples".
- **elementFormDefault="qualified"** - Indicates that any elements used by the XML instance document that were declared in this schema must be namespace qualified.

The XML Schema model java interface for the root <schema> is org.eclipse.xsd.XSDSchema and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDSchema().

Exercise 1: Create a root schema document

The first exercise consists of creating a root schema with target namespace "http://www.eclipse.org/xsd/examples/po". With the help of *Part 1*, create a schema document using the XML Schema Infoset Model APIs that represents the following:

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.eclipse.org/xsd/examples/po"
```

```
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:po="http://www.eclipse.org/xsd/examples/po"/>
```

Hint: You will need to use org.eclipse.xsd.XSDFactory.eINSTANCE to create the schema constructs.

New constructs created in this exercise:
- <schema> - root schema with targetNamespace="http://www.eclipse.org/xsd/examples/po"

Solution: Exercise 1 solution on page 4

Exercise 1 solution

```
//Get the URI of the model file. URI fileURI
= URI.createPlatformResourceURI(xsdFile.getFullPath().toString());

//Create a resource set to manage the different resources
ResourceSet resourceSet = new ResourceSetImpl();

//Create a resource for this file.
Resource resource = resourceSet.createResource(fileURI);

//Create the root XSDSchema object
XSDSchema schema = XSDFactory.eINSTANCE.createXSDSchema();

//Set the target namespace of the given schema document to
schema.setTargetNamespace("http://www.eclipse.org/xsd/examples/po");

java.util.Map qNamePrefixToNamespaceMap = schema.getQNamePrefixToNamespaceMap();
qNamePrefixToNamespaceMap.put(schema.getSchemaForSchemaQNamePrefix(),
        XSDConstants.SCHEMA_FOR_SCHEMA_URI_2001);

//put the following namespace in the root schema namespace map
qNamePrefixToNamespaceMap.put("po", schema.getTargetNamespace());

//We call updateElement to synchronize the MOF model with the underlying DOM model
//This should only have to be done after creating a new model
schema.updateElement();

//Add the root schema to the resource that was created above
resource.getContents().add(schema);

// Save the contents of the resource to the file system.
resource.save(Collections.EMPTY_MAP);
```

Global XML Schema constructs

XML Schema documents can contain global constructs that can be reused within the same schema as the defined construct or inside other XML Schema documents by using xsd:include and xsd:import. All global constructs have a target namespace and a name.

Here is a list of common global constructs that can be added to the root schema element:
- Global complex type definitions
- Global simple type definitions

- Global attribute group definitions
- Global model group definitions
- Global element declarations
- Global attribute declarations

There will be many opportunities to create each of these constructs throughout this tutorial.

Local XML Schema constructs

XML Schema documents can contain local constructs that are not reusable and are scoped within their parent global construct.

Here are common local constructs that can be added within a schema:

- Local complex type definitions
- Local simple type definitions
- Local groups
- Local element declarations
- Local attribute declarations

Section 3. Simple types

Simple type definitions

Simple type definitions define a simple datatype and specify the constraints and information about the values of attributes or text-only elements.

The <simpleType> element contains attributes, which are described by *XML Schema Part 1: Structures* of the standard.

Simple type definitions come in three varieties:
- Atomic data types on page 10
- List data types on page 12
- Union data types on page 13

The XML Schema model java interface for a <simpleType> is org.eclipse.xsd.XSDSimpleTypeDefinition and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDSimpleTypeDefinition().

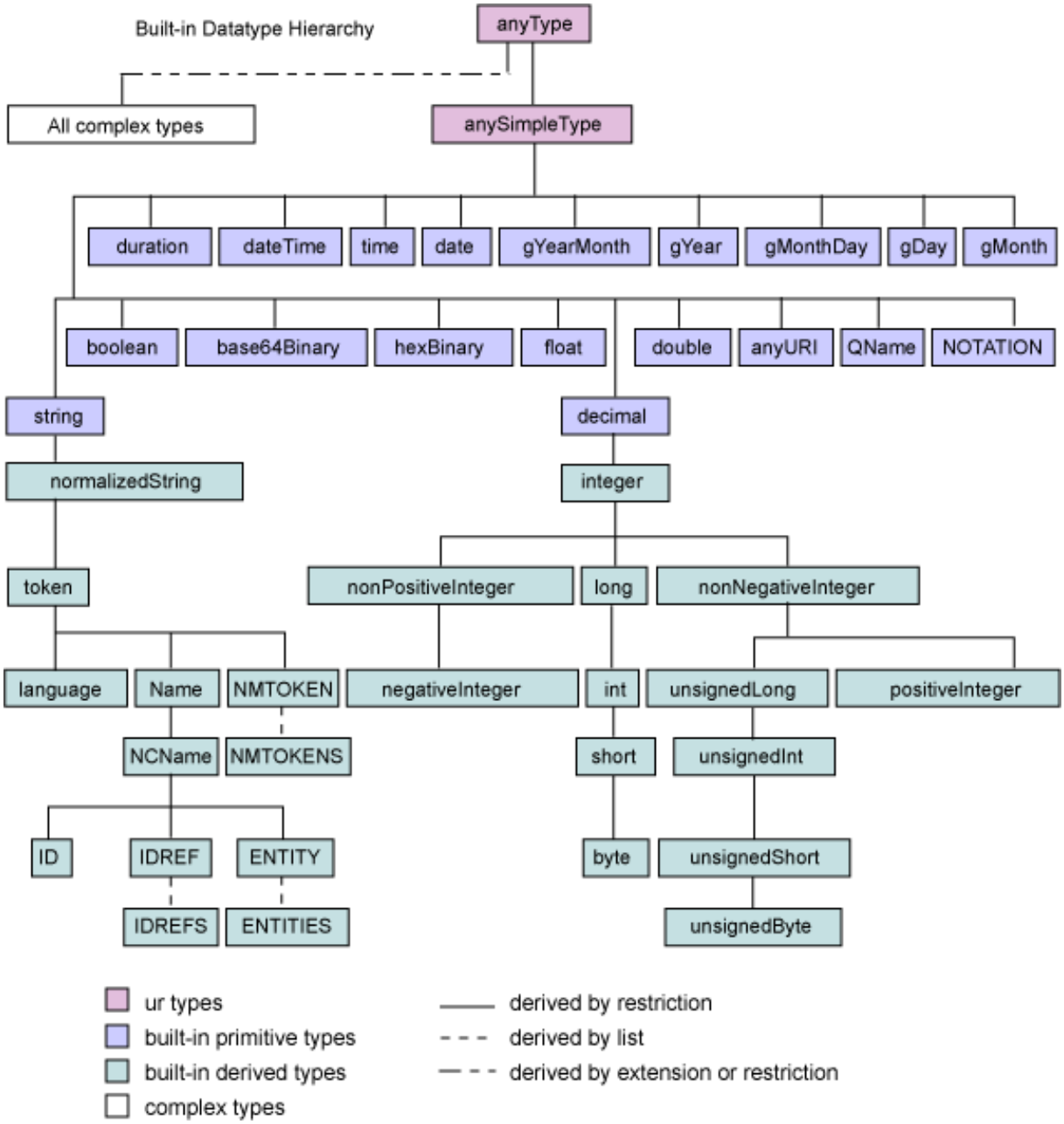
Primitive and derived data types

There are two main kinds of data types: primitive data types and derived data types.

Primitive data types

Primitive data types are those that are not defined in terms of other data types. Since primitive data types are the basis for all other types, they cannot have element content or attributes. However, they can contain values and constraints. Primitive data types are built into XML schemas. Examples of primitive data types are string, float, decimal, anyURI, and QName.

Built-in data types:



The above diagram is from *XML Schema Part 2: Datatypes specification*.

Getting access to the primitive data types (in other words, xsd:string) in the schema for schemas using the XML Schema model is done by calling the following method where rootSchema is the root schema that was created in Exercise 1: Create a root schema document on page 3 .

```
XSDSchemaImpl.getSchemaForSchema(rootSchema.getSchemaForSchemaNamespace()).
        resolveSimpleTypeDefinition(XSDConstants.SCHEMA_FOR_SCHEMA_URI_2001,"string");
```

Derived data types

Derived data types are those that are defined in terms of other data types, which are called

base types. Derived types may have attributes, and may have element or mixed content. Instances of derived types can contain any well-formed XML that is valid according to their data type definition. They may be built-in or user-derived. Base types can be primitive types or derived data types. Derived data types are created using extension and restriction facets. They can be built-in or user-derived data types.

Schema component constraints

XML Schema provides a mechanism of constraining a given simple element or simple attribute, which are known as facets or value constraints. Here is a list of the facets provided by XML Schema:

<enumeration>

Defines a list of acceptable values.

Enumeration facets contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-2/#dc-enumeration.*

The XML Schema model java interface for a <enumeration> is org.eclipse.xsd.XSDEnumerationFacet and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDEnumerationFacet().

<fractionDigits>

Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero.

FractionDigit facets contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-2/#dt-fractionDigits.*

The XML Schema model java interface for a <enumeration> is org.eclipse.xsd.XSDFractionDigitsFacet and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDFractionDigitsFacet().

<length>

Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero.

Length facets contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-2/#dt-length.*

The XML Schema model java interface for a <length> is org.eclipse.xsd.XSDLengthFacet and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDLengthFacet().

<maxExclusive>

Specifies the upper bounds for numeric values (the value must be less than this value).

MaxExclusive facets contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-2/#rf-maxExclusive.*

The XML Schema model java interface for a <maxExclusive> is

org.eclipse.xsd.XSDMaxExclusiveFacet and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDMaxExclusiveFacet().

<maxInclusive>

Specifies the upper bounds for numeric values (the value must be less than or equal to this value).

MaxInclusive facets contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-2/#rf-maxInclusive.*

The XML Schema model java interface for a <maxInclusive> is org.eclipse.xsd.XSDMaxInclusiveFacet and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDMaxInclusiveFacet().

<maxLength>

Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero.

MaxLength facets contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-2/#rf-maxLength.*

The XML Schema model java interface for a <maxLength> is org.eclipse.xsd.XSDMaxLengthFacet and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDLengthFacet().

<minExclusive>

Specifies the lower bounds for numeric values (the value must be greater than this value).

MinExclusive facets contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-2/#rf-minExclusive.*

The XML Schema model java interface for a <minExclusive> is org.eclipse.xsd.XSDMinExclusiveFacet and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDMinExclusiveFacet().

<minInclusive>

Specifies the lower bounds for numeric values (the value must be greater than or equal to this value).

MinInclusive facets contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-2/#rf-minInclusive.*

The XML Schema model java interface for a <minInclusive> is org.eclipse.xsd.XSDMinInclusiveFacet and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDMinInclusiveFacet().

<minLength>

Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero.

MinLength facets contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-2/#rf-minLength.*

The XML Schema model java interface for a <minLength> is org.eclipse.xsd.XSDMinLengthFacet and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDMinLengthFacet().

<pattern>

Defines the exact sequence of characters that are acceptable.

Pattern facets contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-2/#rf-pattern.*

The XML Schema model java interface for a <pattern> is org.eclipse.xsd.XSDPatternFacet and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDPatternFacet().

<totalDigits>

Specifies the exact number of digits allowed. Must be greater than zero.

TotalDigits facets contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-2/#rf-totalDigits.*

The XML Schema model java interface for a <totalDigits> is org.eclipse.xsd.XSDTotalDigitsFacet and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDTotalDigitsFacet().

<whiteSpace>

Specifies how white space (line feeds, tabs, spaces, and carriage returns) are handled.

WhiteSpace facets contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-2/#rf-whiteSpace.*

The XML Schema model java interface for a <whiteSpace> is org.eclipse.xsd.XSDWhiteSpaceFacet and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDWhiteSpaceFacet().

Constraints on simple type definition Schema components

The constraining facets which are allowed to be members of facets are dependent on base type definition as specified in the following table:

http://www.w3.org/TR/xmlschema-2/#cos-applicable-facets.

Atomic data types

Atomic data types have values that cannot be divided or broken down further. Atomic data types can be either primitive or derived, which is explained in Primitive and derived data types on page 6 . Numbers and strings are atomic data types because their values cannot be described using smaller parts.

Atomic data type elements may contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-1/#declare-datatype.*

Exercise 2: Create an atomic simple type with enumeration facets

Using the XML Schema Model APIs and adding on to the schema document created in Exercise 1: Create a root schema document on page 3 , create a user-defined simple type called "USState" that has a primitive base type "string" and three enumeration facets called "AK","AL", and "AR".

```
<?xml version="1.0"?>
<schema
  targetNamespace="http://www.eclipse.org/xsd/examples/po"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:po="http://www.eclipse.org/xsd/examples/po">

  <simpleType name="USState">
    <restriction base="string">
      <enumeration value="AK"/>
      <enumeration value="AL"/>
      <enumeration value="AR"/>
    </restriction>
  </simpleType>

</schema>
```

New constructs created in this exercise:

- <simpleType> - Global simple type with name="USState"
- <enumeration> - Enumeration facets ("AK","AL", and "AR")

Solution: Exercise 2 solution on page 11

Exercise 2 solution

```
//Exercise 2 builds from Exercise 1
Lab1 lab1 = new Lab1();
XSDSchema schema = lab1.createSchema(xsdFile); //Schema from lab1

//Create global simple type with name="USState"
XSDSimpleTypeDefinition simpleType = XSDFactory.eINSTANCE.createXSDSimpleTypeDefinition(
simpleType.setName("USState");

//Set the base type to be a string
XSDSimpleTypeDefinition stringType = schema.getSchemaForSchema().
        resolveSimpleTypeDefinition(XSDConstants.SCHEMA_FOR_SCHEMA_URI_2001,"string");
simpleType.setBaseTypeDefinition(stringType);

//AK Enumeration
XSDEnumerationFacet akEnum = XSDFactory.eINSTANCE.createXSDEnumerationFacet();
akEnum.setLexicalValue("AK");
simpleType.getFacetContents().add(akEnum);

//AL Enumeration
XSDEnumerationFacet alEnum = XSDFactory.eINSTANCE.createXSDEnumerationFacet();
```

```
alEnum.setLexicalValue("AL");
simpleType.getFacetContents().add(alEnum);

//AR Enumeration
XSDEnumerationFacet arEnum = XSDFactory.eINSTANCE.createXSDEnumerationFacet();
arEnum.setLexicalValue("AR");
simpleType.getFacetContents().add(arEnum);

//Add the simple type to the root schema
schema.getContents().add(simpleType);

//Save the contents of the resource to the file system.
schema.eResource().save(Collections.EMPTY_MAP);
```

List data types

List data types have values that consist of a finite length sequence of values of an atomic data type. List data types are made up of a sequence atomic data types.

There are three built-in list types: NMTOKENS, IDREFS, and ENTITIES.

List data type elements may contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-1/#declare-datatype.*

Refer to the *XML Schema Part 0: Primer* for more information on the list data type.

Exercise 3: Create a list simple type

Using the XML Schema Model APIs and adding on to the schema created in Exercise 2: Create an atomic simple type with enumeration facets on page 11 , create a user-defined simple type called "USStateList" that has a item type "USState", which was created in the previous exercise.

```
<?xml version="1.0"?>
<schema
  targetNamespace="http://www.eclipse.org/xsd/examples/po"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:po="http://www.eclipse.org/xsd/examples/po">

  <simpleType name="USState">
    <restriction base="string">
      <enumeration value="AK"/>
      <enumeration value="AL"/>
      <enumeration value="AR"/>
    </restriction>
  </simpleType>

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

</schema>
```

New constructs created in this exercise:

- <simpleType> - global list simple type with name="USStateList" and itemType="po:USState"

Solution: Exercise 3 solution on page 13

Exercise 3 solution

```
//Exercise 3 builds from Exercise 2
Lab2 lab2 = new Lab2();
XSDSchema schema = lab2.createSchema(xsdFile); //Schema from lab2

//USState
XSDSimpleTypeDefinition usStateType = schema.resolveSimpleTypeDefinition("USState");

//Create the list type
XSDSimpleTypeDefinition listType = XSDFactory.eINSTANCE.createXSDSimpleTypeDefinition();

//Set the name="USStateList"
listType.setName("USStateList");

//Set the itemType="po:USState"
listType.setItemTypeDefinition(usStateType);

//Add the listType to the root schema
schema.getContents().add(listType);

// Save the contents of the resource to the file system.
schema.eResource().save(Collections.EMPTY_MAP);
```

Union data types

Union data types are those whose value spaces and lexical spaces are the union of value spaces and lexical spaces of two or more other data types. A union data type enables an element or attribute value to be one or more instances of a type drawn from the union or combination of multiple atomic and list types. Union types have a memberTypes attribute value that is a list of all the types in the union. Union types can also have pattern and enumeration facets.

Union data type elements may contain some attributes, which are described by *http://www.w3.org/TR/xmlschema-1/#declare-datatype.*

Refer to the *XML Schema Part 0: Primer* for more information on the union data type.

Section 4. Complex types

Complex type definitions

The <complexType> element defines a complex datatype. A <complexType> element is an XML element that contains other elements and/or attributes.

The <complexType> element contains attributes, which are described by *XML Schema Part 1: Structures* of the standard. Complex types can either be global (named) or local (un-named).

The XML Schema model java interface for a <complexType> is org.eclipse.xsd.XSDComplexTypeDefinition and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDComplexTypeDefinition().

Section 5. Content models

Content models

A content model is the description of the structure and content of an element that is used to validate an XML instance document. XML Schema content models offer more control on element occurrences than DTD content models. In addition, schema content models allow the validation of mixed content.

A content model can restrict an XML instance document to a certain set of element types and attributes, describe and constrain the relationships between these different components, and uniquely identify specific elements. Sharing these content models allow businesses to exchange structured information.

Model groups

Model Groups

XML Schema contains three model group elements, which are described in the XML Schema specification:

- <sequence>
- <choice>
- <all>

The *<sequence>* element requires the element in the group to appear in the specified sequence within the containing element.

The *<choice>* element allows one and only one of the elements contained in the group to be present within the containing element.

The *<all>* element allows the elements in the group to appear (or not appear) in any order in the containing element.

The XML Schema model java interface for a <sequence>, <choice> or <all> is org.eclipse.xsd.XSDModelGroup and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDModelGroup().

An XSDModelGroup has a compositor, which is one of the following:

- org.eclipse.xsd.XSDCompositor.SEQUENCE_LITERAL
- org.eclipse.xsd.XSDCompositor.CHOICE_LITERAL
- org.eclipse.xsd.XSDCompositor.ALL_LITERAL

Exercise 4: Create a global complex type with a sequence model group

Using the XML Schema Infoset Model APIs and adding on to the schema created in Exercise 3: Create a list simple type on page 12 , create a global complex type called

"PurchaseOrderType" with a sequence model group.

```
<?xml version="1.0"?>
<schema
  targetNamespace="http://www.eclipse.org/xsd/examples/po"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:po="http://www.eclipse.org/xsd/examples/po">

  <simpleType name="USState">
    <restriction base="string">
      <enumeration value="AK"/>
      <enumeration value="AL"/>
      <enumeration value="AR"/>
    </restriction>
  </simpleType>

  <simpleType>
    <list itemType="po:USState"/>
  </simpleType>

  <complexType name="PurchaseOrderType">
    <sequence/>
  </complexType>

</schema>
```

New constructs created in this exercise:

- <complexType> - Global complex type with name="PurchaseOrderType"
- <sequence> - Model group with compositor set to "sequence"

Solution: Exercise 4 solution on page 16

Exercise 4 solution

```
//Exercise 4 builds from Exercise 3
Lab3 lab3 = new Lab3();
XSDSchema schema = lab3.createSchema(xsdFile); //Schema from lab 3

//Create global complex type with name = "PurchaseOrderType"
XSDComplexTypeDefinition complexType = XSDFactory.eINSTANCE.createXSDComplexTypeDefiniti
complexType.setName("PurchaseOrderType");

XSDParticle particle = XSDFactory.eINSTANCE.createXSDParticle();

//Create a sequence model group
XSDModelGroup modelGroup = XSDFactory.eINSTANCE.createXSDModelGroup();
modelGroup.setCompositor(XSDCompositor.SEQUENCE_LITERAL);

//Add the model group to the particle
particle.setContent(modelGroup);

//Set the contents of the complex type to be the particle
complexType.setContent(particle);

//Add the complex type to the root schema
schema.getContents().add(complexType);
```

```
// Save the contents of the resource to the file system.
schema.eResource().save(Collections.EMPTY_MAP);
```

Model group definitions

The <group> element is used both for the definition of a group and for any reference to a named group. You can use a model group to define a set of elements that can be repeated through the XML instance document. This is useful for building a complex type definition and other model groups. Named model groups contain a composition, which can be one of the following <sequence>, <choice>, or <all> child elements. A named model group consists of element declarations, wildcards, and other model groups. Named model groups must have a name attribute and are declared at the top level of the schema.

Referencing a global group definition

You can use the ref attribute to reference a <group> by target namespace prefix and name in order to use the group description. This is done by calling the following method on the XML Schema model:
org.eclipse.xsd.XSDModelGroupDefinition.setResolvedModelGroupDefinition(XSDModelGroupDefinition).

The <group> element contains attributes, which are described by *XML Schema Part 1: Structures* of the standard.

The XML Schema model java interface for a <group> is org.eclipse.xsd.XSDModelGroupDefinition and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDModelGroupDefinition().

Attribute group definitions

The <attributeGroup> element is used to group a set of <attribute>'s together.

```
<xsd:attributeGroup name="myNewAttributeGroup" />
   <xsd:attribute name="attribute1" use="required" />
   <xsd:attribute name="attribute2" use="optional" />
   <xsd:attribute name="attribute3" use="required" />
</xsd:attributeGroup>
```

Referencing a global attribute group declaration

If you want to reference the group of attributes, you must add a reference to it using the ref attribute of the <attributeGroup> element. This is done by calling the following method on the XML Schema model:
org.eclipse.xsd.XSDAttributeGroupDefinition.setResolvedAttributeGroupDefinition(XSDAttributeGroupDefini

The <attributeGroup> element contains attributes, which are described by *XML Schema Part 1: Structures* of the standard.

The XML Schema model java interface for a <attributeGroup> is org.eclipse.xsd.XSDAttributeGroupDefinition and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDAttributeGroupDefinition().

Section 6. Elements and attributes

Element declarations

The <element> element is the tag name that will be used within the XML instance document.

Element declarations can be global, local, or references, and are used for:

- Local validation of element information item values using a type definition
- Specifying default or fixed values for element information items
- Establishing uniqueness and reference constraint relationships among the values of related elements and attributes
- Controlling the substitutability of elements through the mechanism of element substitution groups

The <element> element contains attributes, which are described by *XML Schema Part 1: Structures* of the standard.

The XML Schema model java interface for a <element> is org.eclipse.xsd.XSDElementDeclaration and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDElementDeclaration().

Particles

Local element declarations and element references can repeat by changing the particles minOccurs and maxOccurs attributes.

If an element is local or a reference, it can contain minOccurs and/or maxOccurs attributes, which are described by *http://www.w3.org/TR/xmlschema-1/#cParticles* .

Referencing a global element declaration

If you declare a global element declaration, you can reference this element in the remainder of the schema using the ref attribute. This is done by calling the following method on the XML Schema model: org.eclipse.xsd.XSDElementDeclaration.setResolvedElementDeclaration(XSDElementDeclaration).

Any element

The <any> element enables us to extend the XML instance document with elements not specified by the schema.

The <any> element contain attributes, which are described by *http://www.w3.org/TR/xmlschema-1/#Wildcards* .

The XML Schema model java interface for a <any> is org.eclipse.xsd.XSDWildCard and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDWildCard().

Exercise 5: Create a global, local, and element reference

Using the XML Schema Infoset Model APIs and adding on to the schema created in Exercise 4: Create a global complex type with a sequence model group on page 15 , create the

following schema document:

```
<?xml version="1.0"?>
<schema
  targetNamespace="http://www.eclipse.org/xsd/examples/po"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:po="http://www.eclipse.org/xsd/examples/po">

  <simpleType name="USState">
    <restriction base="string">
      <enumeration value="AK"/>
      <enumeration value="AL"/>
      <enumeration value="AR"/>
    </restriction>
  </simpleType>

  <simpleType>
    <list itemType="po:USState"/>
  </simpleType>

  <complexType name="PurchaseOrderType">
    <sequence>
      <element minOccurs="0" name="state" type="po:USState"/>
      <element ref="po:notes"/>
    </sequence>
  </complexType>

  <element name="notes" type="string"/>
</schema>
```

Hint: You will need to create an org.eclipse.xsd.XSDParticle for the local element and element reference.

New constructs created in this exercise:

- <element> - Global element with name="notes" and type="string"
- <element> - Local element with name="state", type="po:USState" and minOccurs="0"
- <element> - Element reference with ref="po:notes"

Solution: Exercise 5 solution on page 20

Exercise 5 solution

```
//Exercise 5 builds from Exercise 4
Lab4 lab4 = new Lab4();
XSDSchema schema = lab4.createSchema(xsdFile); //Schema from lab4

//Lets get the PurchaseOrderType complex type
XSDComplexTypeDefinition purchaseOrderType = schema.resolveComplexTypeDefinition("Purcha

//Lets get the model group that was created in Exercise 4
XSDModelGroup modelGroup = (XSDModelGroup)((XSDParticle)purchaseOrderType.getContent())

/**
 * Create a global element with name="note" and type="string"
 */
```

```
XSDElementDeclaration noteElement = XSDFactory.eINSTANCE.createXSDElementDeclaration();

//Set the name to "notes"
noteElement.setName("notes");

//Set the type of notes element to string
XSDSimpleTypeDefinition stringType = schema.getSchemaForSchema().
        resolveSimpleTypeDefinition(XSDConstants.SCHEMA_FOR_SCHEMA_URI_2001,"string");
noteElement.setTypeDefinition(stringType);

//Add the global element to the root schema
schema.getContents().add(noteElement);

/**
 * Create local element with name="state", type="po:USState" and minOccurs="0"
 * Local elements have particles as parents
 */
XSDParticle localElementParticle = XSDFactory.eINSTANCE.createXSDParticle();

XSDElementDeclaration stateElement = XSDFactory.eINSTANCE.createXSDElementDeclaration();
localElementParticle.setContent(stateElement);

//Set the name to state
stateElement.setName("state");

//Set the type to USState
stateElement.setTypeDefinition(schema.resolveSimpleTypeDefinition("USState"));

//Set the minOccurs="0"
localElementParticle.setMinOccurs(0);

//Add the new particle to the modelgroup
modelGroup.getContents().add(localElementParticle);

/**
 * Create the element reference to the global element note
 */
XSDParticle elementRefParticle = XSDFactory.eINSTANCE.createXSDParticle();

XSDElementDeclaration notesRef = XSDFactory.eINSTANCE.createXSDElementDeclaration();
elementRefParticle.setContent(notesRef);

//Set the element reference to the global element notes
notesRef.setResolvedElementDeclaration(noteElement);

//Add the new particle to the modelgroup
modelGroup.getContents().add(elementRefParticle);

//Save the contents of the resource to the file system.
schema.eResource().save(Collections.EMPTY_MAP);
```

Attribute declarations

Complex Type definitions contain attributes, which are used by complex elements. Attributes themselves are always declared with a simple type. This means that an element with attributes always has a complex structure.

Attribute declarations can be global, local, or references, and are used for:

- Local validation of attribute information item values using a simple type definition

- Specifying default or fixed values for attribute information items

The <attribute> element contains attributes, which are described by *XML Schema Part 1: Structures* of the standard.

The XML Schema model java interface for a <attribute> is org.eclipse.xsd.XSDAttributeDeclaration and can be created by calling org.eclipse.xsd.XSDFactory.eINSTANCE.createXSDAttributeDeclaration().

Attribute use

Local attribute declarations and attribute references can be optional, required, or prohibited by changing the attribute use usage attribute.

If an attribute is local or a reference, it can contain attributes, which are described by *http://www.w3.org/TR/xmlschema-1/#cAttributeUse* .

Referencing a global attribute ceclaration

If you declare a global attribute declaration, you can reference this attribute in the remainder of the schema using the ref attribute. This is done by calling the following method on the XML Schema model: org.eclipse.xsd.XSDAttributeDeclaration.setResolvedAttributeDeclaration(XSDAttributeDeclaration).

Any attribute

The <anyAttribute> element enables us to extend the XML instance document with attributes not specified by the schema.

Exercise 6: Create a global attribute group, global attribute, local attribute, and attribute reference

Using the XML Schema Infoset Model APIs and adding on to the schema created in Exercise 5: Create a global, local, and element reference on page 19 , create the following schema document:

```
<?xml version="1.0"?>
<schema
  targetNamespace="http://www.eclipse.org/xsd/examples/po"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:po="http://www.eclipse.org/xsd/examples/po">

  <simpleType name="USState">
    <restriction base="string">
      <enumeration value="AK"/>
      <enumeration value="AL"/>
      <enumeration value="AR"/>
    </restriction>
  </simpleType>

  <simpleType>
    <list itemType="po:USState"/>
  </simpleType>
```

```
  <complexType name="PurchaseOrderType">
    <sequence>
      <element minOccurs="0" name="state" type="po:USState"/>
      <element ref="po:notes"/>
    </sequence>
    <attribute name="color" type="string"/>
    <attribute ref="po:lang"/>
    <attributeGroup ref="po:shapeAttributes"/>
  </complexType>

  <element name="notes" type="string"/>

  <attributeGroup name="shapeAttributes">
    <attribute name="size" type="string" use="optional"/>
    <attribute name="length" type="int"/>
  </attributeGroup>

  <attribute name="lang" type="string"/>

</schema>
```

Hint: You will need to create an org.eclipse.xsd.XSDAttributeUse for the local attribute and attribute reference.

New constructs created in this exercise:

- <attributeGroup> - Global attribute group with name="shapeAttributes"
- <attributeGroup> - Attribute group reference with ref="po:shapeAttributes"
- <attribute> - Local attribute with name="size", type="string" and use="required"
- <attribute> - Local attribute with name="length", type="int" and use="optional"
- <attribute> - Local attribute with name="color", type="string"
- <attribute> - Global attribute with name="lang", type="string"
- <attribute> - Attribute reference with ref="po:lang"

Solution: Exercise 6 solution on page 23

Exercise 6 solution

```
//Exercise 6 builds from Exercise 5
Lab5 lab5 = new Lab5();
XSDSchema schema = lab5.createSchema(xsdFile);

/**
 * Create global attributeGroup with name="shapeAttributes" and has the following
 * attributes:
 *   <attributeGroup name="shapeAttributes" />
 *     <attribute name="size" type="string" use="required" />
 *     <attribute name="length" type="int" use="optional" />
 *   </attributeGroup>
 */
XSDAttributeGroupDefinition shapeAttributes = XSDFactory.eINSTANCE.createXSDAttributeGro

//Set the name = "shapeAttributes"
shapeAttributes.setName("shapeAttributes");
```

```
/**
 * Create local attribute size, which need XSDAttributeUse to store the usage
 */
XSDAttributeUse sizeAttrUse = XSDFactory.eINSTANCE.createXSDAttributeUse();

//Set the use="required"
sizeAttrUse.setUse(XSDAttributeUseCategory.REQUIRED_LITERAL);

XSDAttributeDeclaration sizeAttr = XSDFactory.eINSTANCE.createXSDAttributeDeclaration();

//Set the name="size"
sizeAttr.setName("size");

//Set the type="string"
XSDSimpleTypeDefinition stringType = schema.getSchemaForSchema().
        resolveSimpleTypeDefinition(XSDConstants.SCHEMA_FOR_SCHEMA_URI_2001,"string");
sizeAttr.setTypeDefinition(stringType);

//Add the sizeAttr to the sizeAttrUse
sizeAttrUse.setContent(sizeAttr);

//Add the sizeAttrUse to the global attribute group
shapeAttributes.getContents().add(sizeAttrUse);

/**
 * Create local attribute length, which need XSDAttributeUse to store the usage
 */
XSDAttributeUse lengthAttrUse = XSDFactory.eINSTANCE.createXSDAttributeUse();

//Set the use="optional"
sizeAttrUse.setUse(XSDAttributeUseCategory.OPTIONAL_LITERAL);

XSDAttributeDeclaration lengthAttr = XSDFactory.eINSTANCE.createXSDAttributeDeclaration(

//Set the name="length"
lengthAttr.setName("length");

//Set the type="int"
XSDSimpleTypeDefinition intType = schema.getSchemaForSchema().
        resolveSimpleTypeDefinition(XSDConstants.SCHEMA_FOR_SCHEMA_URI_2001,"int");
lengthAttr.setTypeDefinition(intType);

//Add the lengthAttr to the lengthAttrUse
lengthAttrUse.setContent(lengthAttr);

//Add the lengthAttrUse to the global attribute group
shapeAttributes.getContents().add(lengthAttrUse);

//Add the new attribute group to the root schema
schema.getContents().add(shapeAttributes);

/**
 * Now lets create and add some attributes to the given complexType
 *
 * <attribute name="lang" type="string" >
 *
 * <complexType name="PurchaseOrderType" >
 *   <sequence>
 *     <element name="state" type="po:USState" minOccurs="0" />
 *     <element ref="po:notes" />
 *   </sequence>
 *   <attribute name="color" type="string" />
 *   <attribute ref="po:lang" />
 *   <attributeGroup ref="po:shapeAttributes" />
```

```
 * </complexType>
 *
 * Create Global Attribute name="lang" and type="string"
 * Create the local attribute color type="string"
 * Create then attribute reference to global attribute po:lang
 * Create attributeGroup reference to global attributeGroup po:shapeAttributes
 */

//Create Global Attribute name="lang" and type="string"
XSDAttributeDeclaration langAttr = XSDFactory.eINSTANCE.createXSDAttributeDeclaration();

//Set name="lang"
langAttr.setName("lang");

//Set type="string"
langAttr.setTypeDefinition(stringType);

//Add the global attribute to the given schema
schema.getContents().add(langAttr);

//Lets get the PurchaseOrderType complex type
XSDComplexTypeDefinition purchaseOrderType = schema.resolveComplexTypeDefinition("Purcha

//Create the local attribute color type="string"
XSDAttributeUse colorAttrUse = XSDFactory.eINSTANCE.createXSDAttributeUse();

XSDAttributeDeclaration colorAttr = XSDFactory.eINSTANCE.createXSDAttributeDeclaration()

//Set the name="color"
colorAttr.setName("color");

//Set type="string"
colorAttr.setTypeDefinition(stringType);

//Add the colorAttr to the colorAttrUse
colorAttrUse.setContent(colorAttr);

//Add the colorAttrUse to the complex type
purchaseOrderType.getAttributeContents().add(colorAttrUse);

//Create then attribute reference to global attribute po:lang
XSDAttributeUse langRefAttrUse = XSDFactory.eINSTANCE.createXSDAttributeUse();

XSDAttributeDeclaration langRefAttr = XSDFactory.eINSTANCE.createXSDAttributeDeclaration

//Set the reference="po:lang"
langRefAttr.setResolvedAttributeDeclaration(langAttr);

//Add the langRefAttr to the langRefAttrUse
langRefAttrUse.setContent(langRefAttr);

//Add the langRefAttrUse to the complex type
purchaseOrderType.getAttributeContents().add(langRefAttrUse);

//Create attributeGroup reference to global attributeGroup po:shapeAttributes
XSDAttributeGroupDefinition shapeAttributesRef = XSDFactory.eINSTANCE.createXSDAttribute

//Set the resolved attributeGroup
shapeAttributesRef.setResolvedAttributeGroupDefinition(shapeAttributes);

//Add the attribute group reference to the complex type
purchaseOrderType.getAttributeContents().add(shapeAttributesRef);

// Save the contents of the resource to the file system.
```

```
schema.eResource().save(Collections.EMPTY_MAP);
```

Section 7. Pulling it all together

Exercise 7: Create purchase order schema

Using the XML Schema Infoset Model APIs and the different techniques described in this tutorial, create the po.xsd, which is defined in the *XML Schema Part 0: Primer.*

Hint: You will notice that some of the elements defined in the po.xsd schema contain anonymous/local complex types. Local complex types are created the same way as if they were global, which was done in Exercise 4: Create a global complex type with a sequence model group on page 15 , but don't set the name of the complex type, and instead of adding it to the root schema you just need to use the setAnonymousTypeDefinition(XSDTypeDefinition) on the org.eclipse.xsd.XSDElementDeclaration object.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
      <xsd:element name="shipTo" type="USAddress"/>
      <xsd:element name="billTo" type="USAddress"/>
      <xsd:element minOccurs="0" ref="comment"/>
      <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 fixed="US" name="country" type="xsd:NMTOKEN"/>
  </xsd:complexType>

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

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

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

</xsd:schema>
```

Solution: Exercise 7 solution on page 28

Exercise 7 solution

```
//Create the root schema
//Get the URI of the model file.
URI fileURI = URI.createPlatformResourceURI(xsdFile.getFullPath().toString());

//Create a resource set to manage the different resources
ResourceSet resourceSet = new ResourceSetImpl();

//Create a resource for this file.
Resource resource = resourceSet.createResource(fileURI);

XSDFactory xsdFactory = XSDFactory.eINSTANCE;

//Create the root XSDSchema object
XSDSchema schema = xsdFactory.createXSDSchema();

//set the schema for schema QName prefix to "xsd"
schema.setSchemaForSchemaQNamePrefix("xsd");
java.util.Map qNamePrefixToNamespaceMap = schema.getQNamePrefixToNamespaceMap();

//put the following namespace in the root schema namespace map
//xsd:http://www.w3.org/2001/XMLSchema
qNamePrefixToNamespaceMap.put(schema.getSchemaForSchemaQNamePrefix(),
        XSDConstants.SCHEMA_FOR_SCHEMA_URI_2001);

//We call updateElement to synchronize the MOF model with the underlying DOM model
//This should only have to be done after creating a new model
schema.updateElement();

//Add the root schema to the resource that was created above
resource.getContents().add(schema);

/**
 * Create the global constructs
 * Create Global Complex Type with name="PurchaseOrderType"
 * Create Global Complex Type with name="USAddress"
 * Create Global Complex Type with name="Items"
 * Create Global Simple Type with name="SKU"
 * Create Global Element with name="purchaseOrder" type="PurchaseOrderType"
 * Create Global Element with name="comment" type="string"
```

```
 */

//Create Global Complex Type with name="PurchaseOrderType"
XSDComplexTypeDefinition poType = xsdFactory.createXSDComplexTypeDefinition();
poType.setName("PurchaseOrderType");
schema.getContents().add(poType);

//Create Global Complex Type with name="USAddress"
XSDComplexTypeDefinition usAddressType = xsdFactory.createXSDComplexTypeDefinition();
usAddressType.setName("USAddress");
schema.getContents().add(usAddressType);

//Create Global Complex Type with name="Items"
XSDComplexTypeDefinition itemsType = xsdFactory.createXSDComplexTypeDefinition();
itemsType.setName("Items");
schema.getContents().add(itemsType);

//Create Global Simple Type with name="SKU"
XSDSimpleTypeDefinition skuType = xsdFactory.createXSDSimpleTypeDefinition();
skuType.setName("SKU");
schema.getContents().add(skuType);

//Create Global Element with name="purchaseOrder" type="PurchaseOrderType"
XSDElementDeclaration poElement = xsdFactory.createXSDElementDeclaration();
poElement.setName("purchaseOrder");
poElement.setTypeDefinition(poType);
schema.getContents().add(poElement);

//Create Global Element with name="comment" type="string"
XSDElementDeclaration commentElement = xsdFactory.createXSDElementDeclaration();
commentElement.setName("comment");
XSDSimpleTypeDefinition stringType = schema.getSchemaForSchema().
        resolveSimpleTypeDefinition(XSDConstants.SCHEMA_FOR_SCHEMA_URI_2001,"string");
commentElement.setTypeDefinition(stringType);
schema.getContents().add(commentElement);


/**
 * Create the following contents under the PurchaseOrderType complex type
 *
 *   <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>
 */
XSDParticle poSequenceParticle = xsdFactory.createXSDParticle();
XSDModelGroup poSeqeuence = xsdFactory.createXSDModelGroup();
poSeqeuence.setCompositor(XSDCompositor.SEQUENCE_LITERAL);

poSequenceParticle.setContent(poSeqeuence);
poType.setContent(poSequenceParticle);

//Create local element name="shipTo" type="USAddress"
XSDElementDeclaration shipToElement = xsdFactory.createXSDElementDeclaration();
XSDParticle shipToParticle = xsdFactory.createXSDParticle();
shipToParticle.setContent(shipToElement);

shipToElement.setName("shipTo");
shipToElement.setTypeDefinition(usAddressType);
```

```
poSeqeuence.getContents().add(shipToParticle);

//Create local element name="billTo" type="USAddress"
XSDElementDeclaration billToElement = xsdFactory.createXSDElementDeclaration();
XSDParticle billToParticle = xsdFactory.createXSDParticle();
billToParticle.setContent(billToElement);

billToElement.setName("billTo");
billToElement.setTypeDefinition(usAddressType);
poSeqeuence.getContents().add(billToParticle);

//Create element ref="comment" minOccurs="0"
XSDElementDeclaration commentRef = xsdFactory.createXSDElementDeclaration();
XSDParticle commentRefParticle = xsdFactory.createXSDParticle();
commentRefParticle.setContent(commentRef);

commentRef.setResolvedElementDeclaration(commentElement);
commentRefParticle.setMinOccurs(0);
poSeqeuence.getContents().add(commentRefParticle);

//Create local element name="items" type="Items"
XSDElementDeclaration itemsElement = xsdFactory.createXSDElementDeclaration();
XSDParticle itemsParticle = xsdFactory.createXSDParticle();
itemsParticle.setContent(itemsElement);

itemsElement.setName("items");
itemsElement.setTypeDefinition(itemsType);
poSeqeuence.getContents().add(itemsParticle);

//Create local attribute name="orderDate" type="xsd:date"
XSDAttributeDeclaration orderDate = xsdFactory.createXSDAttributeDeclaration();
XSDAttributeUse orderDateUse = xsdFactory.createXSDAttributeUse();
orderDateUse.setContent(orderDate);

orderDate.setName("orderDate");
XSDSimpleTypeDefinition dateType = schema.getSchemaForSchema().
        resolveSimpleTypeDefinition(XSDConstants.SCHEMA_FOR_SCHEMA_URI_2001,"date");
orderDate.setTypeDefinition(dateType);
poType.getAttributeContents().add(orderDateUse);


/**
 * Create the following contents under the USAddress complex type
 *
 *   <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>
 */
XSDParticle addressSequenceParticle = xsdFactory.createXSDParticle();
XSDModelGroup addressSeqeuence = xsdFactory.createXSDModelGroup();
addressSeqeuence.setCompositor(XSDCompositor.SEQUENCE_LITERAL);

addressSequenceParticle.setContent(addressSeqeuence);
usAddressType.setContent(addressSequenceParticle);

//Create local element name="name" type="xsd:string"
XSDElementDeclaration nameElement = xsdFactory.createXSDElementDeclaration();
```

```
XSDParticle nameParticle = xsdFactory.createXSDParticle();
nameParticle.setContent(nameElement);

nameElement.setName("name");
nameElement.setTypeDefinition(stringType);
addressSeqeuence.getContents().add(nameParticle);

//Create local name="street" type="xsd:string"
XSDElementDeclaration streetElement = xsdFactory.createXSDElementDeclaration();
XSDParticle streetParticle = xsdFactory.createXSDParticle();
streetParticle.setContent(streetElement);

streetElement.setName("street");
streetElement.setTypeDefinition(stringType);
addressSeqeuence.getContents().add(streetParticle);

//Create local element name="city" type="xsd:string"
XSDElementDeclaration cityElement = xsdFactory.createXSDElementDeclaration();
XSDParticle cityParticle = xsdFactory.createXSDParticle();
cityParticle.setContent(cityElement);

cityElement.setName("city");
cityElement.setTypeDefinition(stringType);
addressSeqeuence.getContents().add(cityParticle);

//Create local element name="state" type="xsd:string"
XSDElementDeclaration stateElement = xsdFactory.createXSDElementDeclaration();
XSDParticle stateParticle = xsdFactory.createXSDParticle();
stateParticle.setContent(stateElement);

stateElement.setName("state");
stateElement.setTypeDefinition(stringType);
addressSeqeuence.getContents().add(stateParticle);

//Create local element name="zip" type="xsd:string"
XSDElementDeclaration zipElement = xsdFactory.createXSDElementDeclaration();
XSDParticle zipParticle = xsdFactory.createXSDParticle();
zipParticle.setContent(zipElement);

zipElement.setName("zip");
XSDSimpleTypeDefinition decimalType = schema.getSchemaForSchema().
        resolveSimpleTypeDefinition(XSDConstants.SCHEMA_FOR_SCHEMA_URI_2001,"decimal");
zipElement.setTypeDefinition(decimalType);
addressSeqeuence.getContents().add(zipParticle);

//Create local attribute name="country" type="xsd:NMTOKEN" fixed="US"
XSDAttributeDeclaration countryAttribute = xsdFactory.createXSDAttributeDeclaration();
XSDAttributeUse countryUse = xsdFactory.createXSDAttributeUse();
countryUse.setContent(countryAttribute);

countryAttribute.setName("country");
XSDSimpleTypeDefinition nmtokenType = schema.getSchemaForSchema().
        resolveSimpleTypeDefinition(XSDConstants.SCHEMA_FOR_SCHEMA_URI_2001,"NMTOKEN");
countryAttribute.setTypeDefinition(nmtokenType);

countryAttribute.setConstraint(XSDConstraint.FIXED_LITERAL);
countryAttribute.setLexicalValue("US");
usAddressType.getAttributeContents().add(countryUse);

/**
 * Create the following contents under the Items complex type
 *   <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>
 */
XSDParticle itemsSequenceParticle = xsdFactory.createXSDParticle();
XSDModelGroup itemsSeqeuence = xsdFactory.createXSDModelGroup();
itemsSeqeuence.setCompositor(XSDCompositor.SEQUENCE_LITERAL);

itemsSequenceParticle.setContent(itemsSeqeuence);
itemsType.setContent(itemsSequenceParticle);

//Create local name="item" minOccurs="0" maxOccurs="unbounded"
XSDElementDeclaration itemElement = xsdFactory.createXSDElementDeclaration();
XSDParticle itemParticle = xsdFactory.createXSDParticle();
itemParticle.setContent(itemElement);

itemElement.setName("item");
itemParticle.setMinOccurs(0);
itemParticle.setMaxOccurs(-1);

itemsSeqeuence.getContents().add(itemParticle);

//Create anonymous complex type for the item local element
XSDComplexTypeDefinition itemAnonType = xsdFactory.createXSDComplexTypeDefinition();
itemElement.setAnonymousTypeDefinition(itemAnonType);

XSDParticle itemAnonTypeSequenceParticle = xsdFactory.createXSDParticle();
XSDModelGroup itemAnonTypeSeqeuence = xsdFactory.createXSDModelGroup();
itemAnonTypeSeqeuence.setCompositor(XSDCompositor.SEQUENCE_LITERAL);
itemAnonTypeSequenceParticle.setContent(itemAnonTypeSeqeuence);

itemAnonType.setContent(itemAnonTypeSequenceParticle);

//Create local element name="productName" type="xsd:string"
XSDElementDeclaration productNameElement = xsdFactory.createXSDElementDeclaration();
XSDParticle productNameParticle = xsdFactory.createXSDParticle();
productNameParticle.setContent(productNameElement);

productNameElement.setName("productName");
productNameElement.setTypeDefinition(stringType);
itemAnonTypeSeqeuence.getContents().add(productNameParticle);

//Create local element name="quantity" with anon simpleType with restriction="xsd:positi
//Create a maxExclusive facet on the anonymouse simple type with value="100"
XSDElementDeclaration quantityElement = xsdFactory.createXSDElementDeclaration();
XSDParticle quantityParticle = xsdFactory.createXSDParticle();
quantityParticle.setContent(quantityElement);
```

```
quantityElement.setName("quantity");
XSDSimpleTypeDefinition anonSimpleType = xsdFactory.createXSDSimpleTypeDefinition();
XSDSimpleTypeDefinition positiveIntegerType = schema.getSchemaForSchema().
        resolveSimpleTypeDefinition(XSDConstants.SCHEMA_FOR_SCHEMA_URI_2001,"positiveInt
anonSimpleType.setBaseTypeDefinition(positiveIntegerType);

XSDMaxExclusiveFacet exclusiveFacet = xsdFactory.createXSDMaxExclusiveFacet();
exclusiveFacet.setLexicalValue("100");
anonSimpleType.getFacetContents().add(exclusiveFacet);

quantityElement.setAnonymousTypeDefinition(anonSimpleType);

itemAnonTypeSeqeuence.getContents().add(quantityParticle);

//Create local element name="USPrice" type="xsd:decimal
XSDElementDeclaration usPriceElement = xsdFactory.createXSDElementDeclaration();
XSDParticle usPriceParticle = xsdFactory.createXSDParticle();
usPriceParticle.setContent(usPriceElement);

usPriceElement.setName("USPrice");
usPriceElement.setTypeDefinition(decimalType);
itemAnonTypeSeqeuence.getContents().add(usPriceParticle);

//Create element ref="comment" minOccurs="0"
commentRef = xsdFactory.createXSDElementDeclaration();
commentRefParticle = xsdFactory.createXSDParticle();
commentRefParticle.setContent(commentRef);

commentRef.setResolvedElementDeclaration(commentElement);
commentRefParticle.setMinOccurs(0);
itemAnonTypeSeqeuence.getContents().add(commentRefParticle);

//Create local element name="shipDate" type="xsd:date" minOccurs="0"
XSDElementDeclaration shipDateElement = xsdFactory.createXSDElementDeclaration();
XSDParticle shipDateParticle = xsdFactory.createXSDParticle();
shipDateParticle.setContent(shipDateElement);

shipDateElement.setName("shipDate");
shipDateParticle.setMinOccurs(0);
shipDateElement.setTypeDefinition(dateType);
itemAnonTypeSeqeuence.getContents().add(shipDateParticle);

//Create local attribute name="partNum" type="SKU" use="required"
XSDAttributeDeclaration partNum = xsdFactory.createXSDAttributeDeclaration();
XSDAttributeUse partNumUse = xsdFactory.createXSDAttributeUse();
partNumUse.setContent(partNum);

partNum.setName("partNum");
partNum.setTypeDefinition(skuType);
partNumUse.setUse(XSDAttributeUseCategory.REQUIRED_LITERAL);
itemAnonType.getAttributeContents().add(partNumUse);

/**
 * Create the following contents under the SKU simple type
 *   <xsd:simpleType name="SKU">
 *     <xsd:restriction base="xsd:string">
 *               <xsd:pattern value="\d{3}-[A-Z]{2}" />
 *             </xsd:restriction>
 *   </xsd:simpleType>
 */
skuType.setBaseTypeDefinition(stringType);

//Create pattern facet value="\d{3}-[A-Z]{2}"
XSDPatternFacet pattern = xsdFactory.createXSDPatternFacet();
```

```
pattern.setLexicalValue("\\d{3}-[A-Z]{2}");

skuType.getFacetContents().add(pattern);


// Save the contents of the resource to the file system.
resource.save(Collections.EMPTY_MAP);
```

Section 8. You try it!

Copying the code

All of the code snippets used in this tutorial are bundled in this Eclipse plug-in: org.eclipse.xsd.examples.zip.

To install this plug-in and examine the source code:

1. Unzip org.eclipse.xsd.examples.zip into a temporary directory.
2. Create a Simple project called org.eclipse.xsd.examples in the Eclipse Workbench.
3. Copy the contents of the org.eclipse.xsd.examples.zip file into the newly created Eclipse project. Click Yes if a dialog asks if you want to override the .project file.
4. Expand the org.eclipse.xsd.examples/src directory to view the source.
5. The org.eclipse.xsd.examples.schema2 package contains the 7 exercises from this tutorial.

If you have compiler errors, you can fix these by adjusting the "**Windows > Preferences > Plug-In Development > Target Platform**" setting and then updating the project's classpath.

Troubleshooting and bugs

All bugs should be reported in the Eclipse Bugzilla server at:

http://bugs.eclipse.org/bugs

Section 9. Summary and resources

Summary

In this tutorial, we reviewed the steps in creating all of the different constructs in the XML Schema Infoset model, including:

- Global complex types, local/anonymous complex types
- Global simple types, local/anonymous simple types
- Facets and value constraints
- Named groups
- Model groups (sequence, choice, and all)
- Global attribute groups and attribute group references
- Global elements, local elements, and element references
- Global attributes, local attributes, and attribute references

As XML data is becoming more popular, we are seeing more and more companies depending on common definitions, which are being built using XML Schemas. The *XML Schema Infoset Model* provides a well-designed set of interfaces, implementations, and algorithms for representing and manipulating XML Schema models. Hopefully the two tutorials in this series have demonstrated how powerful the XML Schema Infoset Model is.

Resources

Part 1 of this tutorial series

Part 1 of this tutorial series shows how to set up the development environment, use the XML Schema Infoset Model classes, load and create XML Schema models, and link XML Schema models together.

For more information on using XML Schema or the Eclipse Modeling Framework, see the following suggested material:

XML Schema resources

XML Schema Infoset Model

http://www.eclipse.org/xsd/

XML Schema Infoset Model FAQ

http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/xsd-home/faq.html

XML Schema Infoset Model Drivers

http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/xsd-home/downloads/dl.html

XML Schema Infoset Model newsgroup

news://news.eclipse.org/eclipse.technology.xsd

XML Schema Tutorial, Roger L. Costello, September 2001

http://www.xfront.com/

XML Schema Part 0: Primer

http://www.w3.org/TR/xmlschema-0/

XML Schema Part 1: Structures

http://www.w3.org/TR/xmlschema-1/

XML Schema Part 2: Datatypes

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

Eclipse Modeling Framework resources

Eclipse Modeling Framework

http://www.eclipse.org/emf/

EMF Users' Guide

http://dev.eclipse.org/viewcvs/indextools.cgi/~checkout~/emf-home/docs/emfug.pdf

Feedback

We welcome your feedback on this tutorial, and look forward to hearing from you. We'd also like to hear about other tutorial topics you'd like to see covered. Thanks! For questions about the content of this tutorial, contact the author, Dave Spriet, at *spriet@ca.ibm.com*

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial Building tutorials with the Toot-O-Matic demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.