The XML Mini-Symposium

John Tuohy

Presentation Goals

- Provide a basic understanding of XML and how it should be used
- Make sure you understand the why and how of XML Namespaces
- Provide a quick understanding of XML schemas

What is XML

```
<RootElementName>
   <ElementName AttributeName1="AttrValue">ElemValue</ElementName>
   <ElementName2 AttributeName2="AttrValue">
      <ChildElement>ElemValue</ChildElement>
   </ElementName>
</RootElementName>
```

- XML is a markup language used as a universal way to exchange data
- Consists mostly of elements and attributes nodes
  - There are other node types – let's ignore them
- Data is stored in a nested hierarchy
  - Top node is always a single element
  - Elements can be nested within elements
  - Attributes are placed inside of an element

More about XML

- Content of a document can be defined and validated against a definition file - Schemas
- Supports a whole set of sub-technologies
  - Just about any acronym that starts with the letter “x” (xpath, xslt, xquery, xsd, xmen)
- Is case sensitive
- Is white space and format insensitive
- Data is almost always maintained and exchanged as Unicode
- Is strict – either the entire document is correct or it is *all* wrong
  - Must strictly adhere to XML markup rules
  - Optionally can adhere to schema rules
- When XML data is exchanged it is assumed that both sides of the exchange understand the document and the data it contains

More about XML

- XML is meant to be parsed and processed by computers, not people
- Most XML providers use automated tools to define, generate and validate XML
- Most providers of XML don't really understand these technologies
  - Most do understand XML basics
  - Many do not understand schemas
  - Most do not understand namespaces
- Our goal is that you will understand this
- XML was meant to be simple … hah!
  - Creeping featurism strikes again

- Did you know that iTunes stores it library data in XML files

A simple XML document

```
<Envelope>
  <Recipient title="Mr">Fred Smith</Recipient>
  <Address state="CA" zip="12345">
    <Street>123 Elm Street</Street>
    <City>Los Angeles</City>
  </Address>
  <Body>How are you today</Body>
</Envelope>
```

Another XML document

```
<Envelopes>
  <Envelope >
    <Recipient title="Mr">Fred Smith</Recipient>
    <Address state="CA" zip="12345">
      <Street>123 Elm Street</Street>
      <City>Los Angeles</City>
    </Address>
    <Body>How are you today</Body>
  </Envelope>
  :
  :
  <Envelope>
    <Recipient title="Sir">Reginald Smythe</Recipient>
    <Address state="NY" zip="93212">
      <Street>1234 2nd Avenue</Street>
      <City>New York</City>
    </Address>
    <Body>How are you today, sir</Body>
  </Envelope>
</Envelopes>
```

When worlds collide

- Oh no, these two documents use the same element name. Does it matter?

```
<Envelope>
  <Recipient title="Mr">Fred Smith</Recipient>
  <Address state="CA" zip="12345">
    <Street>123 Elm Street</Street>
    <City>Los Angeles</City>
  </Address>
  <Body>How are you today</Body>
</Envelope>
--------------------
<Envelope>
  <Body>
    <AddNumber>
      <number1>100</number1>
      <number2>155</number2>
    </AddNumber>
    <Body>
</Envelope>
```

Namespaces

- Namespaces lets you make an element name or, much less frequently, attribute name unique

```
<x:Envelope xmlns:x="My_unique_identifier_I_hope" >
```

- The full name is the namespace + basename
- Namespace name is assigned with the “xmlns:” attribute

```
xmlns:x="My_unique_identifier_I_hope“
```

- The base name is the name within the element

```
<x:Envelope>
```

- The prefix, in this case “x” binds the base name to the namespace
  - Think of the prefix like a variable
  - In this case the full name is: My_unique_identifier_I_hope + Envelope

The reason for namespaces

- Ideally each XML document definition is unique
  - The name of each element should be unique
- Does it really matter?
  - No – most of the time it is not needed
- However
  - Schemas require that each element name be unique
  - In complex documents where multiple schemas are being used, it may be needed
  - Web Services require namespaces
  - If namespaces are used, they cannot be ignored
- Therefore
  - You need to understand how to use namespaces
  - Once understood they are easy to work with

- Did you know that the iTunes XML doesn't use namespaces?

Namespaces

- These two documents are completely different!

```
<Envelope>
  <Body>
    <AddNumber>
      <number1>100</number1>
      <number2>155</number2>
    </AddNumber>
    <Body>
</Envelope>
-----------------------

<x:Envelope xmlns:x="My_unique_identifier_I_hope">
  <x:Body>
    <x:AddNumber>
      <x:number1>100</x:number1>
      <x:number2>155</x:number2>
    </x:AddNumber>
  </x:Body>
</x:Envelope>
```

Namespace Prefixes

- Prefix names are arbitrary. You can assign any prefix to a namespace. These two documents are identical

```
<x:Envelope xmlns:x="My_unique_identifier_I_hope">
 <x:Body>
  <x:AddNumber>
   <x:number1>100</x:number1>
   <x:number2>155</x:number2>
  </x:AddNumber>
 </x:Body>
</x:Envelope>
----------------------
<MyVar:Envelope xmlns:MyVar="My_unique_identifier_I_hope">
 <MyVar:Body>
  <MyVar:AddNumber>
   <MyVar:number1>100</MyVar:number1>
   <MyVar:number2>155</MyVar:number2>
  </MyVar:AddNumber>
 </MyVar:Body>
</MyVar:Envelope>
```

Namespace Names

- The namespace names matters. If the namespace names are different, the documents are different

```
<x:Envelope xmlns:x="My_unique_identifier_I_hope">
  <x:Body>
    <x:AddNumber>
      <x:number1>100</x:number1>
      <x:number2>155</x:number2>
    </x:AddNumber>
  </x:Body>
</x:Envelope>
----------------------
<x:Envelope xmlns:x=“Yet_Another_unique_identifier">
  <x:Body>
    <x:AddNumber>
      <x:number1>100</x:number1>
      <x:number2>155</x:number2>
    </x:AddNumber>
  </x:Body>
</x:Envelope>
```

Multiple Namespaces

- You can assign multiple namespaces in the same document. That's one of the main points of namespaces.

```
<x:Envelope xmlns:x="My_unique_identifier_I_hope"
      xmlns:y="A_different_unique_id">
 <x:Body>
  <y:AddNumber>
   <y:number1>100</y:number1>
   <y:number2>155</y:number2>
  </y:AddNumber>
 </x:Body>
</x:Envelope>
```

Unique Namespace

- Ideally namespace names (URIs) are unique. Here's an easy way to make sure your namespace is unique. Use your web-address followed by anything you want

```
<x:Envelope
      xmlns:x="http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample1"
      xmlns:y="http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample2">
 <x:Body>
  <y:AddNumber>
   <y:number1>100</y:number1>
   <y:number2>155</y:number2>
  </y:AddNumber>
 </x:Body>
</x:Envelope>
```

Are namespaces that simple?

- That is all there is to namespaces. If you understood this you understand the basics of namespaces
- Now the confusion begins!

Namespace Confusion #1

- You can define namespaces anywhere in the document

```
<x:Envelope
        xmlns:x=http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample1"
        xmlns:y="http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample2">
  <x:Body>
    <y:AddNumber>
      <y:number1>100</y:number1>
      <y:number2>155</y:number2>
    </y:AddNumber>
  </x:Body>
</x:Envelope>
-----------------
<x:Envelope xmlns:x="http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample1" >
  <x:Body>
    <y:AddNumber xmlns:y="http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample2">
      <y:number1>100</y:number1>
      <y:number2>155</y:number2>
    </y:AddNumber>
  </x:Body>
</x:Envelope>
```

Namespace Confusion #2

- You can create default namespaces. These two documents are completely different

```
<Envelope xmlns="http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample1">
  <Body>
    <AddNumber>
      <number1>100</number1>
      <number2>155</number2>
    </AddNumber>
  </Body>
</Envelope>
----------------
<Envelope>
  <Body>
    <AddNumber>
      <number1>100</number1>
      <number2>155</number2>
    </AddNumber>
  </Body>
</Envelope>
```

Namespace Confusion #3

- You can mix default and regular namespaces. The two documents are the same

```
<Envelope xmlns="http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample1"
        xmlns:x="http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample2">
  <Body>
    <x:AddNumber>
      <x:number1>100</x:number1>
      <x:number2>155</x:number2>
    <x:AddNumber>
  </Body>
</Envelope>
----------------
<x:Envelope xmlns:x="http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample1"
        xmlns="http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample2">
  <x:Body>
    <AddNumber>
      <number1>100</number1>
      <number2>155</number2>
    </AddNumber>
  </x:Body>
</x:Envelope>
```

Namespace Confusion #4

- You can mix confusions 1, 2 and 3 together (factorial confusions). These two documents are the same

```
<Envelope xmlns="http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample1" >
  <Body>
    <AddNumber xmlns="http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample2">
      <number1>100</number1>
      <number2>155</number2>
    </AddNumber>
  </Body>
</Envelope>
----------------
<Envelope xmlns=http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample1”
        xmlns:x="http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample2" >
  <Body>
    <AddNumber xmlns="http://www.visualdataflex.com/schemas/SynergyXMLTalk-sample2">
      <x:number1>100</x:number1>
      <x:number2>155</x:number2>
    </AddNumber>
  </Body>
</Envelope>
```

Namespaces in Action

- Here is what a web-service request *might* look like coming into WebApp Server.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <AddNumber xmlns="http://www.visualdataflex.com/examples/testservice">
   <number1>100</number1>
   <number2>155</number2>
  </AddNumber>
 </soap:Body>
</soap:Envelope>
```

- Any equivalent markup is valid and must be accepted.

Namespace Tips

- Either your XML document will use namespaces or will it will not
  - If a document uses namespaces, you must parse and process it with namespaces – no shortcuts!
  - If you are the author of an XML document definition, consider using namespaces.
- Don't obsess about the prefix
  - The prefix is only used to bind an element to a namespace
  - Remember that the actual name of a prefix is arbitrary
  - Let the parser worry about namespace prefixes and default namespaces
- Attributes don't usually use namespaces
  - The idea is that they are already placed inside of a element, which has a namespace, so they are not needed.
  - In your schema, you can configure how an element or attribute with no prefix is handled. It can be in the default namespace or global (confusion #5)
  - By default, elements use the default namespace while attributes are global (confusion #6)

Namespace Tips

- Our XML parser has a non-namespace and a namespace interface
  - The namespace aware messages have a “NS” suffix in their name (e.g., AddElement and AddElementNS)
  - The NS messages are always passed the Namespace URI (name) and not the prefix

```
// not using namespaces
Get ChildElementValue of hoCust "TELEPHONE" to sPhone
Get ChildElement of hoCust "ADDRESS" to hoAddress
Get ChildElementValue of hoAddress "STATE" to sState
Get AttributeValue of hoCust "NAME" to sName

// using namespacess
Move "http://www.dataaccess.com/MyNamespaceURI" to sNamespace
Get ChildElementValueNS of hoCust sNameSpace "TELEPHONE" to sPhone
Get ChildElementNS of hoCust sNameSpace "ADDRESS" to hoAddress
Get ChildElementValueNS of hoAddress sNameSpace "STATE" to sState
Get AttributeValueNS of hoCust "" "NAME" to sName // or just AttributeValue
```

Namespace Tips

- If you pass an empty namespace URI, the NS interface can be used with non namespace documents.

```
Move "" to sNamespace
Get ChildElementValueNS of hoCust sNameSpace "TELEPHONE" to sPhone

<or>

Get ChildElementValue of hoCust "TELEPHONE" to sPhone
```

- Therefore you can always use the XxxxxNS interface, even with XML documents that don't use namespaces

Schemas

- A schema defines the required structure for an XML document
- They are optional
- They are themselves XML documents
- They are usually stored in a separate document – an .xsd file.
- If used:
  - They can be used as a means of identifying the document definition. Web-Service WSDLs do this
  - They can be used to validate a document
- XML documents often do not need to tell you where their schema is located. You are supposed to already know that

- Did you know that the iTunes XML doesn't use schemas?

Schemas

- Let's look at a schema for this document

```
<x:Envelope xmlns:x="http://www.visualdataflex.com/SynergyXMLTalk-schema1" >
  <x:Recipient title="Mr">Fred Smith</x:Recipient>
  <x:Address state="CA" zip="12345">
    <x:Street>123 Elm Street</x:Street>
    <x:City>Los Angeles</x:City>
  </x:Address>
  <x:Body>
    <x:Text>How are you today</x:Text>
  </x:Body>
</x:Envelope>
```

A Schema – I'll explain

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.visualdataflex.com/SynergyXMLTalk-schema1"
   xmlns:tns="http://www.visualdataflex.com/SynergyXMLTalk-schema1">
  <xs:complexType name="recipientType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="title" type="xs:string" use="required" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="addressType">
    <xs:sequence>
      <xs:element name="Street" type="xs:string" />
      <xs:element name="City" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="state" type="xs:string" use="required" />
    <xs:attribute name="zip" type="xs:int" use="required" />
  </xs:complexType>
  <xs:complexType name="envelopeType">
    <xs:sequence>
      <xs:element name="Recipient" type="tns:recipientType" />
      <xs:element name="Address" type="tns:addressType" />
      <xs:element name="Body" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Envelope" type="tns:envelopeType" />
</xs:schema>
```

Schemas

- Let's look at a schema for this slightly expanded document

```
<Envelopes xmlns="http://www.visualdataflex.com/SynergyXMLTalk-schema1" >
  <Envelope >
    <Recipient title="Mr">Fred Smith</Recipient>
    <Address state="CA" zip="12345">
      <Street>123 Elm Street</Street>
      <City>Los Angeles</City>
    </Address>
    <Body>How are you today</Body>
  </Envelope>
  :
  :
  <Envelope>
    <Recipient title="Sir">Reginald Smythe</Recipient>
    <Address state="NY" zip="93212">
      <Street>1234 2nd Avenue</Street>
      <City>New York</City>
    </Address>
    <Body>How are you today</Body>
  </Envelope>
</Envelopes>
```

The Schema

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.visualdataflex.com/SynergyXMLTalk-schema1"
   xmlns:tns="http://www.visualdataflex.com/SynergyXMLTalk-schema1">

 :
 :   all the same as the previous schema
 :

  <xs:complexType name="ArrayOfEnvelopeType">
   <xs:sequence>
    <xs:element name="Envelope" type="tns:envelopeType" maxOccurs="unbounded" />
   </xs:sequence>
  </xs:complexType>

  <xs:element name="Envelopes" type="tns:ArrayOfEnvelopeType" />

</xs:schema>
```

How will you use Schemas

- Maybe never.
  - You can always hope!
  - Web-services usually hide these in the WSDL
- You might need to read a schema to figure out what kind of values are allowed in a document.
  - This can be difficult
  - This just touches on how schemas are defined. It can get quite complicated.
  - The schema bible is the *XML Schema Part 0: Primer Second Edition* (http://www.w3.org/TR/xmlschema-0/)
- To validate an XML document against a known schema XSD document
  - This is pretty easy.
  - You will either validate as you load a document or you can validate a document at any point.
  - Check out these *cXMLDomDocument* messages: *AddExternalSchemaFile*, *ValidateDocument* and *pbValidateOnParse*
  - Check out the XML-Sample in Specialized components