

## XSD for Visual Basic Developers

By [Yasser Shohoud](#)

### Why Schemas?

As you develop distributed Web applications, you will need a way to represent your structured data and send it from one tier to the other possibly over the Web. XML is an excellent choice for representing structured data. For example, you'll find yourself sending XML data between the middle tier and the user interface. The result is a set of middle-tier methods that emit or receive XML strings. You might have a method that receives invoice data like this:

```
Public Function SaveInvoice(sInvoiceXML As String) As Boolean
```

This function receives an XML string that contains the invoice document. The function then validates the invoice data according to your business rules then saves it to the database and returns a Boolean indicating success or failure.

When the UI tier wants to invoke this function, they look at the above function declaration and wonder just what the heck does the invoice XML look like? So you might give them an example invoice XML document and tell them this is what they have to send you. This works for a while, until you change the invoice XML document to accommodate a new feature that you're adding. Now all of a sudden, saving an invoice from the UI does not work and you spend hours troubleshooting the problem before you realize it's a mismatch between what the UI is sending and what you expect.

The problem here is that declaring XML data as just a String means your function must handle validating the contents of that string to make sure that it is XML and that it is a valid invoice document. It becomes very difficult to ensure the UI is sending you the right invoice document unless you write lots of tedious validation code and you maintain that code as the invoice document evolves. XML schema solve this problem by providing a standard language to communicate the invoice document structure to the UI tier and to validate the invoice document that the UI sends to the middle tier before the middle tier starts processing it. Think of the invoice XML schema as an extension to the above function declaration where you specify exactly what the invoice XML looks like.

XML Schema is a W3C Recommendation (a standard) as of May 2, 2001. It is comprised of three specifications: [XML Schema Part 1: structures](#), [XML Schema Part 2: Datatypes](#), and [XML Schema Part 0:Primer](#). The W3C's XML Schema is sometimes referred to as XML Schema Definition language or XSD for short. XSD is an XML-based grammar for describing the structure of XML documents. A schema-aware validating parser, like MSXML 4.0, can validate an XML document against an XSD schema and report any discrepancies.

To solve the problem outlined above, you'd create an XSD schema that describes the invoice document. You'd then make this schema available to the UI tier developers. The schema is now part of the "interface contract" between the middle tier and the UI. While the application is in development, the UI tier can validate the invoice documents that they send against that schema to ensure they are valid. Similarly, the SaveInvoice function can validate the input invoice document against the schema before attempting to process it. Now if you change the invoice document to support a new feature, you must change the schema accordingly. Now the UI team tries to validate the invoice documents they're sending and this validation fails so they immediately realize that the schema has changed and that they must change the invoice documents they are sending. This can also help

Visit [www.LearnXmlWS.com](http://www.LearnXmlWS.com) for more resources

catch version mismatch problems where you have an older client trying to talk to a newer middle tier or vice versa.

## Introduction to XSD

### *Validating with Schema*

Before we dive into the details of creating XSD schemas, let's understand how to use them for validating XML documents. You'll need to get a copy of [MSXML 4.0](#) and install it. Assuming you already have the schema and the XML document, all you need now is a few lines of code to validate. I wrote a simple app that validates an XML document using an XSD schema just to show you how to write the code. [You can download it here](#).

You'll first instantiate an XMLSchema object and add to it the XSD schema file that you already have.

```
Dim schemaCache As MSXML2.XMLSchemaCache40
Set schemaCache = New MSXML2.XMLSchemaCache40
schemaCache.Add "", "D:\schemas\mySchema.xsd"
```

When you call the Add method, the first parameter is the namespace that your XML document uses. If your XML document does not use a namespace, you can just pass an empty string like you see here (To learn more about XML namespaces, see the [XML Namespaces tutorial](#)). The second parameter is the URL or file path pointing to the XSD schema document. Next, you create a DOM document and add to it the schemaCache:

```
Dim doc As MSXML2.DOMDocument40
Set doc = New MSXML2.DOMDocument40
Set doc.schemas = schemaCache
```

Now you are ready to load the XML document and do the validation:

```
doc.async = False
If Not doc.Load("D:\docs\myDoc.xml") Then
    MsgBox "Error loading XML document: " & doc.parseError.reason
End If
```

After loading the document, you must check the parseError property for validation errors. As you can see, validating is quite easy, once you have the XSD schema. Now I'll show you the basics for creating XSD schemas.

### *Authoring XSD Schemas*

An XSD schema starts with a <schema> element like this:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" >
<!-- schema content goes here -->
</schema>
```

The XSD namespace has changed quite a bit as the XSD specification itself has evolved. In the final specification, the XSD namespace is <http://www.w3.org/2001/XMLSchema>. If you come across an XSD schema that's using a different namespace, you'll know it's written to an older (possibly draft) version of the XSD standard.

Within a schema, you define data types and declare elements. For example, assume you have this simple XML document:

```
<top>
  <child1>text</child1>
  <child2>text</child2>
```

Visit [www.LearnXmlWS.com](http://www.LearnXmlWS.com) for more resources

</top>

The schema would first define a type that describes the contents of the <top> element. It would then declare an element of that type:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
  <xsd:complexType name="topType" >
    <xsd:sequence>
      <xsd:element name="child1" type="xsd:string" />
      <xsd:element name="child2" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="top" type="topType" />
</xsd:schema>
```

<xsd:complexType> defines a data type that contains child elements and/or attributes. The <top> element contains child element so it must be a complexType. Within this complex type, we specify that there will be a sequence of child1 and child2 elements. <xsd:sequence> means the elements must appear in that order, so if the XML document had <child2> first then <child1> that would violate the sequence and would be a validation error. Using <xsd:element> we declare each of the <child1> and <child2> elements and specify their data type as xsd:string. Note the use of the xsd: namespace prefix to indicate that this is one of the XSD built-in data types. Now that we have defined the data type that describes our <top> element, we can declare the element itself using <xsd:element>. Note that we specify its type as topType which is the name we used for the complexType.

The basics of creating XSD schemas are: First define your data types then declare elements of those types. Lets take a look at some real-world examples.

## Mapping VB Types to XSD

Say you have an Order class with a bunch of properties as shown in listing 1. One of those properties is a collection of objects of the OrderItem class shown in listing 2.

### Listing 1: The Order class.

```
Order class
'Local variable(s) to hold property value(s)
Private mvarOrderId As Long 'local copy
Private mvarRequiredDate As Date 'local copy
Private mvarShipName As String 'local copy
'This is a collection of OrderDetail objects
Private mvarOrderDetails As Collection 'local copy

Public Property Set OrderDetails(ByVal vData As Collection)
    Set mvarOrderDetails = vData
End Property

Public Property Get OrderDetails() As Collection
    Set OrderDetails = mvarOrderDetails
End Property

Public Property Let ShipName(ByVal vData As String)
    mvarShipName = vData
End Property
Public Property Get ShipName() As String
    ShipName = mvarShipName
End Property
Public Property Let RequiredDate(ByVal vData As Date)
    mvarRequiredDate = vData
End Property
Public Property Get RequiredDate() As Date
    RequiredDate = mvarRequiredDate
End Property
Public Property Let OrderId(ByVal vData As Long)
    mvarOrderId = vData
End Property
Public Property Get OrderId() As Long
    OrderId = mvarOrderId
End Property
```

Visit [www.LearnXmlWS.com](http://www.LearnXmlWS.com) for more resources

**Listing 2: The OrderItem class.**

```
OrderItem class
'Local variable(s) to hold property value(s)
Private mvarProductID As Long 'Local copy
Private mvarUnitPrice As Double 'Local copy
Private mvarQuantity As Integer 'Local copy
Public Property Let Quantity(ByVal vData As Integer)
    mvarQuantity = vData
End Property
Public Property Get Quantity() As Integer
    Quantity = mvarQuantity
End Property
Public Property Let UnitPrice(ByVal vData As Double)
    mvarUnitPrice = vData
End Property
Public Property Get UnitPrice() As Double
    UnitPrice = mvarUnitPrice
End Property
Public Property Let ProductID(ByVal vData As Long)
    mvarProductID = vData
End Property
Public Property Get ProductID() As Long
    ProductID = mvarProductID
End Property
```

You'll need to create a schema for the Order class if you're going to serialize Order objects to XML (possibly as parameter or return values from method calls between tiers or as Web services). There are many ways you could serialize objects of the Order class to XML. You should consider using an element to represent each class property. This provides a consistent serialization technique and works well with Web services. Before we discuss the schema, take a look at the example XML document in listing 4 that represents a serialized Order object.

**Listing 3: Example XML document with an Order object.**

```
<Order>
  <OrderID>2093324</OrderID>
  <RequiredDate>2001-08-01</RequiredDate>
  <ShipName>Yasser Shohoud</ShipName>
  <OrderDetails>
    <OrderItem>
      <ProductID>53034</ProductID>
      <Quantity>9</Quantity>
      <UnitPrice>12.09</UnitPrice>
    </OrderItem>
    <OrderItem>
      <ProductID>66090</ProductID>
      <Quantity>12</Quantity>
      <UnitPrice>10.09</UnitPrice>
    </OrderItem>
    <OrderItem>
      <ProductID>56091</ProductID>
      <Quantity>2</Quantity>
      <UnitPrice>87.95</UnitPrice>
    </OrderItem>
  </OrderDetails>
</Order>
```

Listing 4 shows an example schema for the above document. This schema is the XML-equivalent of the Order and OrderItem class definitions.

**Listing 4: Example schema for Order and OrderItem.**

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified">
  <!-- this is the equivalent of the OrderItem class -->
  <xsd:complexType name="OrderItemType">
    <xsd:sequence>
      <xsd:element name="ProductID" type="xsd:int"/>
      <xsd:element name="Quantity" type="xsd:short"/>
      <xsd:element name="UnitPrice" type="xsd:double"/>
    </xsd:sequence>
  </xsd:complexType>
  <!-- this corresponds to the Order class -->
  <xsd:complexType name="OrderType">
    <xsd:sequence>
      <xsd:element name="OrderID" type="xsd:int"/>
      <xsd:element name="RequiredDate" type="xsd:date"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Visit [www.LearnXmlWS.com](http://www.LearnXmlWS.com) for more resources

```
<xsd:element name="ShipName" type="xsd:string"/>
<xsd:element name="OrderDetails" type="OrderDetailsType"/>
<!-- the following is the equivalent of the OrderDetails collection -->
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="OrderDetailsType">
  <xsd:sequence>
    <xsd:element name="OrderItem" type="OrderItemType" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<!-- this corresponds to the Order class -->
<xsd:element name="Order" type="OrderType"/>
</xsd:schema>
```

First, there's a complex type called OrderItemType, which corresponds to the OrderItem class. Within this type, there is a sequence of three elements, each corresponds to a property of the OrderItem class. Here we have to map the intrinsic VB types such as Long, Integer, and String to the corresponding XSD types. The [WSDL tutorial](#) on this site has a table that shows the mapping between XSD and VB types.

The next step is to define the OrderType. There's again a sequence of elements that correspond to the properties of the Order class. For example, the RequiredDate property becomes an element called RequiredDate with the type xsd:date. The OrderDetails property is a collection of OrderItem objects. To define this, you create an element called OrderDetails. This element corresponds to the collection itself. To actually describe the type of objects in this collection, you need to specify that the OrderDetails element contains one or more OrderItem elements each of type OrderItemType (defined above). As you see in listing 3, this is done by creating a type called OrderDetailsType which contains a sequence of one or more OrderItem elements. To specify that there can be any number of OrderItem elements (any number of objects in the collection) we set the maxOccurs attribute to "unbounded". There's also a minOccurs attribute, which can specify the minimum number of times an element may occur. The default for both maxOccurs and minOccurs is 1, so if you don't specify either of them it means the element appears exactly once.

Now that we have defined the OrderType, we are ready to declare the main element of our document: The Order. The Order element is declared as a global element, that is, it is not part of any type itself because it is the top-most element in the document, also known as the document element.

This is just a simple example of how you can map VB classes (or User-Defined Types) to XSD schemas. Once you've done a few of these, you'll get the hang of it. There's a lot more you can do in XSD to make your validation tighter, let's take a look at some of that.

## Tighter Validation

In the example above, we defined the shipName field as xsd:string. In many cases, you'll want to be more specific than that. For example, you might want to specify that the maximum length is 40 characters. To do this, you define a new type that inherits from the built-in string type and adds a restriction on the length. You do this using the <xsd:simpleType> element:

```
<xsd:simpleType name="shipNameType">
  <xsd:restriction base="xsd:string">
    <xsd:maxLength value="40"/>
  </xsd:restriction>
</xsd:simpleType>
```

Here, you use the <xsd:restriction> element to indicate that this simple type restricts the base type xsd:string. Then you use <xsd:maxLength> to indicate the maximum length of the string. There are other ways you can restrict values, for example you could use patterns to specify that a string must contain valid telephone number in the form (703) 606-1234:

```
<simpleType name="telType">
  <restriction base="string">
    <pattern value="\(\d{3}\) \d{3}-\d{4}"/>
  </restriction>
</simpleType>
```

You can also restrict the values of numbers by specifying valid ranges. For example, to restrict the Quantity between 1 and 100 you'd write:

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

Visit [www.LearnXmlWS.com](http://www.LearnXmlWS.com) for more resources

```
<xsd:restriction base="xsd:short">
  <xsd:minInclusive value="1"/>
  <xsd:maxInclusive value="100"/>
</xsd:restriction>
</xsd:simpleType>
```

## Schemas and Namespaces

If you are building Web services or using XML for data interchange between your business and other businesses, you'll need to use XML namespaces to fully qualify your elements and your data types (see the [XML namespaces tutorial](#) on this site). For example, if you are using the namespace `http://schemas.devxpert.com/order`, your order document might look like this:

```
<Order xmlns="http://schemas.devxpert.com/order">
  <OrderId>2093324</OrderId>
  <RequiredDate>2001-08-01</RequiredDate>
  <ShipName>Yasser Shohoud</ShipName>
  <OrderDetails>
    <OrderItem>
      <ProductID>53034</ProductID>
      <Quantity>9</Quantity>
      <UnitPrice>12.09</UnitPrice>
    </OrderItem>
    <OrderItem>
      <ProductID>66090</ProductID>
      <Quantity>12</Quantity>
      <UnitPrice>10.09</UnitPrice>
    </OrderItem>
    <OrderItem>
      <ProductID>56091</ProductID>
      <Quantity>2</Quantity>
      <UnitPrice>87.95</UnitPrice>
    </OrderItem>
  </OrderDetails>
</Order>
```

In your schema, you specify the `targetNamespace` attribute on the `<xsd:schema>` element.

```
<schema targetNamespace="http://schemas.devxpert.com/order"
  xmlns:dx="http://schemas.devxpert.com/order" xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
```

Besides defining the `targetNamespace`, we also define the prefix `dx`, which maps to the same namespace. We'll use this prefix shortly. We also defined `elementFormDefault="qualified"` which means that elements in the Order XML document will be qualified (either by having a namespace prefix or by using a default namespace like the example above).

All types defined within the schema belong to the target namespace and must be referenced using their fully qualified name, which is the namespace prefix followed by a colon followed by the type name. For example, to reference the `OrderItemType` you'd write `dx:OrderItemType` because `dx` is the prefix that's declared to map to the target namespace.

So the full schema using namespaces would be:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://schemas.devxpert.com/order"
  xmlns:dx="http://schemas.devxpert.com/order"
  elementFormDefault="qualified">
  <!-- this is the equivalent of the OrderItem class -->
  <xsd:complexType name="OrderItemType">
    <xsd:sequence>
      <xsd:element name="ProductID" type="xsd:int"/>
      <xsd:element name="Quantity" type="xsd:short"/>
      <xsd:element name="UnitPrice" type="xsd:double"/>
    </xsd:sequence>
  </xsd:complexType>
  <!-- this corresponds to the Order class -->
  <xsd:complexType name="OrderType">
    <xsd:sequence>
      <xsd:element name="OrderId" type="xsd:int"/>
      <xsd:element name="RequiredDate" type="xsd:date"/>
      <xsd:element name="ShipName" type="xsd:string"/>
      <xsd:element name="OrderDetails" type="dx:OrderDetailsType"/>
    </xsd:sequence>
  </xsd:complexType>
  <!-- the following is the equivalent of the OrderDetails collection -->
```

Visit [www.LearnXmlWS.com](http://www.LearnXmlWS.com) for more resources

```
</xsd: sequence>
</xsd: complexType>
<xsd: complexType name="OrderDetailsType">
  <xsd: sequence>
    <xsd: element name="OrderItem"
      type="dx: OrderItemType" maxOccurs="unbounded" />
  </xsd: sequence>
</xsd: complexType>
<!-- this corresponds to the Order class -->
<xsd: element name="Order" type="dx: OrderType" />
</xsd: schema>
```

In your XML document, you use the attribute `schemaLocation` to specify which XSD schema is used to validate elements that belong to a given namespace:

```
<Order xml ns="http://schemas.devxpert.com/order"
xml ns: xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi: schemaLocation="http://schemas.devxpert.com/order d:\schemas\order.xsd" >
```

The `schemaLocation` attribute lets you specify a namespace and the corresponding schema. Its like saying "All elements that belong to this namespace must be validated using that schema". If your document happens to contain elements from different namespaces (as is the case with WSDL documents) you can specify a different schema for each namespace by listing them all in the `schemaLocation` attribute. This is very powerful because it lets you mix elements from different schemas in the same document while validating each element with the correct schema.

You might have noticed the `xsi` namespace prefix on the `schemaLocation` attribute. XSD defines a few attributes that can be used in the *XML document itself* as opposed to the XSD schema document. These attributes belong to the XML schema instance namespace, which is `http://www.w3.org/2001/XMLSchema-instance`. By convention, the prefix used for this namespace is usually `xsi`.

## Summary

In this brief introduction to XSD, you've seen how you can make a Visual Basic class to an XSD schema and how to use that schema with MSXML 4.0 to validate documents. You also learned the relation between XSD and XML namespaces and how namespaces can be used to combine elements from different schemas in one XML document.

This tutorial barely scratches the surface of what you can do with XSD schemas. There are many more features and details you might be interested in (or might not care about). Once you are comfortable with the concepts explained in this tutorial, check out the [XML Schema Primer](#) (part of the XSD specification).