WHITE PAPER

A First Step to Understanding XML Schemas

By: Travis Vandersypen, President of DilMad Enterprises, Inc.

Abstract

Document Type Definitions have generated quite a few complaints since they were introduced. As a result, the W3C set about creating a new standard for defining a document's structure. What they the W3C created is something even more complex and flexible than DTDs: the XML Schema Definition Language.

The XML Schema Definition Language solves a number of problems posed with Document Type Definitions. For instance, because the language for specifying DTDs is inherently different from the XML document it is describing, DTDs can be difficult to read and understand. Another limitation of a DTD is the method in which data is handled. Unfortunately, DTDs only support character data types: DTDs are unable to make a distinction between the various different data types, such as numerics, dates, and so on. They are all considered character data types. And, probably the most important and notable drawback of using DTDs is the inability of the DTD to provide support for mixing elements from various different documents stored in separate namespaces.

Schemas, while more complex than DTDs, also give an individual much more power and control over how XML documents are validated. For instance, with the new W3C standard, a document definition can specify the data type of an element's contents, the range of values for elements, what the minimum, as well as maximum, number of times an element may occur, annotations to schemas, and much more.

Contents

Introduction to the W3C Schema Recommendation

In May of 2001, the W3C finalized its recommendation for the XML Schema Definition Language. This standard allows an author to define simple and complex elements and the rules governing how those elements and their attributes may show up within an instance document. The author has a large amount of control over how the structure of a conforming XML document must be created. The author can apply various restrictions to the elements and attributes within the document, from specifying the length to specifying an enumerated set of acceptable values for the element or attribute. With the XML Schema Definition Language, an XML schema author possesses an incredible amount of control over the conformance of an associated XML document to the specified schema. You can find more information on the W3C at http://www.w3c.orgwww.w3c.org.

Additionally, the W3C XML Schema recommendation is separated into 3 three main documents, which can be found at:

- The XML Schema Primer (Part 0), which can be found at http://www.w3c.org/TR/xmlschema-0/ - .The XML Schema Primer (Part 0)
- The XML Schema Structures (Part 1), which can be found at http://www.w3c.org/TR/xmlschema-1/ -. The XML Schema Structures (Part 1)
- The XML Schema Data Types (Part 2), which can be found at http://www.w3c.org/TR/xmlschema-2/. - The XML Schema Data Types (Part 2)
- You can find additional information on the XML Schema Definition Language at http://www.w3c.org/XML/Schema.

Sample XML Document

The rest of this paper is devoted to creating and understanding the XML schema for the XML document shown in Listing 1, which details a purchase order for various items that can commonly be found in a grocery store. This document allows one individual to receive the shipment of the goods and an entirely different individual to pay for the purchase. This document also contains specific information about the products ordered, such as how much each product is, how many were ordered, etc. and so on.

Listing 1: PurchaseOrder.xml Contains a Sample Purchase Order for Common Items Found in a Grocery Store.

```
<PurchaseOrder Tax="5.76" Total="75.77">

 <ShippingInformation>
  <Name>Dillon Larsen</Name>
  <Address>
   <Street>123 Jones Rd.</Street>
   <City>Houston</City>
```

```
    <State>TX</State>
    <Zip>77381</Zip>
  </Address>
  <Method>USPS</Method>
  <DeliveryDate>2001-08-12</DeliveryDate>
 </ShippingInformation>

 <BillingInformation>
  <Name>Madi Larsen</Name>
  <Address>
    <Street>123 Jones Rd.</Street>
    <City>Houston</City>
    <State>TX</State>
    <Zip>77381</Zip>
  </Address>
  <PaymentMethod>Credit Card</PaymentMethod>
  <BillingDate>2001-08-09</BillingDate>
 </BillingInformation>

 <Order SubTotal="70.01" ItemsSold="17">
  <Product Name="Baby Swiss" Id="702890" Price="2.89" Quantity="1"/>
  <Product Name="Hard Salami" Id="302340" Price="2.34" Quantity="1"/>
  <Product Name="Turkey" Id="905800" Price="5.80" Quantity="1"/>
  <Product Name="Caesar Salad" Id="991687" Price="2.38" Quantity="2"/>
  <Product Name="Chicken Strips" Id="133382" Price="2.50" Quantity="1"/>
  <Product Name="Bread" Id="298678" Price="1.08" Quantity="1"/>
  <Product Name="Rolls" Id="002399" Price="2.24" Quantity="1"/>
  <Product Name="Cereal" Id="066510" Price="2.18" Quantity="1"/>
  <Product Name="Jalapenos" Id="101005" Price="1.97" Quantity="1"/>
  <Product Name="Tuna" Id="000118" Price="0.92" Quantity="3"/>
  <Product Name="Mayonnaise" Id="126860" Price="1.98" Quantity="1"/>
  <Product Name="Top Sirloin" Id="290502" Price="9.97" Quantity="2"/>
  <Product Name="Soup" Id="001254" Price="1.33" Quantity="1"/>
  <Product Name="Granola Bar" Id="026460" Price="2.14" Quantity="2"/>
  <Product Name="Chocolate Milk" Id="024620" Price="1.58" Quantity="2"/>
  <Product Name="Spaghetti" Id="000265" Price="1.98" Quantity="1"/>
  <Product Name="Laundry Detergent" Id="148202" Price="8.82"
Quantity="1"/>
 </Order>

</PurchaseOrder>
```

As you can see, Listing 1 represents a fairly small and simple order that could be placed online. It contains the necessary information necessary regarding how payment is to be made and, how the order is to be shipped, as well and what day delivery should be. The preceding listing should by no means be construed as an all-inclusive document for an online grocery store order; it has been constructed for use as an example within this book only.

Until the XML Schema Definition Language recommendation was finalized, most authors, in the face of ever-changing standards, decided to stick with a finalized standard of

DTDs. For the preceding listing, an author might construct the DTD as shown in Listing 2.

Listing 2: PurchaseOrder.dtd Contains a Sample DTD for PurchaseOrder.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT PurchaseOrder (ShippingInformation, BillingInformation, Order)>
<!ATTLIST PurchaseOrder
  Tax CDATA #IMPLIED
  Total CDATA #IMPLIED
>
<!ELEMENT ShippingInformation (Name, Address, (((BillingDate,
PaymentMethod)) | ((DeliveryDate, Method))))>
<!ELEMENT BillingInformation (Name, Address, (((BillingDate, PaymentMethod))
| ((DeliveryDate, Method))))>
<!ELEMENT Order (Product+)>
<!ATTLIST Order
  SubTotal CDATA #IMPLIED
  ItemsSold CDATA #IMPLIED
>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Address (Street, City, State, Zip)>
<!ELEMENT BillingDate (#PCDATA)>
<!ELEMENT PaymentMethod (#PCDATA)>
<!ELEMENT DeliveryDate (#PCDATA)>
<!ELEMENT Method (#PCDATA)>
<!ELEMENT Product EMPTY>
<!ATTLIST Product
  Name CDATA #IMPLIED
  Id CDATA #IMPLIED
  Price CDATA #IMPLIED
  Quantity CDATA #IMPLIED
>
<!ELEMENT Street (#PCDATA)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT State (#PCDATA)>
<!ELEMENT Zip (#PCDATA)>
```

Schema for the XML Document

So, now that you've seen the DTD for the XML document in Listing 1, what would the comparative XML schema for it look like? Although the DTD in Listing 2 manages to describe the XML document in Listing 1 in a total of 30 lines, creating an XML schema is not quite so easy. Given the document in Listing 1, the XML schema for it is shown in Listing 3.

Listing 3: PurchaseOrder.xsd Contains the Schema Definition for PurchaseOrder.xml.

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

  <xsd:annotation>
    <xsd:documentation>
      Purchase Order schema for an online grocery store.
```

```
    </xsd:documentation>
  </xsd:annotation>

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

  <xsd:complexType name="PurchaseOrderType">
    <xsd:all>
      <xsd:element name="ShippingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="BillingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="Order" type="OrderType" minOccurs="1"
maxOccurs="1"/>
    </xsd:all>
    <xsd:attribute name="Tax">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="Total">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>

  <xsd:group name="ShippingInfoGroup">
    <xsd:all>
      <xsd:element name="DeliveryDate" type="DateType"/>
      <xsd:element name="Method" type="DeliveryMethodType"/>
    </xsd:all>
  </xsd:group>

  <xsd:group name="BillingInfoGroup">
    <xsd:all>
      <xsd:element name="BillingDate" type="DateType"/>
      <xsd:element name="PaymentMethod" type="PaymentMethodType"/>
    </xsd:all>
  </xsd:group>

  <xsd:complexType name="InfoType">
    <xsd:sequence>
      <xsd:element name="Name" minOccurs="1" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
      </xsd:element>
```

```
    <xsd:element name="Address" type="AddressType" minOccurs="1"
maxOccurs="1"/>
    <xsd:choice minOccurs="1" maxOccurs="1">
     <xsd:group ref="BillingInfoGroup"/>
     <xsd:group ref="ShippingInfoGroup"/>
    </xsd:choice>
   </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="DateType">
   <xsd:restriction base="xsd:date"/>
  </xsd:simpleType>

  <xsd:simpleType name="DeliveryMethodType">
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="USPS"/>
    <xsd:enumeration value="UPS"/>
    <xsd:enumeration value="FedEx"/>
    <xsd:enumeration value="DHL"/>
    <xsd:enumeration value="Other"/>
   </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="PaymentMethodType">
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Check"/>
    <xsd:enumeration value="Cash"/>
    <xsd:enumeration value="Credit Card"/>
    <xsd:enumeration value="Debit Card"/>
    <xsd:enumeration value="Other"/>
   </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="AddressType">
   <xsd:all>
    <xsd:element name="Street" minOccurs="1">
     <xsd:simpleType>
      <xsd:restriction base="xsd:string"/>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="City" minOccurs="1" maxOccurs="1">
     <xsd:simpleType>
      <xsd:restriction base="xsd:string"/>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="State" type="StateType" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="Zip" type="ZipType" minOccurs="1"
maxOccurs="1"/>
   </xsd:all>
  </xsd:complexType>
```

```
  <xsd:simpleType name="ZipType">
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="5"/>
      <xsd:maxLength value="10"/>
      <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="StateType">
    <xsd:restriction base="xsd:string">
      <xsd:length value="2"/>
      <xsd:enumeration value="AR"/>
      <xsd:enumeration value="LA"/>
      <xsd:enumeration value="MS"/>
      <xsd:enumeration value="OK"/>
      <xsd:enumeration value="TX"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="OrderType">
    <xsd:sequence>
      <xsd:element name="Product" type="ProductType" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="SubTotal">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="ItemsSold" type="xsd:positiveInteger"/>
  </xsd:complexType>

  <xsd:complexType name="ProductType">
    <xsd:attribute name="Name" type="xsd:string"/>
    <xsd:attribute name="Id" type="xsd:positiveInteger"/>
    <xsd:attribute name="Price">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="Quantity" type="xsd:positiveInteger"/>
  </xsd:complexType>

</xsd:schema>
```

Examining the preceding XML schema, you can see that defining a schema for a document can become fairly complicated. However, for all the extra complexity involved, the schema gives the author virtually limitless control over how an XML document can

be validated against it. For instance, you may notice the use of the <xsd:choice> element. You'll learn later in the "Model Groups" section of this paper that this element can be used to indicate when one of a group of elements or attributes may show up, but not all, as is the case with the DeliveryDate and BillingDate attributes.

Also, notice the use of the xsd namespace. This namespace can be anything, but for convention, in this paper, we'll use xsd to indicate an XML Schema Definition Language element.

Creating XML Schemas

One of the first things that comes to mind for most people when authoring an XML schema, is the level of complexity that accompanies it. However, the example in Listing 3 demonstrates only a small portion of the power and flexibility within the XML Schema Definition Language. Table 1 below shows a complete listing list of every element that the XML Schema Definition Language supports.

Table 1: XML Schema Elements Supported by the W3C Standard.

Element Name	Description
All	Indicates that the contained elements may appear in any order within a parent element.
Any	Indicates that any element within the specified namespace may appear within the parent element's definition. If a type is not specifically declared, this is the default.
anyAttribute	Indicates that any attribute within the specified namespace may appear within the parent element's definition.
annotation	Indicates an annotation to the schema.
Appinfo	Indicates information that can be used by an application.
Attribute	Declares an occurrence of an attribute.
attributeGroup	Defines a group of attributes that can be included within a parent element.
Choice	Indicates that only one contained element or attribute may appear within a parent element.
complexContent	Defines restrictions and/or extensions to a complexType.
complexType	Defines a complex element's construction.
documentation	Indicates information to be read by an individual.

Element	Declares an occurrence of an element.
Extension	Extends the contents of an element
Field	Indicates a constraint for an element using XPath.
Group	Logically groups a set of elements to be included together within another element definition.
import	Identifies a namespace whose schema elements and attributes can be referenced within the current schema.
include	Indicates that the specified schema should be included in the target namespace.
Key	Indicates that an attribute or element value is a key within the specified scope.
keyref	Indicates that an attribute or element value should correspond with those of the specified key or unique element.
List	Defines a simpleType element as a list of values of a specified data type.
notation	Contains a notation definition.
redefine	Indicates that simple and complex types, as well as groups and attribute groups from an external schema, can be redefined.
restriction	Defines a constraint for the specified element.
schema	Contains the schema definition.
selector	Specifies an XPath expression that selects a set of elements for an identity constraint.
sequence	Indicates that the elements within the specified group must appear in the exact order they appear within the schema.
simpleContent	Defines restrictions and/or extensions of a simpleType element.
simpleType	Defines a simple element type.
Union	Defines a simpleType element as a collection of values from specified simple data types.
unique	Indicates that an attribute or element value must be unique

	within the specified scope.

Each of the elements in Table 1 has its own series of attributes and elements, including a series of constraints that can be placed on each element.

Authoring an XML schema consists of declaring elements and attributes as well as the "properties" of those elements and attributes. We will begin our look at authoring XML schemas by working our way from the least-complex example to the most complex example. Because attributes may not contain other attributes or elements, we will start there.

Declaring Attributes

Attributes in an XML document are contained by elements. To indicate that a complex element has an attribute, use the <attribute> element of the XML Schema Definition Language. For instance, if we you look at the following section from the PurchaseOrder schema, we you can see the basics for declaring an attribute:

```
<xsd:complexType name="ProductType">
  <xsd:attribute name="Name" type="xsd:string"/>
  <xsd:attribute name="Id" type="xsd:positiveInteger"/>
  <xsd:attribute name="Price">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="Quantity" type="xsd:positiveInteger"/>
</xsd:complexType>
```

From this, we you can see that when declaring an attribute, you must specify a type. This type must be one of the simple types defined in Table 2.

Table 2: The Simple XML Data Types.

Data Type	Description
anyURI	Represents a Uniform Resource Identifier (URI).
base64Binary	Represents Base64-encoded binary data.
boolean	Represents Boolean values (True and False).
byte	Represents an integer ranging from –128 to 127. This type is derived from short.
date	Represents a date.

dateTime	Represents a specific time on a specific date.
decimal	Represents a variable-precision number.
double	Represents a double-precision, 64-bit, floating-point number.
duration	Represents a duration of time.
ENTITIES	Represents a set of values of the ENTITY type.
ENTITY	Represents the ENTITY attribute type in XML 1.0. This type is derived from NCName.
float	Represents a single-precision, 32-bit, floating-point number.
gDay	Represents a recurring Gregorian day of the month.
gMonth	Represents a Gregorian month.
gMonthDay	Represents a recurring Gregorian date.
gYear	Represents a Gregorian year.
gYearMonth	Represents a specific Gregorian month in a specific Gregorian year.
hexBinary	Represents hex-encoded binary data.
ID	Represents the ID attribute type defined in XML 1.0. This type is derived from NCName.
IDREF	Represents a reference to an element with the specified ID attribute value. This type is derived from NCName.
IDREFS	Represents a set of values of IDREF attribute types.
int	Represents an integer with a range of –2,147,483,648 to 2,147,483,647. This type is derived from long.
integer	Represents a sequence of decimal digits with an optional leading sign (+ or -). This type is derived from decimal.
language	Represents natural language identifiers. This type is derived from token.

long	Represents an integer with a range of –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. This type is derived from integer.
Name	Represents a token that begins with a letter, underscore, or colon and continues with letters, digits, and other characters. This type is derived from token.
NCName	Represents "non-colonized" names. This type is derived from Name.
negativeInteger	Represents an integer that is less than zero. This type is derived from nonPositiveInteger.
NMTOKEN	Represents a set of letters, digits, and other characters in any combination with no restriction on the starting character. This type is derived from token.
NMTOKENS	Represents a set of values of NMTOKEN types.
nonNegativeInteger	Represents an integer that is greater than or equal to zero. This type is derived from integer.
nonPositiveInteger	Represents an integer that is less than or equal to zero. This type is derived from integer.
normalizedString	Represents whitespace-normalized strings. This type is derived from string.
NOTATION	Represents a set of QNames.
positiveInteger	Represents an integer that is greater than zero. This type is derived from nonNegativeInteger.
QName	Represents a qualified name.
short	Represents an integer with a value range of –32,768 to 32,767. This type is derived from int.
string	Represents a character string.
time	Represents a recurring instance of time.
token	Represents tokenized strings. This type is derived from normalizedString.
unsignedBtye	Represents an integer with a value range of 0 to 255. This

	type is derived from unsignedShort.
unsignedInt	Represents an integer with a value in the range of 0 to 4,294,967,295. This type is derived from unsignedLong.
unsignedLong	Represents an integer with a value in the range of 0 to 18,446,744,073,709,551,615. This type is derived from nonNegativeInteger.
unsignedShort	Represents an integer with a value in the range of 0 to 65,535. This type is derived from unsignedInt.

The types in Table 2 can each type can be further categorized as either a "primitive" data type or a "derived" data type. Here's a list of the primitive data types:

- anyURI
- base64Binary
- Boolean
- date
- dateTime
- decimal
- double
- duration
- float
- gDay
- gMonth
- gMonthDay
- gYear
- gYearMonth
- hexBinary
- NOTATION

- QName
- string
- time

The "derived" data types are "primitive" or other "derived" data types with restrictions placed on them, such as integer, positiveInteger and byte. Here's a list of the "derived" data types:

- byte
- ENTITIES
- ENTITY
- ID
- IDREF
- IDREFS
- int
- integer
- language
- long
- Name
- NCName
- negativeInteger
- NMTOKEN
- NMTOKENS
- nonNegativeInteger
- nonPositiveInteger
- short
- token
- unsignedByte

- unsignedInt
- unsignedLong
- unsignedShort

Aside from defining the type of an attribute, the <attribute> element within the XML Schema Definition Language contains attributes to assist in defining when an attribute is optional, whether its values is fixed, what its default value is, and so on. Here's the basic syntax for the <attribute> element:

```
<attribute name="" type="" [use=""] [fixed=""] [default=""][value=""]
[ref=""]/>
```

The use attribute can contain one of the following possible values:

- optional
- prohibited
- required

If the use attribute is set to required, the parent element must have the attribute; otherwise, the document will be considered invalid. A value of optional indicates the attribute may or may not occur in the document and the attribute may contain any value. By assigning a value of prohibited to the use attribute, you can indicate that the attribute may not appear at all within the parent element.

Specifying a value for the default attribute indicates that if the attribute does not appear within the specified element of the XML document, it is assumed to have the value. A value within the fixed attribute indicates the attribute has a constant value.

The ref attribute for the <attribute> element indicates that the attribute declaration exists somewhere else within the schema. This allows complex attribute declarations to be defined once and referenced when necessary. For instance, let's say you've "inherited" elements and attributes from another schema and would like to simply reuse one of the attribute declarations within the current schema; this would provide the perfect opportunity to take advantage of the ref attribute.

Just as attributes can be defined based on the simple data types included in the XML Schema Definition Language, they can also be defined based on <simpleType> elements. This can easily be accomplished by declaring an attribute that contains a <simpleType> element, as the following example demonstrates:

```
<xsd:attribute name="exampleattribute">
  <xsd:simpleType base="string">
    <xsd:length value="2"/>
  </xsd:simpleType>
</xsd:attribute>
```

```
<xsd:complexType name="exampleelement">
  <xsd:attribute ref="exampleattribute"/>
</xsd:complexType>
```

From this example, you can see that the XML Schema Definition Language gives the schema author a great deal of control over how attributes are validated. One of the wonderful side-effects of the XML Schema Definition Language is the similarity to object-oriented programming. Consider each attribute definition and element definition to be a class definition. These class definitions describe complex structures and behaviors among various different classes, so each individual class definition, whether it's a simple class or complex class, encapsulates everything necessary to perform its job. The same holds true for the declaration of attributes and elements within an XML document. Each item completely describes itself.

Declaring Elements

Elements within an XML schema can be declared using the <element> element from the XML Schema Definition Language. If you look at the following example from Listing 3, you can see a simple element declaration using the XML Schema Definition Language:

```
<xsd:element name='PurchaseOrder' type='PurchaseOrderType'/>

<xsd:complexType name="PurchaseOrderType">
  <xsd:all>
    <xsd:element name="ShippingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="BillingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="Order" type="OrderType" minOccurs="1"
maxOccurs="1"/>
  </xsd:all>
  <xsd:attribute name="Tax">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="Total">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
```

From this sample, you can see that an element's type may be defined elsewhere within the schema. The location at which an element is defined determines certain characteristics about its availability within the schema. For instance, an element defined as a child of the <schema> element can be referenced anywhere within the schema

document, while an element that is defined when it is declared can only have that definition used once. An element's type can be defined with either a <complexType> element, a <simpleType> element, a <complexContent> element, or a <simpleContent> element. The validation requirements for the document will influence the choice for an element's type. For instance, going back our object-oriented analogy, let's say you define a high-level abstract class and then need to refine its definition for certain situations. In that case, you would create a new class based on the existing one and change its definition as needed. The <complexContent> and <simpleContent> elements work much the same way: They provide a way to extend or restrict the existing simple or complex type definition as needed by the specific instance of the element declaration.

The basic construction of an element declaration using the <element> element within the XML Schema Definition Language is as follows:

```
<element name="" [type=""] [abstract=""] [block=""] [default=""] [final=""]
[fixed=""] [minOccurs=""] [maxOccurs=""] [nillable=""] [ref=""]
[substitutionGroup=""]/>
```

From this, you can see that element declarations offer a myriad of possibilities to the author. For instance, the abstract attribute indicates whether the element being declared may show up directly within the XML document. If this attribute is true, the declared element may not show up directly. Instead, this element must be referenced by another element using the substitutionGroup attribute. This substitution works only if the element utilizing the substitutionGroup attribute occurs directly beneath the <schema> element. In other words, for one element declaration to be substituted for another, the element using the substitutionGroup attribute must be a top-level element. Why would anyone in their right mind declare an element as abstract? The answer is really quite simple. Let's say you need to have multiple elements that have the same basic values specified for the attributes on the <element> element. A <complexType> element definition does not allow for those attributes. So, rather than define and set those attribute values for each element, you could make an "abstract" element declaration, set the values once, and substitute the abstract element definition as needed.

The type attribute indicates that the element should be based on a complexType, simpleType, complexContent, or simpleContent element definition. By defining an element's structure using one of these other elements, the author can gain an incredible amount of control over the element's definition. We will cover these various element definitions in the Declaring Complex Elements section and the Declaring Simple Types section later in this paper.

The block attribute prevents any element with the specified derivation type from being used in place of the element. The block attribute may contain any of the following values:

- #all
- extension

- restriction
- substitution

If the value #all is specified within the block attribute, no elements derived from this element declaration may appear in place of this element. A value of extension prevents any element whose definition has been derived by extension from appearing in place on this element. If a value of restriction is assigned, an element derived by restriction from this element declaration is prevented from appearing in place of this element. Finally, a value of substitution indicates that an element derived through substitution cannot be used in place of this element.

The default attribute may only be specified for an element based on a simpleType or whose content is text only. This attribute assigns a default value to an element.

The minOccurs and maxOccurs attributes specify the minimum and maximum number of times this element may appear within a valid XML document. Although you may explicitly set these attributes, they are not required. To indicate that an element's appearance within the parent element is optional, set the minOccurs attribute to 0. To indicate that the element may occur an unlimited number of times within the parent element, set the maxOccurs attribute to the string "unbounded".

The nillable attribute indicates whether an explicit null value can be assigned to the element. If this particular attribute is omitted, it is assumed to be false. If this attribute has a value of true, the nil attribute for the element will be true. So what exactly does this do for you, this nillable attribute? Well let's say you are writing an application that uses a database that supports NULL values for fields and you are representing your data as XML. Now let's say you request the data from your database and convert it into some XML grammar. How do you tell the difference between those elements that are empty and those elements that are NULL? That's where the nillable attribute comes into play. By appending an attribute of nil to the element, you can tell if it is empty or if it's actually NULL.

The fixed attribute specifies that the element has a constant, predetermined value. This attribute only applies to those elements whose type definitions are based on simpleType or whose content is text only.

Defining Complex Elements

Many times within an XML document, an element may contain child elements and/or attributes. To indicate this within the XML Schema Definition Language, you'll use the <complexType> element. If you examine the following sample section from Listing 3, you'll see the basics used to define a complex element within an XML schema:

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:all>
    <xsd:element name="ShippingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
```

```
    <xsd:element name="BillingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="Order" type="OrderType" minOccurs="1"
maxOccurs="1"/>
  </xsd:all>
  <xsd:attribute name="Tax">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="Total">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
```

The preceding sample section specifies the definition of PurchaseOrderType. This particular element contains three child elements: ShippingInformation, BillingInformation, and Order as well as two attributes: Tax and Total. You should also notice the use of the maxOccurs and minOccurs attributes on the element declarations. With a value of 1 indicated for both attributes, the element declarations specify that they must occur one time within the PurchaseOrderType element.

The basic syntax for the <complexType> element is as follows:

```
<xsd:complexType name=" [abstract="] [base="] [block="] [final="]
[mixed="]/>
```

The abstract attribute indicates whether an element may define its content directly from this type definition or it must define its content from a type derived from this type definition. If this attribute is true, an element must define its content from a derived type definition. If this attribute is omitted or its value is false, an element may define its content directly based on this type definition.

The base attribute specifies the data type for the element. This attribute may hold any value from the included simple XML data types listed in Table 2.

The block attribute indicates what types of derivation are prevented for this element definition. This attribute can contain any of the following values:

- #all
- extension
- restriction

A value of #all prevents all complex types derived from this type definition from being used in place of this type definition. A value of extension prevents complex type definitions derived through extension from being used in place of this type definition. Assigning a value of restriction prevents a complex type definition derived through restriction from being used in place of this type definition. If this attribute is omitted, any type definition derived from this type definition may be used in place of this type definition.

The mixed attribute indicates whether character data is permitted to appear between the child elements of this type definition. If this attribute is false or is omitted, no character may appear. If the type definition contains a simpleContent type element, this value must be false. If the complexContent element appears as a child element, the mixed attribute on the complexContent element can override the value specified in the current type definition.

A <complexType> element in the XML Schema Definition Language may contain only one of the following elements:

- all
- choice
- complexContent
- group
- sequence
- simpleContent

For a short description of these elements, refer back to Table 1.

Defining Simple Types

Sometimes, it's not necessary to declare a complex element type within an XML schema. In these cases, you can use the <simpleType> element of the XML Schema Definition Language. These element type definitions support an element based on the simple XML data types listed in Table 2 or any simpleType declaration within the current schema. For example, let's take the following section from the PurchaseOrder schema in Listing 3:

```
<xsd:simpleType name="PaymentMethodType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Check"/>
    <xsd:enumeration value="Cash"/>
    <xsd:enumeration value="Credit Card"/>
    <xsd:enumeration value="Debit Card"/>
    <xsd:enumeration value="Other"/>
  </xsd:restriction>
</xsd:simpleType>
```

This type definition defines the PaymentMethodType element definition, which is based on the string data type included in the XML Schema Definition Language. You may notice the use of the <enumeration> element. This particular element is referred to as a *facet*, which we'll cover in the next section in this paper.

The basic syntax for defining a simpleType element definition is as follows:

```
<xsd:simpleType name="">
  <xsd:restriction base="/>
</xsd:simpleType>
```

The base attribute type may contain any simple XML data type listed in Table 2 or any simpleType declared within the schema. Specifying the value of this attribute determines the type of data it may contain. A simpleType may only contain a value; not other elements or attributes.

You may also notice the inclusion of the <restriction> element. This is probably the most common method in which to declare types and it helps to set more stringent boundaries on the values an element or attribute based on this type definition may hold. So, to indicate that a type definition's value may hold only string values, you would declare a type definition like the following:

```
<xsd:simpleType name='mySimpleType'>
  <xsd:restriction base='xsd:string'/>
</xsd:simpleType>
```

Two other methods are available to an XML schema author to "refine" a simple type definition: <list> and <union>. The <list> element allows an element or attribute based on the type definition to contain a list of values of a specified simple data type. The <union> element allows you to combine two or more simple type definitions to create a collection of values.

Refining Simple Types Using Facets

To give greater control over the definition of elements and attributes, the W3C added facets to the XML Schema Definition Language. A facet can only be specified for a <simpleType> element, and it helps determine the set of values for a <simpleType> element. For example, a facet may help determine the length or size that an element based off the simpleType may have, an enumeration of acceptable values, and so on. Here's a list of the facets included within the XML Schema Definition Language:

- enumeration
- fractionDigits
- length
- maxExclusive
- maxInclusive

- maxLength
- minExclusive
- minInclusive
- minLength
- pattern
- totalDigits
- whiteSpace

The <enumeration> facet constrains the data type to the specified values. For each valid value for a data type, another <enumeration> element must be defined. The following sample section from Listing 3 demonstrates the use of the <enumeration> facet:

```
<xsd:simpleType name="PaymentMethodType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Check"/>
    <xsd:enumeration value="Cash"/>
    <xsd:enumeration value="Credit Card"/>
    <xsd:enumeration value="Debit Card"/>
    <xsd:enumeration value="Other"/>
  </xsd:restriction>
</xsd:simpleType>
```

This example indicates that the only valid values for an element based on PaymentMethodType are the following:

- Check
- Cash
- Credit Card
- Debit Card
- Other

The <fractionDigits> facet specifies the maximum number of decimal digits in the fractional part. The value for this facet must be a nonNegativeInteger. This may sound a bit confusing, but <fractionDigits> determines the number of decimal places allowed to appear within the value for the data type. For example, look at the following attribute declaration from Listing 3:

```
<xsd:attribute name="SubTotal">
  <xsd:simpleType>
```

```
    <xsd:restriction base="xsd:decimal">
      <xsd:fractionDigits value="2"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
```

The <length> facet determines the number of units of length for the specified data type. For instance, let's examine the following sample section from the PurchaseOrder schema in Listing 3:

```
<xsd:simpleType name="StateType">
  <xsd:restriction base="xsd:string">
    <xsd:length value="2"/>
    <xsd:enumeration value="AR"/>
    <xsd:enumeration value="LA"/>
    <xsd:enumeration value="MS"/>
    <xsd:enumeration value="OK"/>
    <xsd:enumeration value="TX"/>
  </xsd:restriction>
</xsd:simpleType>
```

The preceding sample section indicates that elements derived from the StateType type definition have a string value of "2" (that is, two spaces in length). Furthermore, the only acceptable values for elements derived from the StateType type definition are TX, LA, MS, OK, and AR, as indicated by the <enumeration> elements.

The <maxExclusive> facet specifies the upper bound for the values that can be assigned to the element or attribute. This facet ensures that all values are less than the value specified in this facet. The <maxInclusive> facet specifies the maximum value that can be assigned to the element or attribute.

The <maxLength> and <minLength> facets specify the maximum and minimum lengths for values in the type definition. Keep in mind that the values specified in these facets are units of length that depend on the data type of the type definition's value. Also, remember these facets must have a nonNegativeInteger value assigned to them.

The <minExclusive> facet specifies the lower bound for the values that can be assigned to the element or attribute. This facet ensures that all values are greater than the value specified in this facet. The <minInclusive> facet specifies the minimum value that can be assigned to the element or attribute.

The <pattern> facet applies a specific pattern that the type definition's value must match. This facet constrains the type definition's data type to literals, which must match the pattern specified. Furthermore, the value specified for a <pattern> facet must be a regular expression. So what exactly qualifies as a regular expression? Put simply, a regular expression is composed of one or more "atoms" and optional quantifiers combined together with the pipe character (|). For instance, let's examine the following sample section from Listing 3:

```
<xsd:simpleType name="ZipType">
```

```
  <xsd:restriction base="xsd:string">
    <xsd:minLength value="5"/>
    <xsd:maxLength value="10"/>
    <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
  </xsd:restriction>
</xsd:simpleType>
```

The preceding type definition is for a zip code. In this particular definition, we declare that a valid zip code may only contain numbers 0 through 9; nothing else is allowed. Furthermore, the value for the type definition may contain five numbers listed together; then, if the additional four digits are included, the whole value is separated by a hyphen (-) between the fifth and sixth digits.

The <totalDigits> facet specifies the maximum number of digits for a type definition's value. This value must be a positiveInteger value.

The <whiteSpace> facet specifies how whitespace is treated for the type definition's value. This particular facet can hold one of three values:

- collapse
- preserve
- replace

Specifying collapse indicates that all whitespace consisting of more than a single space will be converted to a single space and that all leading and trailing blanks will be removed. A value of preserve leaves the value as is. Assigning a value of replace causes all tabs, line feeds, and carriage returns to be replaced with a single space.

Not all type definitions, however, support every facet. The type definition's data type determines which facets are available. Table 3 shows which data types will support which facets.

Table 3: The Simple XML Data Types and Applicable Facets.

Type	Applicable Facets
anyURI	enumeration, length, maxLength, minLength, pattern, whiteSpace
base64Binary	enumeration, length, maxLength, minLength, pattern, whiteSpace
boolean	pattern, whiteSpace
byte	enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace

date	enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, whiteSpace
dateTime	enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, whiteSpace
decimal	enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace
double	enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, whiteSpace
duration	enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, whiteSpace
ENTITIES	enumeration, length, maxLength, minLength, whiteSpace
ENTITY	enumeration, length, maxLength, minLength, pattern, whiteSpace
float	enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, whiteSpace
gDay	enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, whiteSpace
gMonth	enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, whiteSpace
gMonthDay	enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, whiteSpace
gYear	enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, whiteSpace
gYearMonth	enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, whiteSpace
hexBinary	enumeration, length, maxLength, minLength, pattern, whiteSpace
ID	enumeration, length, maxLength, minLength, pattern, whiteSpace
IDREF	enumeration, length, maxLength, minLength, pattern, whiteSpace

IDREFS	enumeration, length, maxLength, minLength, whiteSpace
int	enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace
integer	enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace
language	enumeration, length, maxLength, minLength, pattern, whiteSpace
long	enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace
Name	enumeration, length, maxLength, minLength, pattern, whiteSpace
NCName	enumeration, length, maxLength, minLength, pattern, whiteSpace
negativeInteger	enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace
NMTOKEN	enumeration, length, maxLength, minLength, pattern, whiteSpace
NMTOKENS	enumeration, length, maxLength, minLength, whiteSpace
nonNegativeInteger	enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace
nonPositiveInteger	enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace
normalizedString	enumeration, length, maxLength, minLength, pattern, whiteSpace
NOTATION	enumeration, length, maxLength, minLength, pattern, whiteSpace
positiveInteger	enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace
QName	enumeration, length, maxLength, minLength, pattern, whiteSpace
short	enumeration, fractionDigits, maxExclusive, maxInclusive,

	minExclusive, minInclusive, pattern, totalDigits, whiteSpace
string	enumeration, length, maxLength, minLength, pattern, whiteSpace
time	enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, whiteSpace
token	enumeration, length, maxLength, minLength, pattern, whiteSpace
unsignedBtye	enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace
unsignedInt	enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace
unsignedLong	enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace
unsignedShort	enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace

Anonymous Type Declarations

Sometimes within an XML schema it may not be necessary to create a separate type definition for an element or attribute. In such cases, you may use "anonymous" type declarations. Let's pull another sample section from the PurchaseOrder schema in Listing 3 and examine it:

```
<xsd:complexType name="InfoType">
  <xsd:sequence>
    <xsd:element name="Name" minOccurs="1" maxOccurs="1">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string"/>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="Address" type="AddressType" minOccurs="1"
maxOccurs="1"/>
    <xsd:choice minOccurs="1" maxOccurs="1">
      <xsd:group ref="BillingInfoGroup"/>
      <xsd:group ref="ShippingInfoGroup"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
```

This section defines the type definition for InfoType. If you look closely, you'll see the declaration of a <Name> element that does not have a type attribute specified. Instead, the <element> element, itself, contains a <simpleType> element without a name

attribute specified. This is known as an "anonymous" type definition. If you're only using this type definition once, there is no need to go through the trouble of declaring and naming it. However, anonymous type declarations are not limited to <simpleType> elements; you can also create an anonymous type definition for a <complexType> element. For instance, let's look at the following example from Listing 3:

```
<xsd:complexType name="OrderType">
  <xsd:sequence>
    <xsd:element name="Product" type="ProductType" minOccurs="1"
maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="SubTotal">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="ItemsSold" type="xsd:positiveInteger"/>
</xsd:complexType>

<xsd:complexType name="ProductType">
  <xsd:attribute name="Name" type="xsd:string"/>
  <xsd:attribute name="Id" type="xsd:positiveInteger"/>
  <xsd:attribute name="Price">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="Quantity" type="xsd:positiveInteger"/>
</xsd:complexType>
```

This example shows the type definition for elements based on OrderType. This type definition contains an element named Product, which is based on the ProductType type definition. Because we only reference the ProductType type definition once, this would be a good candidate for which to use an anonymous type definition. Using an anonymous type definition, the preceding example changes to the following:

```
<xsd:complexType name="OrderType">
  <xsd:sequence>
    <xsd:element name="Product" minOccurs="1" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="Name" type="xsd:string"/>
        <xsd:attribute name="Id" type="xsd:positiveInteger"/>
        <xsd:attribute name="Price">
          <xsd:simpleType>
            <xsd:restriction base="xsd:decimal">
              <xsd:fractionDigits value="2"/>
            </xsd:restriction>
```

```
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="Quantity" type="xsd:positiveInteger"/>
    </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
 <xsd:attribute name="SubTotal">
  <xsd:simpleType>
   <xsd:restriction base="xsd:decimal">
    <xsd:fractionDigits value="2"/>
   </xsd:restriction>
  </xsd:simpleType>
 </xsd:attribute>
 <xsd:attribute name="ItemsSold" type="xsd:positiveInteger"/>
</xsd:complexType>
```

You can see from this example that the only real change necessary was to move the <complexType> element for the ProductType type definition to be contained by the <element> declaration for Product.

Specifying Mixed Content for Elements

So far we have declared a variety of different elements in Listing 3. Some of the elements have text only, some contain elements only, and some contain elements and attributes. However, we have not specified, yet, how to mix the content of elements that is, mix text with child elements. One of the most overlooked attributes of the <complexType> element is the mixed attribute. If this attribute is set to true, elements based on this type definition can mix their contents with both text and child elements. For instance, let's examine the following sample XML document:

```
<Letter>
  <Greeting>Dear Mr.<Name>John Smith</Name>.</Greeting>
Your order of <Quantity>1</Quantity> <Product>Big Screen TV</Product> has
been shipped.
</Letter>
```

Notice the appearance of text among the child elements of <Letter>. The schema for this XML document would appear as follows:

```
<xsd:element name="Letter">
  <xsd:complexType mixed="true">
    <xsd:element name="Greeting">
      <xsd:complexType mixed="true">
        <xsd:element name="Name" type="xsd:string"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="Quantity" type="xsd:postiveInteger"/>
    <xsd:element name="Product" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

So what's the point of having mixed content? Well, if you needed to uniquely identify something within a paragraph or sentence, specifying an element as having mixed content might be useful. For one, you could easily format that one unique element differently than its parent element. You could also perform some special processing of that element within an application. However, unless it is absolutely necessary to use mixed content within an element, it is highly recommended that each element contain either text or other elements: not both as some undesirable side effects may arise. For instance, in the above sample XML document, if you check the value of the text property for the <Greeting> element using the XMLDOM provided by Microsoft, it would contain:

```
Dear Mr.John Smith
```

Annotating Schemas

In a perfect world, everyone would be able to look at our XML schemas and automatically know what everything is and why it shows up in particular places. Although a good self-describing document can accomplish this to some extent with sensible element and attribute names, the truth of the matter is that most people I've met[md]well, all of them actually[md]are not mind readers. What may seem obvious to you may not be obvious to others. For that very reason, it helps to document your schemas. Within the XML Schema Definition Language, this can be accomplished using annotations. The XML Schema Definition Language defines three new elements to add annotations to an XML schema:

- <annotation>
- <appInfo>
- <documentation>

The <annotation> element contains the <appInfo> and <documentation> elements. In other words, you cannot use the <appInfo> and <documentation> elements by themselves they must be contained within an <annotation> element. To see how this works, let's examine the following sample section from Listing 3:

```
<xsd:annotation>
  <xsd:documentation>
    Purchase order schema for an online grocery store.
  </xsd:documentation>
</xsd:annotation>
```

In the preceding example, the <annotation> and <documentation> elements help to identify the purpose of this particular XML schema. In Listing 3, the <annotation> element appears as a child element of the <schema> element. However, the <annotation> element can appear as a child of any elements listed in Table 2, with the exception of the <documentation> and <appInfo> elements. Really, the only difference between the two elements is the target audience. For the <documentation> element, the information it contains is meant to be read by users, whereas the information contained within an <appInfo> element is meant to be read and utilized by applications.

Model Groups

A *model group*, at least in terms of a schema definition, is a logically grouped set of elements. A model group within the XML Schema Definition Language consists of a "compositor" and a list of "particles" (or element declarations). A model group can be constructed using one of the following XML Schema Definition elements:

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

You can declare a group of elements that should be logically associated together by using the <group> element from the XML Schema Definition Language. Here's the basic syntax for the <group> element:

```
<group name="" [maxOccurs=""] [minOccurs=""] [ref=""]>
  .
  .
  .
</group>
```

By default, the maxOccurs and minOccurs attributes are set to 1. The ref attribute is used after you have defined the <group> element and you wish to reference it, as the following example shows:

```
<xsd:group name="exampleGroup">
  <xsd:all>
    <xsd:element name="Element1" type="xsd:string"/>
    <xsd:element name="Element2" type="xsd:string"/>
    <xsd:element name="Element3" type="xsd:string"/>
  </xsd:all>
</xsd:group>

<xsd:element name="ParentElement">
  <xsd:complexType>
    <xsd:group ref="exampleGroup"/>
  </xsd:complexType>
</xsd:element>
```

All Groups

When the order in which child elements appear within their parent element is not important, you may use an <all> element from the XML Schema Definition Language. The <all> element indicates that the elements declared within it may appear in any order within the parent element. For instance, let's examine the InfoType type definition from Listing 3:

```
<xsd:group name="ShippingInfoGroup">
  <xsd:all>
    <xsd:element name="DeliveryDate" type="DateType"/>
```

```
    <xsd:element name="Method" type="DeliveryMethodType"/>
  </xsd:all>
</xsd:group>

<xsd:group name="BillingInfoGroup">
  <xsd:all>
    <xsd:element name="BillingDate" type="DateType"/>
    <xsd:element name="PaymentMethod" type="PaymentMethodType"/>
  </xsd:all>
</xsd:group>

<xsd:complexType name="InfoType">
  <xsd:sequence>
    <xsd:element name="Name" minOccurs="1" maxOccurs="1">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string"/>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="Address" type="AddressType" minOccurs="1"
maxOccurs="1"/>
    <xsd:choice minOccurs="1" maxOccurs="1">
      <xsd:group ref="BillingInfoGroup"/>
      <xsd:group ref="ShippingInfoGroup"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="DateType">
  <xsd:restriction base="xsd:date"/>
</xsd:simpleType>

<xsd:simpleType name="DeliveryMethodType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="USPS"/>
    <xsd:enumeration value="UPS"/>
    <xsd:enumeration value="FedEx"/>
    <xsd:enumeration value="DHL"/>
    <xsd:enumeration value="Other"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="PaymentMethodType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Check"/>
    <xsd:enumeration value="Cash"/>
    <xsd:enumeration value="Credit Card"/>
    <xsd:enumeration value="Debit Card"/>
    <xsd:enumeration value="Other"/>
  </xsd:restriction>
</xsd:simpleType>
```

Notice the occurrence of the <all> elements. In this particular case, either the <DeliveryDate> and <Method> elements may appear in any order or the <BillingDate> and <PaymentMethod> elements may appear in any order.

Choices

Sometimes you might want to declare that any one of a possible group of elements may appear within an element, but not all of them. This is accomplished by using the <choice> element of the XML Schema Definition Language. Let's examine the following sample section from Listing 3:

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:all>
    <xsd:element name="ShippingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="BillingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="Order" type="OrderType" minOccurs="1"
maxOccurs="1"/>
  </xsd:all>
  <xsd:attribute name="Tax">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="Total">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>

<xsd:group name="ShippingInfoGroup">
  <xsd:all>
    <xsd:element name="DeliveryDate" type="DateType"/>
    <xsd:element name="Method" type="DeliveryMethodType"/>
  </xsd:all>
</xsd:group>

<xsd:group name="BillingInfoGroup">
  <xsd:all>
    <xsd:element name="BillingDate" type="DateType"/>
    <xsd:element name="PaymentMethod" type="PaymentMethodType"/>
  </xsd:all>
</xsd:group>

<xsd:complexType name="InfoType">
  <xsd:sequence>
```

```
    <xsd:element name="Name" minOccurs="1" maxOccurs="1">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string"/>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="Address" type="AddressType" minOccurs="1"
maxOccurs="1"/>
    <xsd:choice minOccurs="1" maxOccurs="1">
      <xsd:group ref="BillingInfoGroup"/>
      <xsd:group ref="ShippingInfoGroup"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="DateType">
  <xsd:restriction base="xsd:date"/>
</xsd:simpleType>

<xsd:simpleType name="DeliveryMethodType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="USPS"/>
    <xsd:enumeration value="UPS"/>
    <xsd:enumeration value="FedEx"/>
    <xsd:enumeration value="DHL"/>
    <xsd:enumeration value="Other"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="PaymentMethodType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Check"/>
    <xsd:enumeration value="Cash"/>
    <xsd:enumeration value="Credit Card"/>
    <xsd:enumeration value="Debit Card"/>
    <xsd:enumeration value="Other"/>
  </xsd:restriction>
</xsd:simpleType>
```

In this case, because the information between the <ShippingInformation> and <BillingInformation> elements is so similar, we only want to define that type definition once. However, because the two date elements[md]<DeliveryDate> and <BillingDate>[md]could not appear in both places, we've decided to create a choice: either the <DeliveryDate> element can appear within the element or the <BillingDate> element can appear, but not both. Furthermore, you can specify the minimum and maximum number of times the selected item may appear within the parent element by using the minOccurs and maxOccurs attributes of the <choice> element.

Sequences

The <sequence> element in the XML Schema Definition Language requires the elements contained within it to appear in the same order in the parent element. For instance, let's examine the following sample section from Listing 3:

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:all>
    <xsd:element name="ShippingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="BillingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="Order" type="OrderType" minOccurs="1"
maxOccurs="1"/>
  </xsd:all>
  <xsd:attribute name="Tax">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="Total">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
```

According to this type definition, the <ShippingInformation>, <BillingInformation>, and <Order> elements may appear within the <PurchaseOrder> element in any order. If we want to indicate that the <ShippingInformation> element must appear first, then the <BillingInformation> element, and then the <Order> element, we could do the following:

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:element name="ShippingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="BillingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="Order" type="OrderType" minOccurs="1"
maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="Tax">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="Total">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
```

```
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
```

Attribute Groups

Just as you can logically group a set of elements together using the <group> element within the XML Schema Definition Language, you can create a logical group of attributes to do the same thing. In this case, though, you will need to use the <attributeGroup> element. Here's the basic syntax for the <attributeGroup> element:

```
<attributeGroup [name=""] [ref=""]>
  <attribute …/>
  <attribute …/>
  .
  .
  .
</attributeGroup>
```

Following the preceding syntax, we could define a group of attributes that can be associated with one another, as the following example shows:

```
<xsd:attributeGroup name="exampleGroup">
  <xsd:attribute name="Attr1" type="xsd:string"/>
  <xsd:attribute name="Attr2" type="xsd:positiveInteger"/>
  <xsd:attribute name="Attr3" type="xsd:date"/>
</xsd:attributeGroup>

<xsd:element name="exampleElement">
  <xsd:complexType>
    <xsd:attributeGroup ref="exampleGroup"/>
  </xsd:complexType>
</xsd:element>
```

The preceding example creates a group of attributes named exampleGroup. This group consists of three attributes: Attr1, Attr2, and Attr3. Also, we've defined a complex element named <exampleElement>, which then references the group of attributes. It is the equivalent of the following:

```
<xsd:element name="exampleElement">
  <xsd:complexType>
    <xsd:attribute name="Attr1" type="xsd:string"/>
    <xsd:attribute name="Attr2" type="xsd:positiveInteger"/>
    <xsd:attribute name="Attr3" type="xsd:date"/>
  </xsd:complexType>
</xsd:element>
```

Targeting Namespaces

You can view an XML schema as a collection of type definitions and element declarations targeted for a specific namespace. Namespaces allow us to distinguish element declarations and type definitions of one schema from another. We can assign an intended namespace for an XML schema by using the targetNamespace attribute on the

<schema> element. By assigning a target namespace for the schema, we indicate that an XML document whose elements are declared as belonging to the schema's namespace should be validated against the XML schema. For instance, we could indicate a target namespace for our PurchaseOrder schema as indicated in Listing 4.

Listing 4: PurchaseOrder1.xsd Contains the Schema Definition for PurchaseOrder.xml with a Target Namespace.

```
<xsd:schema targetNamespace="http://www.eps-software.com/poschema"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.eps-
software.com/poschema" elementFormDefault="unqualified"
attributeFormDefault="unqualified">

  <xsd:annotation>
    <xsd:documentation>
      Purchase Order schema for an online grocery store.
    </xsd:documentation>
  </xsd:annotation>

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

  <xsd:complexType name="PurchaseOrderType">
    <xsd:all>
      <xsd:element name="ShippingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="BillingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="Order" type="OrderType" minOccurs="1"
maxOccurs="1"/>
    </xsd:all>
    <xsd:attribute name="Tax">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="Total">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>

  <xsd:group name="ShippingInfoGroup">
    <xsd:all>
      <xsd:element name="DeliveryDate" type="DateType"/>
      <xsd:element name="Method" type="DeliveryMethodType"/>
    </xsd:all>
  </xsd:group>
```

```
  <xsd:group name="BillingInfoGroup">
    <xsd:all>
      <xsd:element name="BillingDate" type="DateType"/>
      <xsd:element name="PaymentMethod" type="PaymentMethodType"/>
    </xsd:all>
  </xsd:group>

  <xsd:complexType name="InfoType">
    <xsd:sequence>
      <xsd:element name="Name" minOccurs="1" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="Address" type="AddressType" minOccurs="1"
maxOccurs="1"/>
      <xsd:choice minOccurs="1" maxOccurs="1">
        <xsd:group ref="BillingInfoGroup"/>
        <xsd:group ref="ShippingInfoGroup"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="DateType">
    <xsd:restriction base="xsd:date"/>
  </xsd:simpleType>

  <xsd:simpleType name="DeliveryMethodType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="USPS"/>
      <xsd:enumeration value="UPS"/>
      <xsd:enumeration value="FedEx"/>
      <xsd:enumeration value="DHL"/>
      <xsd:enumeration value="Other"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="PaymentMethodType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Check"/>
      <xsd:enumeration value="Cash"/>
      <xsd:enumeration value="Credit Card"/>
      <xsd:enumeration value="Debit Card"/>
      <xsd:enumeration value="Other"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="AddressType">
    <xsd:all>
      <xsd:element name="Street" minOccurs="1">
        <xsd:simpleType>
```

```
        <xsd:restriction base="xsd:string"/>
       </xsd:simpleType>
     </xsd:element>
     <xsd:element name="City" minOccurs="1" maxOccurs="1">
       <xsd:simpleType>
         <xsd:restriction base="xsd:string"/>
       </xsd:simpleType>
     </xsd:element>
     <xsd:element name="State" type="StateType" minOccurs="1"
maxOccurs="1"/>
     <xsd:element name="Zip" type="ZipType" minOccurs="1"
maxOccurs="1"/>
   </xsd:all>
  </xsd:complexType>

  <xsd:simpleType name="ZipType">
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="5"/>
      <xsd:maxLength value="10"/>
      <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="StateType">
    <xsd:restriction base="xsd:string">
      <xsd:length value="2"/>
      <xsd:enumeration value="AR"/>
      <xsd:enumeration value="LA"/>
      <xsd:enumeration value="MS"/>
      <xsd:enumeration value="OK"/>
      <xsd:enumeration value="TX"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="OrderType">
    <xsd:sequence>
      <xsd:element name="Product" type="ProductType" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="SubTotal">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="ItemsSold" type="xsd:positiveInteger"/>
  </xsd:complexType>

  <xsd:complexType name="ProductType">
    <xsd:attribute name="Name" type="xsd:string"/>
    <xsd:attribute name="Id" type="xsd:positiveInteger"/>
```

```
    <xsd:attribute name="Price">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="Quantity" type="xsd:positiveInteger"/>
  </xsd:complexType>

</xsd:schema>
```

Now that we've modified our schema file to specify a target namespace, how do we associate the schema with the XML document? This can be accomplished using the http://www.w3.org/2001/XMLSchema-instance namespace and specifying the schema file's location using the <schemaLocation> element defined within the namespace. Typically, this namespace is given the prefix of xsi. We could then change our PurchaseOrder XML document as indicated in Listing 5.

Listing 5: PurchaseOrder1.xml Contains a Sample Purchase Order Based on the PurchaseOrder1 Schema Definition in PurchaseOrder1.xsd.

```
<po:PurchaseOrder xmlns:po="http://www.eps-software.com/poschema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="PurchaseOrder1.xsd" Tax="5.76" Total="75.77">

<ShippingInformation>
    <Method>USPS</Method>
    <DeliveryDate>08/12/2001</DeliveryDate>
    <Name>Dillon Larsen</Name>
    <Address>
      <Street>123 Jones Rd.</Street>
      <City>Houston</City>
      <State>TX</State>
      <Zip>77381</Zip>
    </Address>
  </ShippingInformation>

  <BillingInformation>
    <PaymentMethod>Credit Card</PaymentMethod>
    <BillingDate>08/09/2001</BillingDate>
    <Name>Madi Larsen</Name>
    <Address>
      <Street>123 Jones Rd.</Street>
      <City>Houston</City>
      <State>TX</State>
      <Zip>77381</Zip>
    </Address>
  </BillingInformation>

  <Order SubTotal="70.01" ItemsSold="17">
    <Product Name="Baby Swiss" Id="702890" Price="2.89" Quantity="1"/>
```

```
    <Product Name="Hard Salami" Id="302340" Price="2.34" Quantity="1"/>
    <Product Name="Turkey" Id="905800" Price="5.80" Quantity="1"/>
    <Product Name="Caesar Salad" Id="991687" Price="2.38" Quantity="2"/>
    <Product Name="Chicken Strips" Id="133382" Price="2.50" Quantity="1"/>
    <Product Name="Bread" Id="298678" Price="1.08" Quantity="1"/>
    <Product Name="Rolls" Id="002399" Price="2.24" Quantity="1"/>
    <Product Name="Cereal" Id="066510" Price="2.18" Quantity="1"/>
    <Product Name="Jalapenos" Id="101005" Price="1.97" Quantity="1"/>
    <Product Name="Tuna" Id="000118" Price="0.92" Quantity="3"/>
    <Product Name="Mayonnaise" Id="126860" Price="1.98" Quantity="1"/>
    <Product Name="Top Sirloin" Id="290502" Price="9.97" Quantity="2"/>
    <Product Name="Soup" Id="001254" Price="1.33" Quantity="1"/>
    <Product Name="Granola Bar" Id="026460" Price="2.14" Quantity="2"/>
    <Product Name="Chocolate Milk" Id="024620" Price="1.58" Quantity="2"/>
    <Product Name="Spaghetti" Id="000265" Price="1.98" Quantity="1"/>
    <Product Name="Laundry Detergent" Id="148202" Price="8.82"
Quantity="1"/>
  </Order>

</po:PurchaseOrder>
```

By assigning a namespace to the <PurchaseOrder> element, we associate that element with the global <PurchaseOrder> element declaration within our XML schema. Notice, however, that the <PurchaseOrder> element is the only qualified element. If you look back at our <schema> element from Listing 4, you'll see two attributes: elementFormDefault and attributeFormDefault. These attributes can possess one of two values:

- qualified
- unqualified

If a value of unqualified is specified or the elementFormDefault and attributeFormDefault attributes are omitted, the elements or attributes that are not globally declared within the schema (those that are not children of the <schema> element) do not require a prefix within the XML instance document. However, if a value of qualified is specified, all elements and attributes must have a prefix associated with them. For instance, we could make a change to our PurchaseOrder schema and specify that the elementFormDefault and attributeFormDefault attributes have a value of qualified, as shown in Listing 6.

Listing 6: PurchaseOrder2.xsd Contains the Schema Definition for PurchaseOrder.xml with a Target Namespace and Qualified Elements and Attributes.

```
<xsd:schema targetNamespace="http://www.eps-software.com/poschema"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.eps-
software.com/poschema"
elementFormDefault="qualified" attributeFormDefault="qualified">

  <xsd:annotation>
    <xsd:documentation>
```

```
    Purchase Order schema for an online grocery store.
   </xsd:documentation>
  </xsd:annotation>

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

  <xsd:complexType name="PurchaseOrderType">
   <xsd:all>
    <xsd:element name="ShippingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="BillingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="Order" type="OrderType" minOccurs="1"
maxOccurs="1"/>
   </xsd:all>
   <xsd:attribute name="Tax">
    <xsd:simpleType>
     <xsd:restriction base="xsd:decimal">
      <xsd:fractionDigits value="2"/>
     </xsd:restriction>
    </xsd:simpleType>
   </xsd:attribute>
   <xsd:attribute name="Total">
    <xsd:simpleType>
     <xsd:restriction base="xsd:decimal">
      <xsd:fractionDigits value="2"/>
     </xsd:restriction>
    </xsd:simpleType>
   </xsd:attribute>
  </xsd:complexType>

  <xsd:group name="ShippingInfoGroup">
   <xsd:all>
    <xsd:element name="DeliveryDate" type="DateType"/>
    <xsd:element name="Method" type="DeliveryMethodType"/>
   </xsd:all>
  </xsd:group>

  <xsd:group name="BillingInfoGroup">
   <xsd:all>
    <xsd:element name="BillingDate" type="DateType"/>
    <xsd:element name="PaymentMethod" type="PaymentMethodType"/>
   </xsd:all>
  </xsd:group>

  <xsd:complexType name="InfoType">
   <xsd:sequence>
    <xsd:element name="Name" minOccurs="1" maxOccurs="1">
     <xsd:simpleType>
      <xsd:restriction base="xsd:string"/>
     </xsd:simpleType>
    </xsd:element>
```

```
    <xsd:element name="Address" type="AddressType" minOccurs="1"
maxOccurs="1"/>
    <xsd:choice minOccurs="1" maxOccurs="1">
     <xsd:group ref="BillingInfoGroup"/>
     <xsd:group ref="ShippingInfoGroup"/>
    </xsd:choice>
   </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="DateType">
   <xsd:restriction base="xsd:date"/>
  </xsd:simpleType>

  <xsd:simpleType name="DeliveryMethodType">
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="USPS"/>
    <xsd:enumeration value="UPS"/>
    <xsd:enumeration value="FedEx"/>
    <xsd:enumeration value="DHL"/>
    <xsd:enumeration value="Other"/>
   </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="PaymentMethodType">
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Check"/>
    <xsd:enumeration value="Cash"/>
    <xsd:enumeration value="Credit Card"/>
    <xsd:enumeration value="Debit Card"/>
    <xsd:enumeration value="Other"/>
   </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="AddressType">
   <xsd:all>
    <xsd:element name="Street" minOccurs="1">
     <xsd:simpleType>
      <xsd:restriction base="xsd:string"/>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="City" minOccurs="1" maxOccurs="1">
     <xsd:simpleType>
      <xsd:restriction base="xsd:string"/>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="State" type="StateType" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="Zip" type="ZipType" minOccurs="1"
maxOccurs="1"/>
   </xsd:all>
  </xsd:complexType>
```

```
  <xsd:simpleType name="ZipType">
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="5"/>
      <xsd:maxLength value="10"/>
      <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="StateType">
    <xsd:restriction base="xsd:string">
      <xsd:length value="2"/>
      <xsd:enumeration value="AR"/>
      <xsd:enumeration value="LA"/>
      <xsd:enumeration value="MS"/>
      <xsd:enumeration value="OK"/>
      <xsd:enumeration value="TX"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="OrderType">
    <xsd:sequence>
      <xsd:element name="Product" type="ProductType" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="SubTotal">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="ItemsSold" type="xsd:positiveInteger"/>
  </xsd:complexType>

  <xsd:complexType name="ProductType">
    <xsd:attribute name="Name" type="xsd:string"/>
    <xsd:attribute name="Id" type="xsd:positiveInteger"/>
    <xsd:attribute name="Price">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="Quantity" type="xsd:positiveInteger"/>
  </xsd:complexType>

</xsd:schema>
```

Based on the PurchaseOrder schema in Listing 6, the new version of the PurchaseOrder XML would appear as shown in Listing 7.

Listing 7: PurchaseOrder2.xml Contains a Sample Purchase Order Based on the PurchaseOrder2 Schema Definition in PurchaseOrder2.xsd.

```
<po:PurchaseOrder xmlns:po="http://www.eps-software.com/poschema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="purchaseorder2.xsd"
  Tax="5.76" Total="75.77">

   <po:ShippingInformation>
      <po:Method>USPS</po:Method>
      <po:DeliveryDate>08/12/2001</po:DeliveryDate>
      <po:Name>Dillon Larsen</po:Name>
      <po:Address>
         <po:Street>123 Jones Rd.</po:Street>
         <po:City>Houston</po:City>
         <po:State>TX</po:State>
         <po:Zip>77381</po:Zip>
      </po:Address>
   </po:ShippingInformation>

   <po:BillingInformation>
      <po:PaymentMethod>Credit Card</po:PaymentMethod>
      <po:BillingDate>08/09/2001</po:BillingDate>
      <po:Name>Madi Larsen</po:Name>
      <po:Address>
         <po:Street>123 Jones Rd.</po:Street>
         <po:City>Houston</po:City>
         <po:State>TX</po:State>
         <po:Zip>77381</po:Zip>
      </po:Address>
   </po:BillingInformation>

   <po:Order SubTotal="70.01" ItemsSold="17">
      <po:Product Name="Baby Swiss" Id="702890" Price="2.89" Quantity="1"/>
      <po:Product Name="Hard Salami" Id="302340" Price="2.34"
Quantity="1"/>
      <po:Product Name="Turkey" Id="905800" Price="5.80" Quantity="1"/>
      <po:Product Name="Caesar Salad" Id="991687" Price="2.38"
Quantity="2"/>
      <po:Product Name="Chicken Strips" Id="133382" Price="2.50"
Quantity="1"/>
      <po:Product Name="Bread" Id="298678" Price="1.08" Quantity="1"/>
      <po:Product Name="Rolls" Id="002399" Price="2.24" Quantity="1"/>
      <po:Product Name="Cereal" Id="066510" Price="2.18" Quantity="1"/>
      <po:Product Name="Jalapenos" Id="101005" Price="1.97" Quantity="1"/>
      <po:Product Name="Tuna" Id="000118" Price="0.92" Quantity="3"/>
      <po:Product Name="Mayonnaise" Id="126860" Price="1.98"
Quantity="1"/>
      <po:Product Name="Top Sirloin" Id="290502" Price="9.97" Quantity="2"/>
      <po:Product Name="Soup" Id="001254" Price="1.33" Quantity="1"/>
      <po:Product Name="Granola Bar" Id="026460" Price="2.14"
Quantity="2"/>
```

```
    <po:Product Name="Chocolate Milk" Id="024620" Price="1.58"
Quantity="2"/>
    <po:Product Name="Spaghetti" Id="000265" Price="1.98" Quantity="1"/>
    <po:Product Name="Laundry Detergent" Id="148202" Price="8.82"
Quantity="1"/>
  </po:Order>

</po:PurchaseOrder>
```

"Inheriting" from Other Schemas

As you can see from the XML schema in Listing 6, things can get rather complex and long. Plus, you may wish, at times, to define a common piece for multiple XML schemas and maintain and extend it separately from the individual schemas that need it. For this reason, the W3C included the <include> and <import> elements in the XML Schema Definition Language. Through the use of these elements, you can effectively "inherit" elements and attributes from the referenced schema. For instance, if you look at Listing 3, you can see the declaration of an <Address> element. We may want to use this same element over and over again in multiple schemas. However, we wouldn't want to redefine this element in each schema. Instead, it would be nice to have that element declaration and type definition within a separate document.

As long as the targetNamespace attribute on the <schema> element of both schemas match, or the targetNamespace attribute for the <schema> element in the referenced XML schema is empty, you can "inherit" any and all elements and attributes within the XML schema using the <include> element. The <import> element doesn't care what the target namespace is in the referenced schema.

Going back Listing 3, we can separate out the <Address> element declaration (and the various type definitions that go along with it) into its own schema, as shown in Listing 8.

Listing 8: Address.xsd Contains a Sample Address Schema Definition.

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

 <xsd:annotation>
  <xsd:documentation>
   Address schema for a typical US address
  </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="Address" type="AddressType"/>

 <xsd:complexType name="AddressType">
  <xsd:all>
   <xsd:element name="Street" minOccurs="1">
    <xsd:simpleType>
     <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
   </xsd:element>
   <xsd:element name="City" minOccurs="1" maxOccurs="1">
```

```
      <xsd:simpleType>
        <xsd:restriction base="xsd:string"/>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="State" type="StateType" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="Zip" type="ZipType" minOccurs="1"
maxOccurs="1"/>
  </xsd:all>
 </xsd:complexType>

 <xsd:simpleType name="ZipType">
  <xsd:restriction base="xsd:string">
    <xsd:minLength value="5"/>
    <xsd:maxLength value="10"/>
    <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
  </xsd:restriction>
 </xsd:simpleType>

 <xsd:simpleType name="StateType">
  <xsd:restriction base="xsd:string">
    <xsd:length value="2"/>
    <xsd:enumeration value="AR"/>
    <xsd:enumeration value="LA"/>
    <xsd:enumeration value="MS"/>
    <xsd:enumeration value="OK"/>
    <xsd:enumeration value="TX"/>
  </xsd:restriction>
 </xsd:simpleType>

</xsd:schema>
```

Notice in the new Address schema that we did not specify a value for the targetNamespace attribute: This will allow us to include the schema in a modified version of the PurchaseOrder schema by using the <include> element as shown in Listing 9.

Listing 9: PurchaseOrder3.xsd Includes the Contents of Address.xsd.

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

 <xsd:include schemaLocation="Address.xsd"/>

 <xsd:annotation>
  <xsd:documentation>
    Purchase Order schema for an online grocery store.
  </xsd:documentation>
 </xsd:annotation>

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

 <xsd:complexType name="PurchaseOrderType">
```

```
    <xsd:all>
      <xsd:element name="ShippingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="BillingInformation" type="InfoType" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="Order" type="OrderType" minOccurs="1"
maxOccurs="1"/>
    </xsd:all>
    <xsd:attribute name="Tax">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="Total">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>

  <xsd:group name="ShippingInfoGroup">
    <xsd:all>
      <xsd:element name="DeliveryDate" type="DateType"/>
      <xsd:element name="Method" type="DeliveryMethodType"/>
    </xsd:all>
  </xsd:group>

  <xsd:group name="BillingInfoGroup">
    <xsd:all>
      <xsd:element name="BillingDate" type="DateType"/>
      <xsd:element name="PaymentMethod" type="PaymentMethodType"/>
    </xsd:all>
  </xsd:group>

  <xsd:complexType name="InfoType">
    <xsd:sequence>
      <xsd:element name="Name" minOccurs="1" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element ref="Address" minOccurs="1" maxOccurs="1"/>
      <xsd:choice>
        <xsd:group ref="BillingInfoGroup"/>
        <xsd:group ref="ShippingInfoGroup"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
```

```
 <xsd:simpleType name="DateType">
   <xsd:restriction base="xsd:date"/>
 </xsd:simpleType>

 <xsd:simpleType name="DeliveryMethodType">
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="USPS"/>
     <xsd:enumeration value="UPS"/>
     <xsd:enumeration value="FedEx"/>
     <xsd:enumeration value="DHL"/>
     <xsd:enumeration value="Other"/>
   </xsd:restriction>
 </xsd:simpleType>

 <xsd:simpleType name="PaymentMethodType">
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="Check"/>
     <xsd:enumeration value="Cash"/>
     <xsd:enumeration value="Credit Card"/>
     <xsd:enumeration value="Debit Card"/>
     <xsd:enumeration value="Other"/>
   </xsd:restriction>
 </xsd:simpleType>

 <xsd:complexType name="OrderType">
   <xsd:sequence>
     <xsd:element name="Product" type="ProductType" minOccurs="1"
maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="SubTotal">
     <xsd:simpleType>
       <xsd:restriction base="xsd:decimal">
         <xsd:fractionDigits value="2"/>
       </xsd:restriction>
     </xsd:simpleType>
   </xsd:attribute>
   <xsd:attribute name="ItemsSold" type="xsd:positiveInteger"/>
 </xsd:complexType>

 <xsd:complexType name="ProductType">
   <xsd:attribute name="Name" type="xsd:string"/>
   <xsd:attribute name="Id" type="xsd:positiveInteger"/>
   <xsd:attribute name="Price">
     <xsd:simpleType>
       <xsd:restriction base="xsd:decimal">
         <xsd:fractionDigits value="2"/>
       </xsd:restriction>
     </xsd:simpleType>
   </xsd:attribute>
   <xsd:attribute name="Quantity" type="xsd:positiveInteger"/>
 </xsd:complexType>
```

```
</xsd:schema>
```

Because we did not explicitly declare a target namespace for the Address schema, we can include it within the new PurchaseOrder schema. Because there is no reference to a namespace, however, we can simply refer to the declared <Address> element in the Address schema without having to qualify it. However, to prevent schemas from getting confused with other <Address> elements from other schemas, we may want to specify a value for the targetNamespace attribute for our Address schema as shown in Listing 10.

Listing 10: Address1.xsd Modified to Specify a Target Namespace.

```
<xsd:schema targetNamespace="http://www.eps-software.com/addressschema"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.eps-
software.com/addressschema">

  <xsd:annotation>
    <xsd:documentation>
      Address schema for a typical US address
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="Address" type="AddressType"/>

  <xsd:complexType name="AddressType">
    <xsd:all>
      <xsd:element name="Street" minOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="City" minOccurs="1" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="State" type="StateType" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="Zip" type="ZipType" minOccurs="1"
maxOccurs="1"/>
    </xsd:all>
  </xsd:complexType>

  <xsd:simpleType name="ZipType">
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="5"/>
      <xsd:maxLength value="10"/>
      <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
    </xsd:restriction>
  </xsd:simpleType>
```

```
  <xsd:simpleType name="StateType">
    <xsd:restriction base="xsd:string">
      <xsd:length value="2"/>
      <xsd:enumeration value="AR"/>
      <xsd:enumeration value="LA"/>
      <xsd:enumeration value="MS"/>
      <xsd:enumeration value="OK"/>
      <xsd:enumeration value="TX"/>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

Because we have just specified a target namespace for the Address schema, unless the target namespace for the PurchaseOrder schema is the same, we can no longer use the <include> element to "inherit" the element declarations from the Address schema. However, we can use the <import> element to include the newly modified Address schema as shown in Listing 11.

Listing 11: PurchaseOrder4.xsd "Imports" the Contents of Address1.xsd.

```
<xsd:schema xmlns:adr="http://www.eps-software.com/addressschema"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:import namespace="http://www.eps-software.com/addressschema"
schemaLocation="Address1.xsd"/>

  <xsd:annotation>
    <xsd:documentation>
      Purchase Order schema for an online grocery store.
    </xsd:documentation>
  </xsd:annotation>

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

  <xsd:complexType name="PurchaseOrderType">
    <xsd:all>
      <xsd:element name="ShippingInformation" type="InfoType"/>
      <xsd:element name="BillingInformation" type="InfoType"/>
      <xsd:element name="Order" type="OrderType"/>
    </xsd:all>
    <xsd:attribute name="Tax">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="Total">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
```

```
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>

<xsd:group name="ShippingInfoGroup">
  <xsd:all>
    <xsd:element name="DeliveryDate" type="DateType"/>
    <xsd:element name="Method" type="DeliveryMethodType"/>
  </xsd:all>
</xsd:group>

<xsd:group name="BillingInfoGroup">
  <xsd:all>
    <xsd:element name="BillingDate" type="DateType"/>
    <xsd:element name="PaymentMethod" type="PaymentMethodType"/>
  </xsd:all>
</xsd:group>

<xsd:complexType name="InfoType">
  <xsd:sequence>
    <xsd:element name="Name">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string"/>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element ref="adr:Address"/>
    <xsd:choice>
      <xsd:group ref="BillingInfoGroup"/>
      <xsd:group ref="ShippingInfoGroup"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="DateType">
  <xsd:restriction base="xsd:date"/>
</xsd:simpleType>

<xsd:simpleType name="DeliveryMethodType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="USPS"/>
    <xsd:enumeration value="UPS"/>
    <xsd:enumeration value="FedEx"/>
    <xsd:enumeration value="DHL"/>
    <xsd:enumeration value="Other"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="PaymentMethodType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Check"/>
    <xsd:enumeration value="Cash"/>
```

```
      <xsd:enumeration value="Credit Card"/>
      <xsd:enumeration value="Debit Card"/>
      <xsd:enumeration value="Other"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="OrderType">
    <xsd:sequence>
      <xsd:element name="Product" type="ProductType"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="SubTotal">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="ItemsSold" type="xsd:positiveInteger"/>
  </xsd:complexType>

  <xsd:complexType name="ProductType">
    <xsd:attribute name="Name" type="xsd:string"/>
    <xsd:attribute name="Id" type="xsd:positiveInteger"/>
    <xsd:attribute name="Price">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="Quantity" type="xsd:positiveInteger"/>
  </xsd:complexType>

</xsd:schema>
```

You can see that the <import> element supports two attributes: namespace and schemaLocation. You'll also notice the declaration of the adr namespace within the <schema> element. This namespace declaration is necessary for the <import> element to work correctly: The namespace attribute on the <import> element refers to a namespace that has been previously declared within the <schema> element.

Summary

An XML schema consists of components, primarily elements, attributes, and type definitions. These components are assembled within an XML schema to indicate whether an XML document conforms to the schema specified. In May 2001, the W3C finalized its recommendation for the XML Schema Definition Language, which provides the individual language elements needed to create an XML schema.

The XML Schema Definition Language provides a very powerful and flexible way in which to validate XML documents. It includes everything from declaring elements and attributes to "inheriting" elements from other schemas, from defining complex element definitions to defining restrictions for even the simplest of data types. This gives the XML schema author such control over specifying a valid construction for an XML document that there is almost nothing that cannot be defined with an XML schema.

DTDs and XML Schemas are two very different means to the same end: providing a "roadmap" with which to validate XML documents. However, so much more detail can be specified with an XML Schema than with a DTD. Schemas support varying data types, namespaces, and allow the author to define the structure of an XML document using XML syntax. DTDs are limited to character data types, provide no support for namespaces, and define the structure of an XML document using a very archaic and cumbersome standard. Since an XML schema is nothing more than an XML document in itself, it can be loaded into an XML parser just like any other XML document which allows applications to provide added functionality with a very common interface through the XMLDOM.