Introduction to XML Schema

with examples and
hands-on exercises

WEBUCATOR

Copyright © 2024 by Webucator. All rights reserved.

No part of this manual may be reproduced or used in any manner without written permission of the copyright owner.

Version: 1.4.0

The Author

Nat Dunn

Nat Dunn is the founder of Webucator (www.webucator.com), a company that has provided training for tens of thousands of students from thousands of organizations. Nat started the company in 2003 to combine his passion for technical training with his business expertise, and to help companies benefit from both. His previous experience was in sales, business and technical training, and management. Nat has an MBA from Harvard Business School and a BA in International Relations from Pomona College.

Follow Nat on Twitter at @natdunn and Webucator at @webucator.

Class Files

Download the class files used in this manual at https://static.webucator.com/media/public/materials/classfiles/XSD101-1.4.0-introduction-to-xml-schema.zip.

Errata

Corrections to errors in the manual can be found at https://www.webucator.com/books/errata/.

Table of Contents

LESSON 1. XML Schema Basics....1
The Purpose of XML Schema....1
The Power of XML Schema....2
A First Look....2
Validating an XML Instance Document....5
LESSON 2. Simple-Type Elements....7
Overview....7
Built-in Simple Types....9
Exercise 1: Building a Simple Schema....12
User-derived Simple Types....14
Exercise 2: Restricting Element Content....23
Specifying Element Type Locally....24
Nonatomic Types....25
Exercise 3: Adding Nonatomic Types....29
Declaring Global Simple-Type Elements....31
Exercise 4: Converting Simple-Type Element Declarations from Local to Global....34
Default Values....37
Fixed Values....38
Nil Values....40
LESSON 3. Complex-Type Elements....43
Overview....43
Content Models....45
Complex Model Groups....47
Occurrence Constraints....49
Exercise 5: Adding Complex-Type Elements....51
Declaring Global Complex-Type Elements....52
Exercise 6: Converting Complex-Type Elements from Local to Global....54
Mixed Content....56
Defining Complex Types Globally....58

LESSON 4. Attributes....61
Overview....61
Empty Elements....62
Adding Attributes to Elements with Complex Content....62
Adding Attributes to Elements with Simple Content....63
Restricting Attribute Values....65
Default and Fixed Values....68
Requiring Attributes....70
Exercise 7: Adding Attributes to Elements....72
LESSON 5. Reusing Schema Components....77
Overview....77
Groups....78
Extending Complex Types....83
LESSON 6. XML Schema Keys....89
Uniqueness....89
Keys....91
LESSON 7. Tying It All Together: XML Schema....95
Tying it all Together....95
Exercise 8: Tying It All Together....96
LESSON 8. Annotating XML Schemas....101
Overview....101
Annotating a Schema....102
Exercise 9: Annotating an XML Schema....106
LESSON 9. Namespaces....107
Overview....107
Purpose of Namespaces....108
Target Namespaces....108
Default Namespaces....110
Locally Declared Elements and Attributes....111
Qualified Locals....113
The XMLSchema-instance Namespace....115
Using Multiple Namespaces....116

LESSON 1
XML Schema Basics

EVALUATION COPY: Not to be used in class.

Topics Covered

☑ The purpose of XML Schema.

☑ The limitations of DTDs.

☑ The power of XML Schema.

☑ Validating an XML instance with an XML schema.

Introduction

In this lesson, you will learn the purpose of XML Schema, the limitations of DTDs, the power of XML Schema, and to validate an XML instance with an XML schema.

Evaluation Copy

EVALUATION COPY: Not to be used in class.

1.1. The Purpose of XML Schema

XML Schema is an XML-based language used to create XML-based languages and data models. An XML schema defines element and attribute names for a class of XML documents. The schema also specifies the structure that those documents must adhere to and the type of content that each element can hold.

XML documents that attempt to adhere to an XML schema are said to be instances of that schema. If they correctly adhere to the schema, then they are valid instances. This is not the same as being well formed. A well-formed XML document follows all the syntax rules of XML, but it does not necessarily adhere to any particular schema. So, an XML document can be well formed without being valid, but it cannot be valid unless it is well formed.

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

1.2. The Power of XML Schema

You might be wondering why you need XML Schema when DTDs can be for the same purpose? The reason is that XML Schemas are more powerful than DTDs.

DTDs are similar to XML schemas in that they are used to create classes of XML documents. DTDs were around long before the advent of XML. They were originally created to define languages based on SGML, the parent of XML. Although DTDs are still common, XML Schema is a much more powerful language.

As a means of understanding the power of XML Schema, let's look at the limitations of DTD.

1. DTDs do not have built-in data types.
2. DTDs do not support user-derived data types.
3. DTDs allow only limited control over cardinality (the number of occurrences of an element within its parent).
4. DTDs do not support Namespaces or any simple way of reusing or importing other schemas.

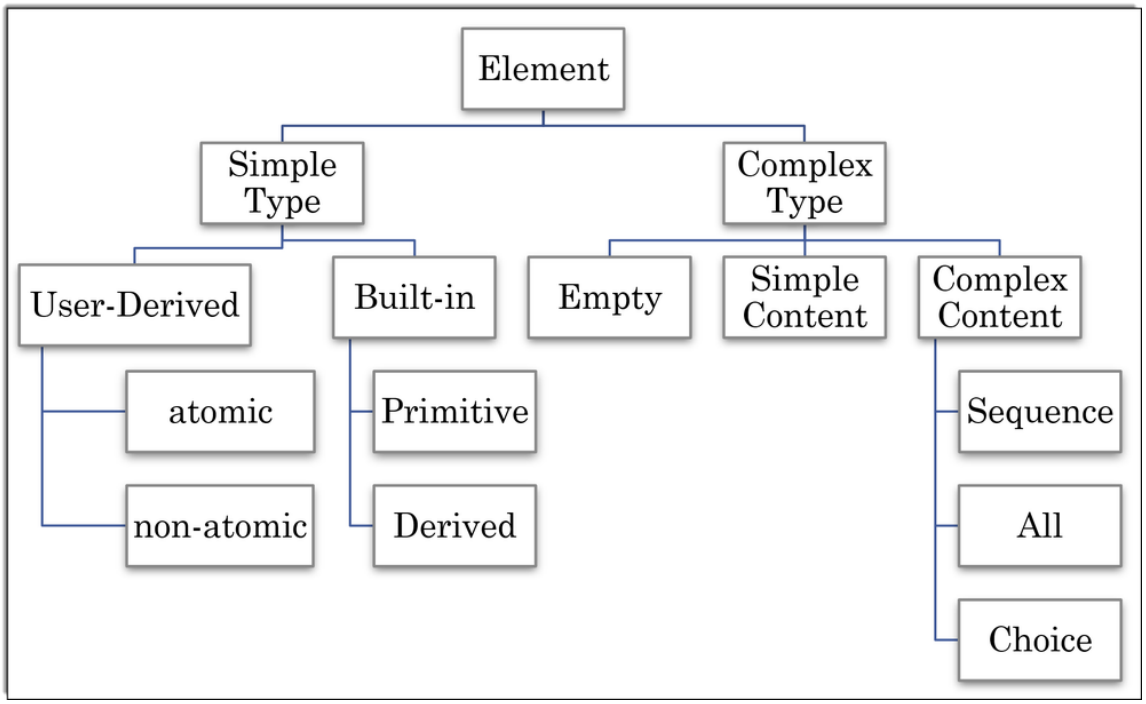
Evaluation Copy

EVALUATION COPY: Not to be used in class.

1.3. A First Look

An XML schema describes the structure of an XML instance document by defining what each element must or may contain. An element is limited by its type. For example, an element of complex type can contain child elements and attributes, whereas a simple-type element can only contain text. The diagram below gives a first look at the types of XML Schema elements:

EVALUATION COPY: Not to be used in class.



Schema authors can define their own types or use the built-in types. Throughout this course, we will refer back to this diagram as we learn to define elements. You may want to bookmark this page, so that you can easily reference it.

Evaluation Copy

The following is a high-level overview of Schema types.

1. Elements can be of simple type or complex type.
2. Simple type elements can only contain text. They cannot have child elements or attributes.
3. All the built-in types are simple types (e.g, `xs:string`).
4. Schema authors can derive simple types by restricting another simple type. For example, an email type could be derived by limiting a string to a specific pattern.
5. Simple types can be atomic (e.g, strings and integers) or non-atomic (e.g, lists).
6. Complex-type elements can contain child elements and attributes as well as text.
7. By default, complex-type elements have complex content, meaning that they have child elements.
8. Complex-type elements can be limited to having simple content, meaning they only contain text. They are different from simple type elements in that they have attributes.
9. Complex types can be limited to having no content, meaning they are empty, but they may have attributes.

EVALUATION COPY: Not to be used in class.

10. Complex types may have mixed content — a combination of text and child elements.

❖ 1.3.1. A Simple XML Schema

Let's take a look at a simple XML schema, which is made up of one complex type element with two child simple type elements.

Demo 1.1: SchemaBasics/Demos/Author.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.     <xs:element name="Author">
4.       <xs:complexType>
5.         <xs:sequence>
6.           <xs:element name="FirstName" type="xs:string" />
7.           <xs:element name="LastName" type="xs:string" />
8.         </xs:sequence>
9.       </xs:complexType>
10.    </xs:element>
11.  </xs:schema>
```

Evaluation Copy

As you can see, an XML schema is an XML document and must follow all the syntax rules of any other XML document; that is, it must be well formed. XML schemas also have to follow the rules defined in the "Schema of schemas," which defines, among other things, the structure of and element and attribute names in an XML schema.

Although it is not required, it is a common practice to use the `xs` qualifier[1] to identify Schema elements and types.

The document element of XML schemas is `xs:schema`. It takes the attribute `xmlns:xs` with the value of `http://www.w3.org/2001/XMLSchema`, indicating that the document should follow the rules of XML Schema. This will be clearer after you learn about namespaces.

In this XML schema, we see a `xs:element` element within the `xs:schema` element. `xs:element` is used to define an element. In this case it defines the element `Author` as a complex type element, which contains a sequence of two elements: `FirstName` and `LastName`, both of which are of the simple type, string.

1. Qualifiers are used to distinguish between elements and attributes from different namespaces or XML classes.

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

1.4. Validating an XML Instance Document

In the last section, you saw an example of a simple XML schema, which defined the structure of an `Author` element. The code sample below shows a valid XML instance of this XML schema.

Demo 1.2: SchemaBasics/Demos/MarkTwain.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Author xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.       xsi:noNamespaceSchemaLocation="Author.xsd">
4.       <FirstName>Mark</FirstName>
5.       <LastName>Twain</LastName>
6.   </Author>
```

Evaluation Copy

This is a simple XML document. Its document element is Author, which contains two child elements: `FirstName` and `LastName`, just as the associated XML schema requires.

The `xmlns:xsi` attribute of the document element indicates that this XML document is an instance of an XML schema. The document is tied to a specific XML schema with the `xsi:noNamespaceSchemaLocation` attribute.

There are many ways to validate the XML instance. If you are using an XML authoring tool, it very likely is able to perform the validation for you. Alternatively, there is a simple online XML Schema validator tool listed below.

- `https://www.corefiling.com/opensource/schemaValidate/` provided by DecisionSoft.

Conclusion

In this lesson, you have learned to create a very simple XML Schema and to use it to validate an XML instance document. You are now ready to learn more advanced features of XML Schema.

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

LESSON 2
Simple-Type Elements

EVALUATION COPY: Not to be used in class.

Topics Covered

☑ Built-in simple types.

☑ Deriving your own types.

☑ List types.

☑ Union types.

☑ Global simple-type elements.

☑ Default and fixed values.

☑ Nil values.

Introduction

In this lesson, you will learn to use XML Schema's built-in simple types, to derive your own types, to define list types and union types, to declare global simple-type elements, to set default and fixed values, and to allow for nil values.

EVALUATION COPY: Not to be used in class.

2.1. Overview

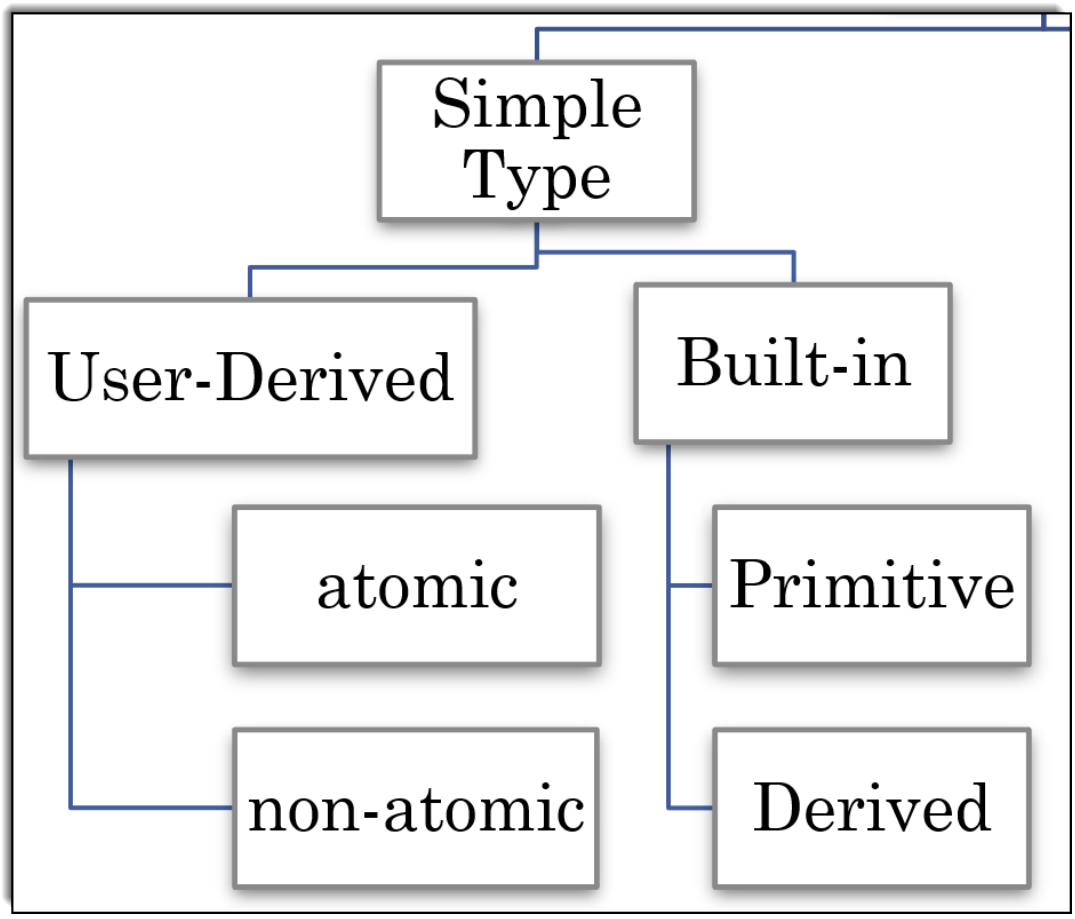
Simple-type elements have no children or attributes. For example, the `Name` element below is a simple-type element; whereas the `Person` and `HomePage` elements are not.

EVALUATION COPY: Not to be used in class.

Demo 2.1: SimpleTypes/Demos/SimpleType.xml

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <Person>
3.        <Name>Mark Twain</Name>
4.        <HomePage URL="https://www.webucator.com/marktwain"/>
5.    </Person>
```

As the diagram below shows, a simple type can either be built-in or user-derived. In this lesson, we will examine both:



EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

2.2. Built-in Simple Types

XML Schema specifies 44 built-in types, 19 of which are primitive.

❖ 2.2.1. 19 Primitive Data Types

The 19 built-in primitive types are listed below.

1. `string` — A sequence of characters, typically used to represent textual data.
2. `boolean` — A logical data type that can have only the values true or false.
3. `decimal` — A numeric data type that represents decimal numbers, with an arbitrary precision.
4. `float` — A single-precision floating-point number, for representing approximate numeric values with fractional parts.
5. `double` — A double-precision floating-point number, similar to float but with greater precision.
6. `duration` — Represents a duration of time, expressing the difference between two date/time instances.
7. `dateTime` — Represents specific instances in time, typically expressed in a date and time of day.
8. `time` — Represents a time of day, independent of any specific day.
9. `date` — Represents a calendar date, independent of time of day and time zone.
10. `gYearMonth` — Represents a specific month of a specific year.
11. `gYear` — Represents a specific year, without reference to a month or day.
12. `gMonthDay` — Represents a specific day of a specific month, without reference to a year.
13. `gDay` — Represents a specific day of the month, without reference to a month or year.
14. `gMonth` — Represents a specific month, without reference to a day or year.
15. `hexBinary` — Binary data encoded as a hexadecimal string.
16. `base64Binary` — Binary data encoded as a Base64 string.
17. `anyURI` — Represents a Uniform Resource Identifier (URI), a string of characters used to identify a resource.
18. `QName` — Represents a qualified name, which combines a namespace with a local part.

Evaluation Copy

EVALUATION COPY: Not to be used in class.

19. `NOTATION` — Represents a `QName` declared as a notation. It's used to identify non-XML data types and cannot be used directly in XML Schema.

❖ 2.2.2. Built-in Derived Data Types

The other 25 built-in data types are derived from one of the primitive types listed above.

1. `normalizedString` — A whitespace-normalized string. Line feeds, carriage returns, and tab characters are replaced with spaces.
2. `token` — A normalized string with leading and trailing whitespace removed and sequences of spaces reduced to a single space.
3. `language` — A token that represents natural language identifiers as defined in `RFC 1766` (`https://www.ietf.org/rfc/rfc1766.txt`) and `RFC 3066` (`https://www.ietf.org/rfc/rfc3066.txt`).
4. `NMTOKEN` — A token that represents XML non-markup tokens, used in attribute values.
5. `NMTOKENS` — A whitespace-separated list of `NMTOKEN` values.
6. `Name` — Represents a valid XML name, which can contain letters, digits, hyphens, underscores, and periods.
7. `NCName` — A non-colonized (no colon character) Name, used as identifiers for elements and attributes.
8. `ID` — A unique identifier type, ensuring that an `ID` value is unique within an XML document.
9. `IDREF` — A reference to an ID, used to create linkages between elements in a document.
10. `IDREFS` — A whitespace-separated list of `IDREF` values.
11. `ENTITY` — Represents a reference to an unparsed entity.
12. `ENTITIES` — A whitespace-separated list of `ENTITY` values.
13. `integer` — Represents an integer of arbitrary length.
14. `nonPositiveInteger` — Represents an integer that is zero or negative.
15. `negativeInteger` — Represents an integer that is strictly negative (less than zero).
16. `long` — Represents an integer between approximately -9.2 quintillion and 9.2 quintillion.
17. `int` — Represents an integer between approximately -2.1 billion and 2.1 billion.
18. `short` — Represents an integer between -32,768 and 32,767.
19. `byte` — Represents an integer between -128 and 127.

Evaluation Copy

EVALUATION COPY: Not to be used in class.

20. `nonNegativeInteger` — Represents an integer that is zero or positive.
21. `unsignedLong` — Represents a non-negative integer typically up to around 18.4 quintillion.
22. `unsignedInt` — Represents a non-negative integer typically up to around 4.3 billion.
23. `unsignedShort` — Represents a non-negative integer up to 65,535.
24. `unsignedByte` — Represents a non-negative integer up to 255.
25. `positiveInteger` — Represents an integer that is strictly positive (greater than zero).

❖ 2.2.3. Defining a Simple-type Element

A simple-type element is defined using the `type` attribute.

Demo 2.2: SimpleTypes/Demos/Author.xsd

Evaluation Copy

```
1.   <?xml version="1.0" ?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.       <xs:element name="Author">
4.           <xs:complexType>
5.               <xs:sequence>
6.                   <xs:element name="FirstName" type="xs:string"/>
7.                   <xs:element name="LastName" type="xs:string"/>
8.               </xs:sequence>
9.           </xs:complexType>
10.      </xs:element>
11.  </xs:schema>
```

Notice the `FirstName` and `LastName` elements in the code sample above. They are not explicitly defined as simple type elements. Instead, the type is defined with the type attribute. Because the value (`string` in both cases) is a simple type, the elements themselves are simple-type elements.

EVALUATION COPY: Not to be used in class.

Exercise 1: Building a Simple Schema

10 to 15 minutes

In this exercise, you will build a simple XML schema.

1. Open `SimpleTypes/Exercises/Song.xsd` for editing.
2. Between the open and close `xs:sequence` tags, declare three new elements:
   - `Title` of type `xs:string`.
   - `Year` of type `xs:gYear`.
   - `Artist` of type `xs:string`.
3. Save the file.
4. Try to validate `LoveMeDo.xml` against the schema you just created. If the XML document is invalid, fix your schema.

Evaluation Copy

Exercise Code 1.1: SimpleTypes/Exercises/Song.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:element name="Song">
4.          <xs:complexType>
5.              <xs:sequence>
6.                  <!--
7.                      Add three simple-type elements:
8.                      1. Title
9.                      2. Year
10.                     3. Artist
11.                 -->
12.             </xs:sequence>
13.         </xs:complexType>
14.     </xs:element>
15. </xs:schema>
```


EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

Solution: SimpleTypes/Solutions/Song.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.       <xs:element name="Song">
4.           <xs:complexType>
5.               <xs:sequence>
6.                   <xs:element name="Title" type="xs:string"/>
7.                   <xs:element name="Year" type="xs:gYear"/>
8.                   <xs:element name="Artist" type="xs:string"/>
9.               </xs:sequence>
10.          </xs:complexType>
11.      </xs:element>
12.  </xs:schema>
```

2.3. User-derived Simple Types

A schema author can derive a new simple type using the `<xs:simpleType>` element. This simple type can then be used in the same way that built-in simple types are.

Simple types are derived by restricting built-in simple types or other user-derived simple types. For example, you might want to create a simple type called password that is an eight-character string. To do so, you would start with the `xs:string` type and restrict its length to eight characters. This is done nesting the `<xs:restriction>` element inside of the `<xs:simpleType>` element.

EVALUATION COPY: Not to be used in class.

Demo 2.3: SimpleTypes/Demos/Password.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:simpleType name="Password">
4.          <xs:restriction base="xs:string">
5.              <xs:length value="8"/>
6.          </xs:restriction>
7.      </xs:simpleType>
8.      <xs:element name="User">
9.          <xs:complexType>
10.             <xs:sequence>
11.                 <xs:element name="PW" type="Password"/>
12.             </xs:sequence>
13.         </xs:complexType>
14.     </xs:element>
15. </xs:schema>
```

Demo 2.4: SimpleTypes/Demos/Password.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <User xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.      xsi:noNamespaceSchemaLocation="Password.xsd">
4.      <PW>MyPasWrd</PW>
5.  </User>
```

Evaluation Copy

❖ 2.3.1. Applying Facets

Simple types can be derived by applying one or more of the following facets.

1. `length` — Specifies the exact number of characters or list items allowed in a value.
2. `minLength` — Sets the minimum number of characters or list items a value must contain.
3. `maxLength` — Defines the maximum number of characters or list items a value can have.
4. `pattern` — Establishes a regular expression that the value must match.
5. `enumeration` — Provides a list of acceptable values.
6. `whiteSpace` — Controls how whitespace is handled in string values.
7. `minInclusive` — Sets the minimum value allowed, including the specified value itself.
8. `minExclusive` — Sets the minimum value allowed, excluding the specified value itself.

EVALUATION COPY: Not to be used in class.

9. `maxInclusive` — Defines the maximum value allowed, including the specified value itself.
10. `maxExclusive` — Defines the maximum value allowed, excluding the specified value itself.
11. `totalDigits` — Specifies the exact number of digits allowed in a numeric value.
12. `fractionDigits` — Determines the number of digits allowed after the decimal point in a numeric value.

❖ 2.3.2. Controlling Length

The length of a string can be controlled with the `length`, `minLength`, and `maxLength` facets. We used the `length` facet in the example above to create a `Password` simple type as an eight-character string. We could use `minLength` and `maxLength` to allow passwords that were between six and twelve characters in length.

The schema below shows how this is done. The two XML instances shown below it are both valid, because the length of the password is between six and twelve characters.

Evaluation Copy

Demo 2.5: SimpleTypes/Demos/Password2.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:simpleType name="Password">
4.          <xs:restriction base="xs:string">
5.              <xs:minLength value="6"/>
6.              <xs:maxLength value="12"/>
7.          </xs:restriction>
8.      </xs:simpleType>
9.      <xs:element name="User">
10.         <xs:complexType>
11.             <xs:sequence>
12.                 <xs:element name="PW" type="Password"/>
13.             </xs:sequence>
14.         </xs:complexType>
15.     </xs:element>
16. </xs:schema>
```


EVALUATION COPY: Not to be used in class.

Demo 2.6: SimpleTypes/Demos/Password2.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <User xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.      xsi:noNamespaceSchemaLocation="Password2.xsd">
4.      <PW>MyPass</PW>
5.  </User>
```

Demo 2.7: SimpleTypes/Demos/Password2b.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <User xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.      xsi:noNamespaceSchemaLocation="Password2.xsd">
4.      <PW>MyPassWord</PW>
5.  </User>
```

❖ 2.3.3. Specifying Patterns

Evaluation Copy

Patterns are specified using the `xs:pattern` element and regular expressions.[2] For example, you could use the `xs:pattern` element to restrict the Password simple type to consist of between six and twelve characters, which can only be lowercase and uppercase letters and underscores.

Demo 2.8: SimpleTypes/Demos/Password3.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:simpleType name="Password">
4.          <xs:restriction base="xs:string">
5.              <xs:pattern value="[A-Za-z_]{6,12}"/>
6.          </xs:restriction>
7.      </xs:simpleType>
8.      <xs:element name="User">
9.          <xs:complexType>
10.             <xs:sequence>
11.                 <xs:element name="PW" type="Password"/>
12.             </xs:sequence>
13.         </xs:complexType>
14.     </xs:element>
15. </xs:schema>
```

2. Regular expressions are not covered in this course. For more information on regular expressions in XML Schema, see `https://www.w3.org/TR/xmlschema-2/#dt-regex`.

EVALUATION COPY: Not to be used in class.

Demo 2.9: SimpleTypes/Demos/Password3.xml

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <User xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.        xsi:noNamespaceSchemaLocation="Password3.xsd">
4.        <PW>MyPassword</PW>
5.    </User>
```

❖ 2.3.4. Working with Numbers

Numeric simple types can be derived by limiting the value to a certain range using `minExclusive`, `minInclusive`, `maxExclusive`, and maxInclusive. You can also limit the total number of digits and the number of digits after the decimal point using `totalDigits` and `fractionDigits`, respectively.

Mins and Maxs

The following example shows how to derive a simple type called `Salary`, which is a decimal between 10,000 and 90,000.

Evaluation Copy

Demo 2.10: SimpleTypes/Demos/Employee.xsd

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.        <xs:simpleType name="Salary">
4.            <xs:restriction base="xs:decimal">
5.                <xs:minInclusive value="10000"/>
6.                <xs:maxInclusive value="90000"/>
7.            </xs:restriction>
8.        </xs:simpleType>
9.        <xs:element name="Employee">
10.           <xs:complexType>
11.               <xs:sequence>
12.                   <xs:element name="Salary" type="Salary"/>
13.               </xs:sequence>
14.           </xs:complexType>
15.       </xs:element>
16.   </xs:schema>
```


EVALUATION COPY: Not to be used in class.

Demo 2.11: SimpleTypes/Demos/JohnSmith.xml

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <Employee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.        xsi:noNamespaceSchemaLocation="Employee.xsd">
4.        <Salary>55000</Salary>
5.    </Employee>
```

Number of Digits

Using `totalDigits` and `fractionDigits`, we can further specify that the `Salary` type should consist of seven digits, two of which come after the decimal point. Both `totalDigits` and `fractionDigits` are maximums. That is, if `totalDigits` is specified as 7 and `fractionDigits` is specified as 2, a valid number could have no more than five digits total and no more than two digits after the decimal point.

Demo 2.12: SimpleTypes/Demos/Employee2.xsd

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.        <xs:simpleType name="Salary">
4.            <xs:restriction base="xs:decimal">
5.                <xs:minInclusive value="10000"/>
6.                <xs:maxInclusive value="90000"/>
7.                <xs:fractionDigits value="2"/>
8.                <xs:totalDigits value="7"/>
9.            </xs:restriction>
10.       </xs:simpleType>
11.       <xs:element name="Employee">
12.           <xs:complexType>
13.               <xs:sequence>
14.                   <xs:element name="Salary" type="Salary"/>
15.               </xs:sequence>
16.           </xs:complexType>
17.       </xs:element>
18.   </xs:schema>
```

Evaluation Copy

Demo 2.13: SimpleTypes/Demos/MarySmith.xml

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <Employee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.        xsi:noNamespaceSchemaLocation="Employee2.xsd">
4.        <Salary>55000.00</Salary>
5.    </Employee>
```


EVALUATION COPY: Not to be used in class.

❖ 2.3.5. Enumerations

A derived type can be a list of possible values. For example, the `JobTitle` element could be a list of pre-defined job titles.

Demo 2.14: SimpleTypes/Demos/Employee3.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:simpleType name="Salary">
4.          <xs:restriction base="xs:decimal">
5.              <xs:minInclusive value="10000"/>
6.              <xs:maxInclusive value="90000"/>
7.              <xs:fractionDigits value="2"/>
8.              <xs:totalDigits value="7"/>
9.          </xs:restriction>
10.     </xs:simpleType>
11.     <xs:simpleType name="JobTitle">
12.         <xs:restriction base="xs:string">
13.             <xs:enumeration value="Sales Manager"/>
14.             <xs:enumeration value="Salesperson"/>
15.             <xs:enumeration value="Receptionist"/>
16.             <xs:enumeration value="Developer"/>
17.         </xs:restriction>
18.     </xs:simpleType>
19.     <xs:element name="Employee">
20.         <xs:complexType>
21.             <xs:sequence>
22.                 <xs:element name="Salary" type="Salary"/>
23.                 <xs:element name="Title" type="JobTitle"/>
24.             </xs:sequence>
25.         </xs:complexType>
26.     </xs:element>
27. </xs:schema>
```

Demo 2.15: SimpleTypes/Demos/SteveSmith.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <Employee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.      xsi:noNamespaceSchemaLocation="Employee3.xsd">
4.      <Salary>90000.00</Salary>
5.      <Title>Sales Manager</Title>
6.  </Employee>
```


EVALUATION COPY: Not to be used in class.

❖ 2.3.6. Whitespace-handling

By default, whitespace in elements of the data type `xs:string` is preserved in XML documents; however, this can be changed for data types derived from `xs:string`. This is done with the `xs:whiteSpace` element, the value of which must be one of the following.

- `preserve` - whitespace is not normalized. That is to say, it is kept as is.
- `replace` - all tabs, line feeds, and carriage returns are replaced by single spaces.
- `collapse` - all tabs, line feeds, and carriage returns are replaced by single spaces and then all groups of single spaces are replaced with one single space. All leading and trailing spaces are then removed (i.e, trimmed).

In `SimpleTypes/Demos/Password.xsd`, we looked at restricting the `length` of a `Password` data type to eight characters using the `xs:length` element. If whitespace is preserved, then leading and trailing spaces are considered part of the password. In the following example, we set `xs:whiteSpace` to collapse, thereby discounting any leading or trailing whitespace. As you can see, this allows the XML instance author to format the document without consideration of whitespace.

Evaluation Copy

Demo 2.16: SimpleTypes/Demos/Password4.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.       <xs:simpleType name="Password">
4.           <xs:restriction base="xs:string">
5.               <xs:length value="8"/>
6.               <xs:whiteSpace value="collapse"/>
7.           </xs:restriction>
8.       </xs:simpleType>
9.       <xs:element name="User">
10.          <xs:complexType>
11.              <xs:sequence>
12.                  <xs:element name="PW" type="Password"/>
13.              </xs:sequence>
14.          </xs:complexType>
15.      </xs:element>
16.  </xs:schema>
```


EVALUATION COPY: Not to be used in class.

Demo 2.17: SimpleTypes/Demos/Password4.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <User xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.       xsi:noNamespaceSchemaLocation="Password4.xsd">
4.       <PW>
5.           12345678
6.       </PW>
7.   </User>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

Exercise 2: Restricting Element Content

15 to 20 minutes

In this exercise, you will further restrict the Song schema, so that the `Title` and `Artist` elements will have a specified pattern and the `Year` will be 1950 or later.

1. Open `SimpleTypes/Exercises/Song.xsd` and save it as `Song2.xsd` in the same directory.
2. Define a simple type called `ProperName` that follows this pattern. Note that the only space in the pattern is the one before the question mark.

```
([A-Z0-9][A-Za-z0-9\-']* ?)+
```

3. Change the `Title` and `Artist` elements to be of the `ProperName` type.
4. Define another simple type called `Year`, which is derived from `gYear` and only accepts years between 1950 and 1970, inclusive.
5. Change the `Year` element to be of the `Year` type.
6. Try to validate `SimpleTypes/Exercises/CantBuyMeLove.xml` against the schema you just created. If the XML document is invalid, fix your schema.

Evaluation Copy

Exercise Code 2.1: SimpleTypes/Exercises/Song.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:element name="Song">
4.          <xs:complexType>
5.              <xs:sequence>
6.                  <!--
7.                      Add three simple-type elements:
8.                      1. Title
9.                      2. Year
10.                     3. Artist
11.                 -->
12.             </xs:sequence>
13.         </xs:complexType>
14.     </xs:element>
15. </xs:schema>
```


EVALUATION COPY: Not to be used in class.

Solution: SimpleTypes/Solutions/Song2.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:simpleType name="ProperName">
4.          <xs:restriction base="xs:string">
5.              <xs:pattern value="([A-Z0-9][A-Za-z0-9\-']* ?)+"/>
6.              <xs:whiteSpace value="collapse"/>
7.          </xs:restriction>
8.      </xs:simpleType>
9.      <xs:simpleType name="Year">
10.         <xs:restriction base="xs:gYear">
11.             <xs:minInclusive value="1950"/>
12.             <xs:maxInclusive value="1970"/>
13.         </xs:restriction>
14.     </xs:simpleType>
15.     <xs:element name="Song">
16.         <xs:complexType>
17.             <xs:sequence>
18.                 <xs:element name="Title" type="ProperName"/>
19.                 <xs:element name="Year" type="Year"/>
20.                 <xs:element name="Artist" type="ProperName"/>
21.             </xs:sequence>
22.         </xs:complexType>
23.     </xs:element>
24. </xs:schema>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

2.4. Specifying Element Type Locally

So far in this lesson, we have been defining simple types globally and then setting the `type` attribute of element declarations to be of our derived simple types. This makes it easy to reuse a simple type across multiple elements, as we saw with the `ProperName` type in the last exercise.

It is also possible to define the type of an element locally. The type is then unnamed and applicable only to that element. The only reason to do this is to clearly show that the type is specific to that element and not meant for reuse.

EVALUATION COPY: Not to be used in class.

Demo 2.18: SimpleTypes/Demos/PasswordLocal.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:element name="User">
4.          <xs:complexType>
5.              <xs:sequence>
6.                  <xs:element name="PW">
7.                      <xs:simpleType>
8.                          <xs:restriction base="xs:string">
9.                              <xs:length value="8"/>
10.                             <xs:whiteSpace value="collapse"/>
11.                         </xs:restriction>
12.                     </xs:simpleType>
13.                 </xs:element>
14.             </xs:sequence>
15.         </xs:complexType>
16.     </xs:element>
17. </xs:schema>
```

EVALUATION COPY: Not to be used in class.

2.5. Nonatomic Types

All of XML Schema's built-in types are atomic, meaning that they cannot be broken down into meaningful bits. XML Schema provides for two nonatomic types: lists and unions.

❖ 2.5.1. Lists

List types are sequences of atomic types separated by whitespace; you can have a list of integers or a list of dates. Lists should not be confused with enumerations. Enumerations provide optional values for an element. Lists represent a space delimited list of multiple values within an element's text node.

EVALUATION COPY: Not to be used in class.

Demo 2.19: SimpleTypes/Demos/EmployeeList.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:simpleType name="Salary">
4.          <xs:restriction base="xs:decimal">
5.              <xs:minInclusive value="10000"/>
6.              <xs:maxInclusive value="90000"/>
7.              <xs:fractionDigits value="2"/>
8.              <xs:totalDigits value="7"/>
9.          </xs:restriction>
10.     </xs:simpleType>
11.     <xs:simpleType name="JobTitle">
12.         <xs:restriction base="xs:string">
13.             <xs:enumeration value="Sales Manager"/>
14.             <xs:enumeration value="Salesperson"/>
15.             <xs:enumeration value="Receptionist"/>
16.             <xs:enumeration value="Developer"/>
17.         </xs:restriction>
18.     </xs:simpleType>
19.     <xs:simpleType name="DateList">
20.         <xs:list itemType="xs:date"/>
21.     </xs:simpleType>
22.     <xs:element name="Employee">
23.         <xs:complexType>
24.             <xs:sequence>
25.                 <xs:element name="Salary" type="Salary"/>
26.                 <xs:element name="Title" type="JobTitle"/>
27.                 <xs:element name="VacationDays" type="DateList"/>
28.             </xs:sequence>
29.         </xs:complexType>
30.     </xs:element>
31. </xs:schema>
```

Evaluation Copy

Demo 2.20: SimpleTypes/Demos/SandySmith.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <Employee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.      xsi:noNamespaceSchemaLocation="EmployeeList.xsd">
4.      <Salary>44000</Salary>
5.      <Title>Salesperson</Title>
6.      <VacationDays>2023-08-13 2023-08-14 2023-08-15</VacationDays>
7.  </Employee>
```


EVALUATION COPY: Not to be used in class.

❖ 2.5.2. Unions

Union types are groupings of types, essentially allowing for the value of an element to be of more than one type. In the example below, two atomic simple types are derived: `RunningRace` and `Gymnastics`. A third simple type, `Event`, is then derived as a union of the previous two. The `Event` element is of the `Event` type, which means that it can either be of the `RunningRace` or the `Gymnastics` type.

Demo 2.21: SimpleTypes/Demos/Program.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:simpleType name="RunningRace">
4.          <xs:restriction base="xs:string">
5.              <xs:enumeration value="100 meters"/>
6.              <xs:enumeration value="10 kilometers"/>
7.              <xs:enumeration value="440 yards"/>
8.              <xs:enumeration value="10 miles"/>
9.              <xs:enumeration value="Marathon"/>
10.         </xs:restriction>
11.     </xs:simpleType>
12.     <xs:simpleType name="Gymnastics">
13.         <xs:restriction base="xs:string">
14.             <xs:enumeration value="Vault"/>
15.             <xs:enumeration value="Floor"/>
16.             <xs:enumeration value="Rings"/>
17.             <xs:enumeration value="Beam"/>
18.             <xs:enumeration value="Uneven Bars"/>
19.         </xs:restriction>
20.     </xs:simpleType>
21.     <xs:simpleType name="Event">
22.         <xs:union memberTypes="RunningRace Gymnastics"/>
23.     </xs:simpleType>
24.     <xs:element name="Program">
25.         <xs:complexType>
26.             <xs:sequence>
27.                 <xs:element name="Event" type="Event"/>
28.             </xs:sequence>
29.         </xs:complexType>
30.     </xs:element>
31. </xs:schema>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

Demo 2.22: SimpleTypes/Demos/100Meters.xml

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <Program xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.        xsi:noNamespaceSchemaLocation="Program.xsd">
4.        <Event>100 meters</Event>
5.    </Program>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

Exercise 3: Adding Nonatomic Types

10 to 15 minutes

In this exercise, you will add a nonatomic type to the song schema.

1. Open `SimpleTypes/Exercises/Song2.xsd` and save it as `Song3.xsd` in the same directory.
2. Define a new simple type called `SongLength`, which is an enumeration of three values: “Short”, “Medium’, and ”Long".
3. Define another new simple type called `SongTime`, which is a union of `xs:duration` and `SongLength`.
4. At the end of the sequence of elements within the `Song` element, insert an additional element, `Length`, which is of the `SongTime` data type.
5. Try to validate `SimpleTypes/Exercises/TicketToRide.xml` and `SimpleTypes/Exercises/EleanorRigby.xml` against the schema you just created. If either XML document is invalid, fix your schema.

Evaluation Copy

EVALUATION COPY: Not to be used in class.

Solution: SimpleTypes/Solutions/Song3.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:simpleType name="ProperName">
4.          <xs:restriction base="xs:string">
5.              <xs:whiteSpace value="collapse"/>
6.              <xs:pattern value="([A-Z0-9][A-Za-z0-9\-']* ?)+"/>
7.          </xs:restriction>
8.      </xs:simpleType>
9.      <xs:simpleType name="Year">
10.         <xs:restriction base="xs:gYear">
11.             <xs:minInclusive value="1950"/>
12.             <xs:maxInclusive value="1970"/>
13.         </xs:restriction>
14.     </xs:simpleType>
15.     <xs:simpleType name="SongLength">
16.         <xs:restriction base="xs:string">
17.             <xs:enumeration value="Short"/>
18.             <xs:enumeration value="Medium"/>
19.             <xs:enumeration value="Long"/>
20.         </xs:restriction>
21.     </xs:simpleType>
22.     <xs:simpleType name="SongTime">
23.         <xs:union memberTypes="xs:duration SongLength"/>
24.     </xs:simpleType>
25.     <xs:element name="Song">
26.         <xs:complexType>
27.             <xs:sequence>
28.                 <xs:element name="Title" type="ProperName"/>
29.                 <xs:element name="Year" type="Year"/>
30.                 <xs:element name="Artist" type="ProperName"/>
31.                 <xs:element name="Length" type="SongTime"/>
32.             </xs:sequence>
33.         </xs:complexType>
34.     </xs:element>
35. </xs:schema>
```

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

2.6. Declaring Global Simple-Type Elements

When an element declaration is a child of the `xs:schema` element, the declared element is global. Global elements can be referenced by other element declarations, allowing for element reuse. Take a look at the following example.

Demo 2.23: SimpleTypes/Demos/AuthorGlobal.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.       <xs:element name="FirstName" type="xs:string"/>
4.       <xs:element name="LastName" type="xs:string"/>
5.       <xs:element name="Author">
6.           <xs:complexType>
7.               <xs:sequence>
8.                   <xs:element ref="FirstName"/>
9.                   <xs:element ref="LastName"/>
10.              </xs:sequence>
11.          </xs:complexType>
12.      </xs:element>
13.  </xs:schema>
```

Evaluation Copy

In this example, the `FirstName` and `LastName` elements are both declared globally. The global elements are then referenced as children of the `Author` sequence.

❖ 2.6.1. Global vs. Local Simple-Type Elements

The major advantage of declaring an element globally is that the element can then be referenced throughout the schema. This makes the code more modular and easier to maintain. For example, suppose that the song schema contained `MusicWriter`, `LyricsWriter`, and `Singer` elements. Each of these elements might have the child element Name. If the Name element is declared globally, any changes to that element can be made in one location.

The major disadvantage of declaring elements globally is that all global elements must have unique names.

EVALUATION COPY: Not to be used in class.

Demo 2.24: SimpleTypes/Demos/BookLocal.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:simpleType name="PersonTitle">
4.          <xs:restriction base="xs:string">
5.              <xs:enumeration value="Mr."/>
6.              <xs:enumeration value="Ms."/>
7.              <xs:enumeration value="Dr."/>
8.          </xs:restriction>
9.      </xs:simpleType>
10.     <xs:element name="Book">
11.         <xs:complexType>
12.             <xs:sequence>
13.                 <xs:element name="Title" type="xs:string"/>
14.                 <xs:element name="Author">
15.                     <xs:complexType>
16.                         <xs:sequence>
17.                             <xs:element name="Title" type="PersonTitle"/>
18.                             <xs:element name="Name" type="xs:string"/>
19.                         </xs:sequence>
20.                     </xs:complexType>
21.                 </xs:element>
22.             </xs:sequence>
23.         </xs:complexType>
24.     </xs:element>
25. </xs:schema>
```

Evaluation Copy

Notice that there are two elements named `Title`, which can appear in different locations in the XML instance and are of different types. When the `Title` element appears at the root of the XML instance, its value can be any string; whereas, when it appears as a child of `Author`, its value is limited to "Mr.", "Ms.", or "Dr."

The example below defines a similar content model; however, because the elements are declared globally, the name `Title` cannot be used twice.

EVALUATION COPY: Not to be used in class.

Demo 2.25: SimpleTypes/Demos/BookGlobal.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:simpleType name="PersonTitle">
4.          <xs:restriction base="xs:string">
5.              <xs:enumeration value="Mr."/>
6.              <xs:enumeration value="Ms."/>
7.              <xs:enumeration value="Dr."/>
8.          </xs:restriction>
9.      </xs:simpleType>
10.     <xs:element name="BookTitle" type="xs:string"/>
11.     <xs:element name="Title" type="PersonTitle"/>
12.     <xs:element name="Name" type="xs:string"/>
13.     <xs:element name="Book">
14.         <xs:complexType>
15.             <xs:sequence>
16.                 <xs:element ref="BookTitle"/>
17.                 <xs:element name="Author">
18.                     <xs:complexType>
19.                         <xs:sequence>
20.                             <xs:element ref="Title"/>
21.                             <xs:element ref="Name"/>
22.                         </xs:sequence>
23.                     </xs:complexType>
24.                 </xs:element>
25.             </xs:sequence>
26.         </xs:complexType>
27.     </xs:element>
28. </xs:schema>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

Exercise 4: Converting Simple-Type Element Declarations from Local to Global

10 to 20 minutes

In this exercise, you will convert the element declarations in the song schema from local to global.

Evaluation Copy

1. Open `SimpleTypes/Exercises/Song3.xsd` and save it as `Song4.xsd` in the same directory.
2. Change the `Title`, `Year`, `Artist`, and `Length` elements to be declared globally.
3. Try to validate `SimpleTypes/Exercises/StrawberryFields.xml` against the schema you just created. If the XML document is invalid, fix your schema.
4. Try to validate `StrawberryFields.xml` against `Songs3.xsd`.
5. Try to validate `EleanorRigby.xml` and `TicketToRide.xml` against `Song4.xsd`.

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

Solution: SimpleTypes/Solutions/Song4.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.       <xs:simpleType name="ProperName">
4.           <xs:restriction base="xs:string">
5.               <xs:whiteSpace value="collapse"/>
6.               <xs:pattern value="([A-Z0-9][A-Za-z0-9\-']* ?)+"/>
7.           </xs:restriction>
8.       </xs:simpleType>
9.       <xs:simpleType name="Year">
10.          <xs:restriction base="xs:gYear">
11.              <xs:minInclusive value="1950"/>
12.              <xs:maxInclusive value="1970"/>
13.          </xs:restriction>
14.      </xs:simpleType>
15.      <xs:simpleType name="SongLength">
16.          <xs:restriction base="xs:string">
17.              <xs:enumeration value="Short"/>
18.              <xs:enumeration value="Medium"/>
19.              <xs:enumeration value="Long"/>
20.          </xs:restriction>
21.      </xs:simpleType>
22.      <xs:simpleType name="SongTime">
23.          <xs:union memberTypes="xs:duration SongLength"/>
24.      </xs:simpleType>
25.      <xs:element name="Title" type="ProperName"/>
26.      <xs:element name="Year" type="Year"/>
27.      <xs:element name="Artist" type="ProperName"/>
28.      <xs:element name="Length" type="SongTime"/>
29.      <xs:element name="Song">
30.          <xs:complexType>
31.              <xs:sequence>
32.                  <xs:element ref="Title"/>
33.                  <xs:element ref="Year"/>
34.                  <xs:element ref="Artist"/>
35.                  <xs:element ref="Length"/>
36.              </xs:sequence>
37.          </xs:complexType>
38.      </xs:element>
39.  </xs:schema>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

2.7. Default Values

Elements that do not have any children can have default values. To specify a default value, use the `default` attribute of the `xs:element` element.

Demo 2.26: SimpleTypes/Demos/EmployeeDefault.xsd

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      -------Lines 3 through 18 Omitted-------
19.       <xs:element name="Employee">
20.           <xs:complexType>
21.               <xs:sequence>
22.                   <xs:element name="Salary" type="Salary"/>
23.                  <xs:element name="Title" type="JobTitle" default="Salesperson"/>
24.               </xs:sequence>
25.           </xs:complexType>
26.       </xs:element>
27.   </xs:schema>
```

Evaluation Copy

When defaults are set in the XML schema, the following rules apply for the instance document.

1. If the element appears in the document with content, the default value is ignored.
2. If the element appears without content, the default value is applied.
3. If the element does not appear, the element is left out. In other words, providing a default value does not imply that the element should be inserted if the XML instance author leaves it out.

Examine the following XML instance. The `Title` element cannot be empty; it requires one of the values from the enumeration defined in the `JobTitle` simple type. However, in accordance with number 2 above, the schema processor applies the default value of `Salesperson` to the `Title` element, so the instance validates successfully.

EVALUATION COPY: Not to be used in class.

Demo 2.27: SimpleTypes/Demos/MikeSmith.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Employee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.       xsi:noNamespaceSchemaLocation="EmployeeDefault.xsd">
4.       <Salary>90000</Salary>
5.       <Title/>
6.   </Employee>
```

EVALUATION COPY: Not to be used in class.

2.8. Fixed Values

Element values can be fixed, meaning that, if they appear in the instance document, they must contain a specified value. Fixed elements are often used for boolean switches.

EVALUATION COPY: Not to be used in class.

Demo 2.28: SimpleTypes/Demos/EmployeeFixed.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:simpleType name="Salary">
4.          <xs:restriction base="xs:decimal">
5.              <xs:minInclusive value="10000"/>
6.              <xs:maxInclusive value="90000"/>
7.              <xs:fractionDigits value="2"/>
8.              <xs:totalDigits value="7"/>
9.          </xs:restriction>
10.     </xs:simpleType>
11.     <xs:simpleType name="JobTitle">
12.         <xs:restriction base="xs:string">
13.             <xs:enumeration value="Sales Manager"/>
14.             <xs:enumeration value="Salesperson"/>
15.             <xs:enumeration value="Receptionist"/>
16.             <xs:enumeration value="Developer"/>
17.         </xs:restriction>
18.     </xs:simpleType>
19.     <xs:element name="Employee">
20.         <xs:complexType>
21.             <xs:sequence>
22.                 <xs:element name="Salary" type="Salary"/>
23.                 <xs:element name="Title" type="JobTitle"/>
24.                 <xs:element name="Status" type="xs:string" fixed="current"
25.                     minOccurs="0"/>
26.             </xs:sequence>
27.         </xs:complexType>
28.     </xs:element>
29. </xs:schema>
```

Evaluation Copy

The `MinOccurs` attribute is used to make the `Status` element optional. However, if it is used, it must contain the value current or be left empty, in which case, the value current is implied. The file `SimpleTypes/Demos/LauraSmith.xml` in the `Demos` folder validates against this schema.

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

2.9. Nil Values

When an optional element is left out of an XML instance, it has no clear meaning. For example, suppose a schema declares a Name element as having required `FirstName` and `LastName` elements and an optional `MiddleName` element. And suppose a particular instance of this schema does not include the `MiddleName` element. Does this mean that the instance author did not know the middle name of the person in question or does it mean the person in question has no middle name?

Setting the `nillable` attribute of `xs:element` to true indicates that such elements can be set to nil by setting the `xsi:nil` attribute to true.

Demo 2.29: SimpleTypes/Demos/AuthorNillable.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.       <xs:element name="Author">
4.           <xs:complexType>
5.               <xs:sequence>
6.                   <xs:element name="FirstName" type="xs:string"/>
7.                   <xs:element name="MiddleName" type="xs:string" nillable="true"/>
8.                   <xs:element name="LastName" type="xs:string"/>
9.               </xs:sequence>
10.          </xs:complexType>
11.      </xs:element>
12.  </xs:schema>
```

Evaluation Copy

Demo 2.30: SimpleTypes/Demos/MarkTwain.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Author xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.       xsi:noNamespaceSchemaLocation="AuthorNillable.xsd">
4.       <FirstName>Mark</FirstName>
5.       <MiddleName xsi:nil="true"/>
6.       <LastName>Twain</LastName>
7.   </Author>
```

By including the `MiddleName` element and setting `xsi:nil` to true, we are explicitly stating that we do not know anything about Mark Twain's middle name. He might have had one, but we don't know what it is, and if we do, we're not saying.

EVALUATION COPY: Not to be used in class.

This is, of course, somewhat strange. It implies that if we had simply left the `MiddleName` out, we would be indicating that Mark Twain has no middle name. The important thing is that the application consuming your XML data can differentiate between nil values, empty elements, and missing elements.

Evaluation Copy

Conclusion

In this lesson, you have learned to work with XML Schema's built-in simple types, to derive your own simple types, and to declare simple-type elements. We will now take a look at complex-type elements.

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

LESSON 3
Complex-Type Elements

EVALUATION COPY: Not to be used in class.

Topics Covered

☑ Complex-type elements.

☑ Content models.

☑ Controlling cardinality (quantity of elements).

☑ Elements of mixed types.

☑ Named complex types.

Introduction

You have learned the basics of XML Schema and you know how to declare simple-type elements. In this lesson, you will learn to declare complex-type elements, to use content models, to control cardinality (quantity of elements), to declare elements of mixed types, and to define named complex types.

EVALUATION COPY: Not to be used in class.

3.1. Overview

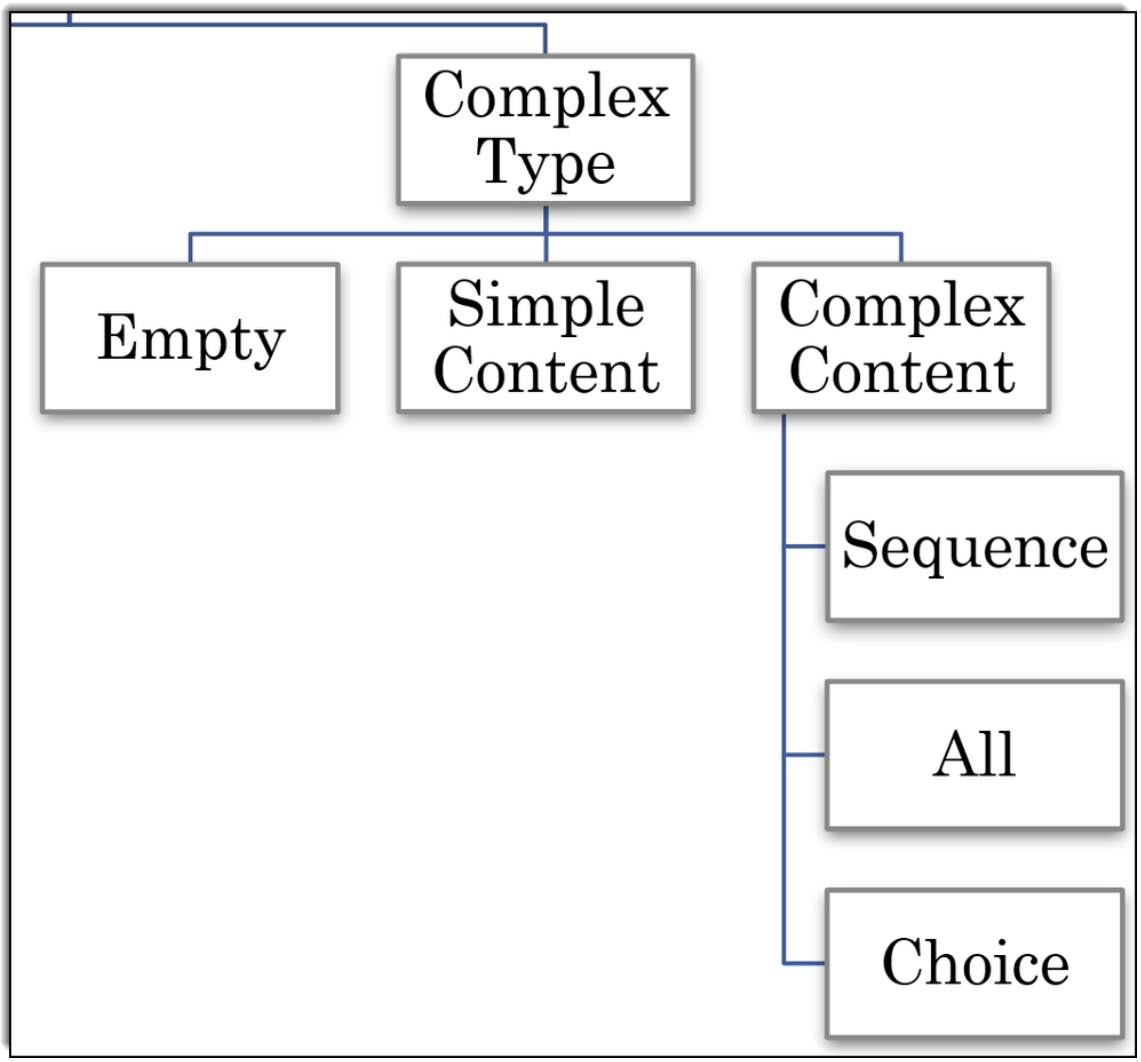
Complex-type elements have attributes, child elements, or some combination of the two. For example, the `Name` and `HomePage` elements below are both complex-type elements.

EVALUATION COPY: Not to be used in class.

Demo 3.1: ComplexTypes/Demos/ComplexType.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Person>
3.     <Name>
4.       <FirstName>Mark</FirstName>
5.       <LastName>Twain</LastName>
6.     </Name>
7.     <HomePage URL="https://www.marktwain.com"/>
8.   </Person>
```

As the diagram below shows, a complex-type element can be empty, contain simple content such as a string, or can contain complex content such as a sequence of elements.



EVALUATION COPY: Not to be used in class.

Whereas it is not necessary to explicitly declare that a simple-type element is a simple type, it is necessary to specify that a complex-type element is a complex type. This is done with the `xs:complexType` element as shown below.

```
<xs:element name="ElementName">
  <xs:complexType>
    <!--Content Model Goes Here-->
  </xs:complexType>
</xs:element>
```

EVALUATION COPY: Not to be used in class.

3.2. Content Models

Evaluation Copy

Content models are used to indicate the structure and order in which child elements can appear within their parent element. Content models are made up of model groups. The three types of model groups are listed below.

1. `xs:sequence` - the elements must appear in the order specified.
2. `xs:all` - the elements must appear, but order is not important.
3. `xs:choice` - only one of the elements can appear.

❖ 3.2.1. `xs:sequence`

The following sample shows the syntax for declaring a complex-type element as a sequence, meaning that the elements must show up in the order they are declared.

EVALUATION COPY: Not to be used in class.

```
<xs:element name="ElementName">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Child1" type="xs:string"/>
      <xs:element name="Child2" type="xs:string"/>
      <xs:element name="Child3" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

❖ 3.2.2. xs:all

The following sample shows the syntax for declaring a complex-type element as a conjunction, meaning that the elements can show up in any order.

```
<xs:element name="ElementName">
  <xs:complexType>
    <xs:all>
      <xs:element name="Child1" type="xs:string"/>
      <xs:element name="Child2" type="xs:string"/>
      <xs:element name="Child3" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Evaluation Copy

❖ 3.2.3. xs:choice

The following sample shows the syntax for declaring a complex-type element as a choice, meaning that only one of the child elements may show up.

```
<xs:element name="ElementName">
  <xs:complexType>
    <xs:choice>
      <xs:element name="Child1" type="xs:string"/>
      <xs:element name="Child2" type="xs:string"/>
      <xs:element name="Child3" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```


EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

3.3. Complex Model Groups

In the examples above, the model groups are all made up of simple-type elements. However, complex-type elements can contain other complex-type elements.

```
<xs:element name="ElementName">
  <xs:complexType>
    <xs:choice>
      <xs:element name="Child1" type="xs:string"/>
      <xs:element name="Child2">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="GC1" type="xs:string"/>
            <xs:element name="GC2" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Child3" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Evaluation Copy

Furthermore, model groups can be nested within each other. The following example illustrates this. Notice that the choice model group, which allows for either a `Salary` element or a `Wage` element is nested with in a sequence model group. Both of the subsequent instances are valid according to this schema.

EVALUATION COPY: Not to be used in class.

Demo 3.2: ComplexTypes/Demos/Employee.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.     <xs:simpleType name="Salary">
4.       <xs:restriction base="xs:decimal">
5.         <xs:minInclusive value="10000"/>
6.         <xs:maxInclusive value="90000"/>
7.       </xs:restriction>
8.     </xs:simpleType>
9.     <xs:element name="Employee">
10.      <xs:complexType>
11.        <xs:sequence>
12.          <xs:element name="Name">
13.            <xs:complexType>
14.              <xs:sequence>
15.                <xs:element name="FirstName"/>
16.                <xs:element name="LastName"/>
17.              </xs:sequence>
18.            </xs:complexType>
19.          </xs:element>
20.          <xs:choice>
21.            <xs:element name="Salary" type="Salary"/>
22.            <xs:element name="Wage" type="xs:decimal"/>
23.          </xs:choice>
24.        </xs:sequence>
25.      </xs:complexType>
26.    </xs:element>
27.  </xs:schema>
```

Demo 3.3: ComplexTypes/Demos/DaveSmith.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Employee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.     xsi:noNamespaceSchemaLocation="Employee.xsd">
4.     <Name>
5.       <FirstName>Dave</FirstName>
6.       <LastName>Smith</LastName>
7.     </Name>
8.     <Salary>90000</Salary>
9.   </Employee>
```


EVALUATION COPY: Not to be used in class.

Demo 3.4: ComplexTypes/Demos/JillSmith.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Employee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.     xsi:noNamespaceSchemaLocation="Employee.xsd">
4.     <Name>
5.       <FirstName>Jill</FirstName>
6.       <LastName>Smith</LastName>
7.     </Name>
8.     <Wage>20.50</Wage>
9.   </Employee>
```

EVALUATION COPY: Not to be used in class.

3.4. Occurrence Constraints

By default, elements that are declared locally must show up once and only once within their parent. This constraint can be changed using the `minOccurs` and `maxOccurs` attributes. The default value of each of these attributes is 1. The value of `minOccurs` can be any non-negative integer. The value of `maxOccurs` can be any positive integer or unbounded, meaning that the element can appear an infinite number of times.

The example below shows how `minOccurs` can be used to make an element optional and how `maxOccurs` can be used to allow an element to be repeated indefinitely.

EVALUATION COPY: Not to be used in class.

Demo 3.5: ComplexTypes/Demos/Employee2.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.     <xs:simpleType name="Salary">
4.       <xs:restriction base="xs:decimal">
5.         <xs:minInclusive value="10000"/>
6.         <xs:maxInclusive value="90000"/>
7.       </xs:restriction>
8.     </xs:simpleType>
9.     <xs:element name="Employee">
10.      <xs:complexType>
11.        <xs:sequence>
12.          <xs:element name="Name">
13.            <xs:complexType>
14.              <xs:sequence>
15.                <xs:element name="FirstName"/>
16.                <xs:element name="MiddleName" minOccurs="0"/>
17.                <xs:element name="LastName"/>
18.              </xs:sequence>
19.            </xs:complexType>
20.          </xs:element>
21.          <xs:choice>
22.            <xs:element name="Salary" type="Salary"/>
23.            <xs:element name="Wage" type="xs:decimal"/>
24.          </xs:choice>
25.          <xs:element name="Responsibilities">
26.            <xs:complexType>
27.              <xs:sequence>
28.                <xs:element name="Responsibility" type="xs:string"
29.                  maxOccurs="unbounded"/>
30.              </xs:sequence>
31.            </xs:complexType>
32.          </xs:element>
33.        </xs:sequence>
34.      </xs:complexType>
35.    </xs:element>
36.  </xs:schema>
```

Evaluation Copy

Note that `minOccurs` and `maxOccurs` can also be applied to model groups (e.g, `xs:sequence`) to control the number of times a model group can be repeated.

EVALUATION COPY: Not to be used in class.

Exercise 5: Adding Complex-Type Elements

15 to 25 minutes

In this exercise, you will modify the song schema, adding model groups and occurrence constraints.

1. Open `ComplexTypes/Exercises/Song.xsd` for editing.
2. Modify the schema so that the `Song` element's children can appear in any order.
3. Change the content model of the `Song` element so that it contains an `Artists` element in place of the `Artist` element.
4. Make the `Artists` element allow for one or more child `Artist` elements. Note that the `Artist` element is already declared globally.
5. Make the `Length` element optional.
6. Try to validate `ComplexTypes/Exercises/WeAreTheWorld.xml` against the schema you just created. If the XML document is invalid, fix your schema.

Evaluation Copy

EVALUATION COPY: Not to be used in class.

Solution: ComplexTypes/Solutions/Song.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
     -------Lines 3 through 28 Omitted-------
29.    <xs:element name="Song">
30.      <xs:complexType>
31.        <xs:all>
32.          <xs:element ref="Title"/>
33.          <xs:element ref="Year"/>
34.          <xs:element name="Artists">
35.            <xs:complexType>
36.              <xs:sequence>
37.                <xs:element ref="Artist" maxOccurs="unbounded"/>
38.              </xs:sequence>
39.            </xs:complexType>
40.          </xs:element>
41.          <xs:element ref="Length" minOccurs="0"/>
42.        </xs:all>
43.      </xs:complexType>
44.    </xs:element>
45.  </xs:schema>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

3.5. Declaring Global Complex-Type Elements

As with simple-type elements, complex-type elements can be declared globally by placing the element declaration as a child of the `xs:schema` element.

Globally declared elements cannot take occurrence constraints. However, the `minOccurs` and `maxOccurs` constraints can be applied to references to globally declared elements. To illustrate, look at the following example. Notice that all elements, both simple-type and complex-type, are declared globally and then referenced within the model groups. Some of the references (e.g, `Responsibilities`) have occurrence constraints assigned to them.

EVALUATION COPY: Not to be used in class.

Demo 3.6: ComplexTypes/Demos/Employee3.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.     <xs:simpleType name="Salary">
4.       <xs:restriction base="xs:decimal">
5.         <xs:minInclusive value="10000"/>
6.         <xs:maxInclusive value="90000"/>
7.       </xs:restriction>
8.     </xs:simpleType>
9.     <xs:element name="Name">
10.      <xs:complexType>
11.        <xs:sequence>
12.          <xs:element ref="FirstName"/>
13.          <xs:element ref="MiddleName" minOccurs="0"/>
14.          <xs:element ref="LastName"/>
15.        </xs:sequence>
16.      </xs:complexType>
17.    </xs:element>
18.    <xs:element name="FirstName"/>
19.    <xs:element name="MiddleName"/>
20.    <xs:element name="LastName"/>
21.    <xs:element name="Wage" type="xs:decimal"/>
22.    <xs:element name="Salary" type="Salary"/>
23.    <xs:element name="Responsibilities">
24.      <xs:complexType>
25.        <xs:sequence>
26.          <xs:element ref="Responsibility" maxOccurs="unbounded"/>
27.        </xs:sequence>
28.      </xs:complexType>
29.    </xs:element>
30.    <xs:element name="Responsibility" type="xs:string"/>
31.    <xs:element name="Employee">
32.      <xs:complexType>
33.        <xs:sequence>
34.          <xs:element ref="Name"/>
35.          <xs:choice>
36.            <xs:element ref="Salary"/>
37.            <xs:element ref="Wage"/>
38.          </xs:choice>
39.          <xs:element ref="Responsibilities" minOccurs="0"/>
40.        </xs:sequence>
41.      </xs:complexType>
42.    </xs:element>
43.  </xs:schema>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

Exercise 6: Converting Complex-Type Elements from Local to Global

10 to 15 minutes

In this exercise, you will convert the `Artists` element declaration in the song schema from local to global.

Evaluation Copy

1. Open `ComplexTypes/Exercises/Song.xsd` and save it as `Song2.xsd` in the same directory.
2. Change the `Artists` element to be declared globally.
3. Try to validate `ComplexTypes/Exercises/TheGirlIsMine.xml` against the schema you just created. If the XML document is invalid, fix your schema.
4. Then try to valiate `TheGirlIsMine.xml` against `Song`.
5. Now try to valiate `WeAreTheWord.xml` against `Song2`.

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

Solution: ComplexTypes/Solutions/Song2.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
     -------Lines 3 through 28 Omitted-------
29.    <xs:element name="Song">
30.      <xs:complexType>
31.        <xs:all>
32.          <xs:element ref="Title"/>
33.          <xs:element ref="Year"/>
34.          <xs:element ref="Artists"/>
35.          <xs:element ref="Length" minOccurs="0"/>
36.        </xs:all>
37.      </xs:complexType>
38.    </xs:element>
39.    <xs:element name="Artists">
40.      <xs:complexType>
41.        <xs:sequence>
42.          <xs:element ref="Artist" maxOccurs="unbounded"/>
43.        </xs:sequence>
44.      </xs:complexType>
45.    </xs:element>
46.  </xs:schema>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

3.6. Mixed Content

Sometimes an element will contain both child elements and character text. For example, a `para` element might contain mostly plain character text, but it could also have other elements (e.g, `emphasis`) littered throughout the character text.

As an example, let's examine look at the following XML instance document.

EVALUATION COPY: Not to be used in class.

Demo 3.7: ComplexTypes/Demos/PaulMcCartney.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Employee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.     xsi:noNamespaceSchemaLocation="Employee4.xsd">
4.     <Name>
5.       <FirstName>Paul</FirstName>
6.       <LastName>McCartney</LastName>
7.     </Name>
8.     <Salary>90000</Salary>
9.     <Bio>
10.      Worked for <Company>the Beatles</Company> as a
11.        <JobTitle>Singer</JobTitle>.
12.      Worked for <Company>the Beatles</Company> as a
13.        <JobTitle>Bass Guitarist</JobTitle>.
14.      Worked for <Company>the Wings</Company> as a
15.        <JobTitle>Singer</JobTitle>.
16.    </Bio>
17.  </Employee>
```

Evaluation Copy

Notice that the `Bio` element contains child elements `Company` and `JobTitle` as well as character text. Such elements are said to contain mixed content. The syntax for declaring elements with mixed content is shown below.

```
<xs:element name="ElementName">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="Child1" type="xs:string"/>
      <xs:element name="Child2" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The following example illustrates how to define this in our employee schema.

EVALUATION COPY: Not to be used in class.

Demo 3.8: ComplexTypes/Demos/Employee4.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
     -------Lines 3 through 39 Omitted-------
40.          <xs:element name="Bio">
41.            <xs:complexType mixed="true">
42.              <xs:sequence maxOccurs="unbounded">
43.                <xs:element name="Company" type="xs:string"/>
44.                <xs:element name="JobTitle" type="xs:string"/>
45.              </xs:sequence>
46.            </xs:complexType>
47.          </xs:element>
     -------Lines 48 through 50 Omitted-------
51.  </xs:schema>
```

EVALUATION COPY: Not to be used in class.

3.7. Defining Complex Types Globally

As with simple types, complex types can be defined globally. The example below shows how this is done.

Demo 3.9: ComplexTypes/Demos/Author.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.     <xs:complexType name="Person">
4.       <xs:sequence>
5.         <xs:element name="FirstName" type="xs:string"/>
6.         <xs:element name="LastName" type="xs:string"/>
7.       </xs:sequence>
8.     </xs:complexType>
9.     <xs:element name="Author" type="Person"/>
10.  </xs:schema>
```

As you can see, complex types are defined with the `xs:complexType` element. The major advantage of defining a complex type globally is that it can be reused. For example, a schema might allow for an

EVALUATION COPY: Not to be used in class.

`Illustrator` element as well as an `Author` element. Both elements could be of type `Person`. This way, if the `Person` type is changed later, the change will apply to both elements.

The instance document below will validate properly against the schema above.

Demo 3.10: ComplexTypes/Demos/MarkTwain.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Author xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.     xsi:noNamespaceSchemaLocation="Author.xsd">
4.     <FirstName>Mark</FirstName>
5.     <LastName>Twain</LastName>
6.   </Author>
```

Evaluation Copy

Conclusion

In this lesson, you have learned to work with complex types and complex-type elements. We will now take a look at declaring attributes.

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

LESSON 4
Attributes

EVALUATION COPY: Not to be used in class.

Topics Covered

☑ Empty elements.

☑ Element attributes.

☑ Restricting attribute values.

Introduction

In this lesson, you will learn to define empty elements, to declare element attributes, and to restrict attribute values.

EVALUATION COPY: Not to be used in class.

4.1. Overview

While attributes themselves must be of simple type, only complex-type elements can contain attributes.

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

4.2. Empty Elements

An empty element is an element that contains no content, but it may have attributes. The `HomePage` element in the instance document below is an empty element. Below the instance is the snippet from the `Author.xsd` schema that declares the `HomePage` element.

Demo 4.1: Attributes/Demos/MarkTwain.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Author xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames  ↵↵
              paceSchemaLocation="Author.xsd">
3.     <Name>
4.       <FirstName>Mark</FirstName>
5.       <LastName>Twain</LastName>
6.     </Name>
7.     <HomePage URL="https://www.marktwain.com"/>
8.   </Author>
```

Demo 4.2: Attributes/Demos/Author.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
     -------Lines 3 through 13 Omitted-------
14.          <xs:element name="HomePage">
15.            <xs:complexType>
16.              <xs:attribute name="URL" type="xs:anyURI"/>
17.            </xs:complexType>
18.          </xs:element>
     -------Lines 19 through 21 Omitted-------
22.  </xs:schema>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

4.3. Adding Attributes to Elements with Complex Content

Elements that have child elements are said to contain *complex content*. Attributes for such elements are declared after the element's model group. For example, the `Name` element in the XML instance below

EVALUATION COPY: Not to be used in class.

has two child elements and two attributes. Below the instance is the snippet from the `Author2.xsd` schema that declares the `Name` element.

Demo 4.3: Attributes/Demos/MarkTwain2.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <Author xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames ↵↵
            paceSchemaLocation="Author2.xsd">
3.    <Name Pseudonym="true" HomePage="https://www.marktwain.com">
4.      <FirstName>Mark</FirstName>
5.      <LastName>Twain</LastName>
6.    </Name>
7.  </Author>
```

Demo 4.4: Attributes/Demos/Author2.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    -------Lines 3 through 5 Omitted-------
6.          <xs:element name="Name">
7.            <xs:complexType>
8.              <xs:sequence>
9.                <xs:element name="FirstName" type="xs:string"/>
10.               <xs:element name="LastName" type="xs:string"/>
11.             </xs:sequence>
12.             <xs:attribute name="Pseudonym" type="xs:boolean"/>
13.             <xs:attribute name="HomePage" type="xs:anyURI"/>
14.           </xs:complexType>
15.         </xs:element>
    -------Lines 16 through 18 Omitted-------
19. </xs:schema>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

4.4. Adding Attributes to Elements with Simple Content

An element with simple content is one that only contains character data. If such an element contains one or more attributes, then it is a complex-type element. Elements with simple content and attributes are declared using the `xs:simpleContent` element and then extending the element with the

EVALUATION COPY: Not to be used in class.

`xs:extension` element, which must specify the type of simple content contained with the `base` attribute. The syntax is shown below.

```
<xs:element name="ElementName">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="AttName" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Evaluation Copy

For example, the `FirstName` element in the XML instance below contains only simple content and has a single attribute. Below the instance is the snippet from the `Author3.xsd` schema that declares the `FirstName` element.

Demo 4.5: Attributes/Demos/NatHawthorne.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Author xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.     xsi:noNamespaceSchemaLocation="Author3.xsd">
4.     <Name Pseudonym="true" HomePage="https://www.nathanielhawthorne.com">
5.       <FirstName Full="false">Nat</FirstName>
6.       <LastName>Hawthorne</LastName>
7.     </Name>
8.   </Author>
```


EVALUATION COPY: Not to be used in class.

Demo 4.6: Attributes/Demos/Author3.xsd

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      -------Lines 3 through 8 Omitted-------
9.                <xs:element name="FirstName">
10.                 <xs:complexType>
11.                   <xs:simpleContent>
12.                     <xs:extension base="xs:string">
13.                       <xs:attribute name="Full" type="xs:boolean"/>
14.                     </xs:extension>
15.                   </xs:simpleContent>
16.                 </xs:complexType>
17.               </xs:element>
      -------Lines 18 through 26 Omitted-------
27.   </xs:schema>
```

EVALUATION COPY: Not to be used in class.

4.5. Restricting Attribute Values

Attribute values are restricted in the same way that the values of simple-type elements are restricted. Below are three examples.

This first example shows how to restrict an attribute value by defining its type locally. You may test `Attributes/Demos/HuckFinn.xml` against this schema.

EVALUATION COPY: Not to be used in class.

Demo 4.7: Attributes/Demos/Book.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.     <xs:element name="Book">
4.       <xs:complexType>
5.         <xs:sequence>
6.           <xs:element name="Title" type="xs:string"/>
7.           <xs:element name="Author">
8.             <xs:complexType>
9.               <xs:sequence>
10.                <xs:element name="Name" type="xs:string"/>
11.              </xs:sequence>
12.              <xs:attribute name="Title">
13.                <xs:simpleType>
14.                  <xs:restriction base="xs:string">
15.                    <xs:enumeration value="Mr."/>
16.                    <xs:enumeration value="Ms."/>
17.                    <xs:enumeration value="Dr."/>
18.                  </xs:restriction>
19.                </xs:simpleType>
20.              </xs:attribute>
21.            </xs:complexType>
22.          </xs:element>
23.        </xs:sequence>
24.      </xs:complexType>
25.    </xs:element>
26.  </xs:schema>
```

This second example shows how to restrict an attribute value by applying a globally defined simple type. You may test `Attributes/Demos/TomSawyer.xml` against this schema.

EVALUATION COPY: Not to be used in class.

Demo 4.8: Attributes/Demos/Book2.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.     <xs:simpleType name="PersonTitle">
4.       <xs:restriction base="xs:string">
5.         <xs:enumeration value="Mr."/>
6.         <xs:enumeration value="Ms."/>
7.         <xs:enumeration value="Dr."/>
8.       </xs:restriction>
9.     </xs:simpleType>
10.    <xs:element name="Book">
11.      <xs:complexType>
12.        <xs:sequence>
13.          <xs:element name="Title" type="xs:string"/>
14.          <xs:element name="Author">
15.            <xs:complexType>
16.              <xs:sequence>
17.                <xs:element name="Name" type="xs:string"/>
18.              </xs:sequence>
19.              <xs:attribute name="Title" type="PersonTitle"/>
20.            </xs:complexType>
21.          </xs:element>
22.        </xs:sequence>
23.      </xs:complexType>
24.    </xs:element>
25.  </xs:schema>
```

Evaluation Copy

This third example shows how to declare an attribute with a derived type globally. You may test `Attributes/Demos/LifeOnTheMississippi.xml` against this schema.

EVALUATION COPY: Not to be used in class.

Demo 4.9: Attributes/Demos/Book3.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.    <xs:attribute name="Title">
4.      <xs:simpleType>
5.        <xs:restriction base="xs:string">
6.          <xs:enumeration value="Mr."/>
7.          <xs:enumeration value="Ms."/>
8.          <xs:enumeration value="Dr."/>
9.        </xs:restriction>
10.     </xs:simpleType>
11.   </xs:attribute>
12.   <xs:element name="Book">
13.     <xs:complexType>
14.       <xs:sequence>
15.         <xs:element name="Title" type="xs:string"/>
16.         <xs:element name="Author">
17.           <xs:complexType>
18.             <xs:sequence>
19.               <xs:element name="Name" type="xs:string"/>
20.             </xs:sequence>
21.             <xs:attribute ref="Title"/>
22.           </xs:complexType>
23.         </xs:element>
24.       </xs:sequence>
25.     </xs:complexType>
26.   </xs:element>
27. </xs:schema>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

4.6. Default and Fixed Values

❖ 4.6.1. Default Values

Attributes can have default values. To specify a default value, use the `default` attribute of the `xs:attribute` element. Default values for attributes work slightly differently than they do for elements.

EVALUATION COPY: Not to be used in class.

If the attribute is not included in the instance document, the schema processor inserts it with the default value. You may test `Attributes/Demos/NatHawthorne2.xml` against this schema.

Demo 4.10: Attributes/Demos/Author4.xsd

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      -------Lines 3 through 8 Omitted-------
9.                  <xs:element name="FirstName">
10.                   <xs:complexType>
11.                     <xs:simpleContent>
12.                       <xs:extension base="xs:string">
13.                       <xs:attribute name="Full" type="xs:boolean" default="true"/>
14.                       </xs:extension>
15.                     </xs:simpleContent>
16.                   </xs:complexType>
17.                 </xs:element>
      -------Lines 18 through 26 Omitted-------
27.   </xs:schema>
```

Evaluation Copy

❖ 4.6.2. Fixed Values

Attribute values can be fixed, meaning that, if they appear in the instance document, they must contain a specified value. Like with simple-type elements, this is done with the `fixed` attribute. You may test `Attributes/Demos/NatHawthorne3.xml` against this schema.

EVALUATION COPY: Not to be used in class.

Demo 4.11: Attributes/Demos/Author5.xsd

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      -------Lines 3 through 5 Omitted-------
6.            <xs:element name="Name">
7.              <xs:complexType>
8.                <xs:sequence>
9.                  <xs:element name="FirstName">
10.                   <xs:complexType>
11.                     <xs:simpleContent>
12.                       <xs:extension base="xs:string">
13.                       <xs:attribute name="Full" type="xs:boolean" default="true"/>
14.                       </xs:extension>
15.                     </xs:simpleContent>
16.                   </xs:complexType>
17.                 </xs:element>
18.                 <xs:element name="LastName" type="xs:string"/>
19.               </xs:sequence>
20.               <xs:attribute name="Pseudonym" type="xs:boolean" fixed="true"/>
21.               <xs:attribute name="HomePage" type="xs:anyURI"/>
22.             </xs:complexType>
23.           </xs:element>
24.         </xs:sequence>
      -------Lines 25 through 26 Omitted-------
27.   </xs:schema>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

4.7. Requiring Attributes

By default, attributes are optional, but they can be required by setting the `use` attribute of `xs:attribute` to required as shown below:

EVALUATION COPY: Not to be used in class.

Demo 4.12: Attributes/Demos/Author6.xsd

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      -------Lines 3 through 5 Omitted-------
6.            <xs:element name="Name">
7.              <xs:complexType>
8.                <xs:sequence>
9.                  <xs:element name="FirstName">
10.                   <xs:complexType>
11.                     <xs:simpleContent>
12.                       <xs:extension base="xs:string">
13.                        <xs:attribute name="Full" type="xs:boolean" default="true"/>
14.                       </xs:extension>
15.                     </xs:simpleContent>
16.                   </xs:complexType>
17.                 </xs:element>
18.                 <xs:element name="LastName" type="xs:string"/>
19.               </xs:sequence>
20.               <xs:attribute name="Pseudonym" type="xs:boolean" fixed="true"/>
21.               <xs:attribute name="HomePage" type="xs:anyURI" use="required"/>
22.             </xs:complexType>
23.           </xs:element>
      -------Lines 24 through 26 Omitted-------
27.   </xs:schema>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

Exercise 7: Adding Attributes to Elements

15 to 20 minutes

In this exercise, you will modify the song schema, so that it can successfully validate `Attributes/Exercises/TheGirlIsMine.xml`.

Evaluation Copy

1. Open `Attributes/Exercises/Song.xsd` for editing.
2. Change the schema so that the `Title` element can take the `Type` attribute, which is of type `xs:string`.
3. Change the schema so that the `Stanza` element can take an `Artist` attribute, which is of type `xs:string`. The attribute should be required.
4. Try to validate `Attributes/Exercises/TheGirlIsMine.xml` against the schema you just created. If the XML document is invalid, fix your schema.

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

Solution: Attributes/Solutions/Song.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.    <xs:element name="Song">
4.      <xs:complexType>
5.        <xs:sequence>
6.          <xs:element name="Title">
7.            <xs:complexType>
8.              <xs:simpleContent>
9.                <xs:extension base="xs:string">
10.                 <xs:attribute name="Type" type="xs:string"/>
11.               </xs:extension>
12.             </xs:simpleContent>
13.           </xs:complexType>
14.         </xs:element>
15.         <xs:element name="Year" type="xs:gYear"/>
16.         <xs:element name="Length" type="xs:string"/>
17.         <xs:element name="Artists">
18.           <xs:complexType>
19.             <xs:sequence>
20.              <xs:element name="Artist" type="xs:string" maxOccurs="unbounded"/>
21.             </xs:sequence>
22.           </xs:complexType>
23.         </xs:element>
24.         <xs:element name="Lyrics">
25.           <xs:complexType>
26.             <xs:sequence>
27.               <xs:element name="Stanza" maxOccurs="unbounded">
28.                 <xs:complexType>
29.                   <xs:sequence>
30.                   <xs:element name="Line" type="xs:string" maxOccurs="unbound ↵↵
          ed"/>
31.                   </xs:sequence>
32.                   <xs:attribute name="Artist" use="required" type="xs:string"/>
33.                 </xs:complexType>
34.               </xs:element>
35.             </xs:sequence>
36.           </xs:complexType>
37.         </xs:element>
38.       </xs:sequence>
39.     </xs:complexType>
40.   </xs:element>
41. </xs:schema>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

Conclusion

In this lesson, you have learned about creating attributes. You now are able to accomplish a lot with XML Schema.

Evaluation Copy

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

LESSON 5
Reusing Schema Components

EVALUATION COPY: Not to be used in class.

Topics Covered

☑ Element groups.

☑ Attribute groups.

☑ Reusing element and attribute groups.

Introduction

In this lesson, you will learn to define and reuse element groups and attribute groups.

EVALUATION COPY: Not to be used in class.

5.1. Overview

We have already seen several methods of reusing schema parts.

- Declaring elements globally.
- Declaring attributes globally.
- Defining global simple types.
- Defining global complex types.

We will now look at some other methods of reuse.

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

5.2. Groups

Element and attribute groups can be used to create a set structure for reuse. To illustrate the benefit of groups, let's first look at a simple XML instance and its (rather long) schema that does not use groups.

Demo 5.1: ReusingComponents/Demos/WinnieThePooh.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Book xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.       xsi:noNamespaceSchemaLocation="Book.xsd">
4.       <Title>Winnie the Pooh</Title>
5.       <Author Title="Mr." BirthYear="1882">
6.           <FirstName>A.</FirstName>
7.           <MiddleName>A.</MiddleName>
8.           <LastName>Milne</LastName>
9.           <Specialty>Childrens</Specialty>
10.      </Author>
11.      <Illustrator Title="Mr." BirthYear="1879">
12.          <FirstName>Ernest</FirstName>
13.          <MiddleName>H.</MiddleName>
14.          <LastName>Shepard</LastName>
15.      </Illustrator>
16.  </Book>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

Demo 5.2: ReusingComponents/Demos/Book.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:element name="Book">
4.          <xs:complexType>
5.              <xs:sequence>
6.                  <xs:element name="Title" type="xs:string"/>
7.                  <xs:element name="Author">
8.                      <xs:complexType>
9.                          <xs:sequence>
10.                             <xs:element name="FirstName" type="xs:string"/>
11.                             <xs:element name="MiddleName" type="xs:string"
        minOccurs="0"/>
12.                             <xs:element name="LastName" type="xs:string"/>
13.                             <xs:element name="Specialty">
14.                                 <xs:simpleType>
15.                                         <xs:restriction base="xs:string">
16.                                             <xs:enumeration value="Mystery"/>
17.                                             <xs:enumeration value="Humor"/>
18.                                             <xs:enumeration value="Horror"/>
19.                                             <xs:enumeration value="Childrens"/>
20.                                         </xs:restriction>
21.                                 </xs:simpleType>
22.                             </xs:element>
23.                         </xs:sequence>
24.                         <xs:attribute name="Title">
25.                             <xs:simpleType>
26.                                 <xs:restriction base="xs:string">
27.                                     <xs:enumeration value="Mr."/>
28.                                     <xs:enumeration value="Ms."/>
29.                                     <xs:enumeration value="Dr."/>
30.                                 </xs:restriction>
31.                             </xs:simpleType>
32.                         </xs:attribute>
33.                         <xs:attribute name="BirthYear" type="xs:gYear"/>
34.                     </xs:complexType>
35.                 </xs:element>
36.                 <xs:element name="Illustrator" minOccurs="0">
37.                     <xs:complexType>
38.                         <xs:sequence>
39.                             <xs:element name="FirstName" type="xs:string"/>
40.                             <xs:element name="MiddleName" type="xs:string"
        minOccurs="0"/>
41.                             <xs:element name="LastName" type="xs:string"/>
42.                         </xs:sequence>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

```
                        <xs:attribute name="Title">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="Mr."/>
                                    <xs:enumeration value="Ms."/>
                                    <xs:enumeration value="Dr."/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
                        <xs:attribute name="BirthYear" type="xs:gYear"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Evaluation Copy

The `Author` element and the `Illustrator` element have some elements and attributes in common. Let's see how we can make this code more modular.

❖ 5.2.1. Element Groups

First, we'll look at how we can group the `FirstName`, `MiddleName`, and `LastName` elements with `xs:group` to avoid rewriting the elements.

EVALUATION COPY: Not to be used in class.

Demo 5.3: ReusingComponents/Demos/Book2.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:group name="GroupName">
4.          <xs:sequence>
5.              <xs:element name="FirstName" type="xs:string"/>
6.              <xs:element name="MiddleName" type="xs:string" minOccurs="0"/>
7.              <xs:element name="LastName" type="xs:string"/>
8.          </xs:sequence>
9.      </xs:group>
10.     <xs:element name="Book">
11.         <xs:complexType>
12.             <xs:sequence>
13.                 <xs:element name="Title" type="xs:string"/>
14.                 <xs:element name="Author">
15.                     <xs:complexType>
16.                         <xs:sequence>
17.                             <xs:group ref="GroupName"/>
    -------Lines 18 through 38 Omitted-------
39.                     </xs:complexType>
40.                 </xs:element>
41.                 <xs:element name="Illustrator" minOccurs="0">
42.                     <xs:complexType>
43.                         <xs:sequence>
44.                             <xs:group ref="GroupName"/>
45.                         </xs:sequence>
    -------Lines 46 through 55 Omitted-------
56.                     </xs:complexType>
57.                 </xs:element>
58.             </xs:sequence>
59.         </xs:complexType>
60.     </xs:element>
61. </xs:schema>
```

Evaluation Copy

❖ 5.2.2. Attribute Groups

Now let's look at how we can use the `xs:attributeGroup` element to avoiding rewriting those attributes.

EVALUATION COPY: Not to be used in class.

Demo 5.4: ReusingComponents/Demos/Book3.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:group name="GroupName">
4.          <xs:sequence>
5.              <xs:element name="FirstName" type="xs:string"/>
6.              <xs:element name="MiddleName" type="xs:string" minOccurs="0"/>
7.              <xs:element name="LastName" type="xs:string"/>
8.          </xs:sequence>
9.      </xs:group>
10.     <xs:attributeGroup name="AttGroupPerson">
11.         <xs:attribute name="Title">
12.             <xs:simpleType>
13.                 <xs:restriction base="xs:string">
14.                     <xs:enumeration value="Mr."/>
15.                     <xs:enumeration value="Ms."/>
16.                     <xs:enumeration value="Dr."/>
17.                 </xs:restriction>
18.             </xs:simpleType>
19.         </xs:attribute>
20.         <xs:attribute name="BirthYear" type="xs:gYear"/>
21.     </xs:attributeGroup>
22.     <xs:element name="Book">
23.         <xs:complexType>
24.             <xs:sequence>
25.                 <xs:element name="Title" type="xs:string"/>
26.                 <xs:element name="Author">
27.                     <xs:complexType>
28.                         <xs:sequence>
29.                             <xs:group ref="GroupName"/>
30.                             <xs:element name="Specialty">
31.                                 <xs:simpleType>
32.                                     <xs:restriction base="xs:string">
33.                                         <xs:enumeration value="Mystery"/>
34.                                         <xs:enumeration value="Humor"/>
35.                                         <xs:enumeration value="Horror"/>
36.                                         <xs:enumeration value="Childrens"/>
37.                                     </xs:restriction>
38.                                 </xs:simpleType>
39.                             </xs:element>
40.                         </xs:sequence>
41.                         <xs:attributeGroup ref="AttGroupPerson"/>
42.                     </xs:complexType>
43.                 </xs:element>
44.                 <xs:element name="Illustrator" minOccurs="0">
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

```
45.                     <xs:complexType>
46.                         <xs:sequence>
47.                             <xs:group ref="GroupName"/>
48.                         </xs:sequence>
49.                         <xs:attributeGroup ref="AttGroupPerson"/>
50.                     </xs:complexType>
51.                 </xs:element>
52.             </xs:sequence>
53.         </xs:complexType>
54.     </xs:element>
55. </xs:schema>
```

EVALUATION COPY: Not to be used in class.

5.3. Extending Complex Types

New complex types can be derived by extending existing complex types. Both elements and attributes can be added in the new type, but nothing in the existing type can be overridden. New elements are appended to the content model, such that the original elements and new elements act as two groups that must appear in sequence.

The example below shows how the `Person` element can be extended.

EVALUATION COPY: Not to be used in class.

Demo 5.5: ReusingComponents/Demos/Book4.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.       <xs:complexType name="Person">
4.           <xs:sequence>
5.               <xs:element name="FirstName" type="xs:string"/>
6.               <xs:element name="MiddleName" type="xs:string" minOccurs="0"/>
7.               <xs:element name="LastName" type="xs:string"/>
8.           </xs:sequence>
9.           <xs:attributeGroup ref="AttGroupPerson"/>
10.      </xs:complexType>
11.      <xs:complexType name="PersonExtended">
12.          <xs:complexContent>
13.              <xs:extension base="Person">
14.                  <xs:sequence>
15.                      <xs:element name="Specialty">
16.                          <xs:simpleType>
17.                              <xs:restriction base="xs:string">
18.                                  <xs:enumeration value="Mystery"/>
19.                                  <xs:enumeration value="Humor"/>
20.                                  <xs:enumeration value="Horror"/>
21.                                  <xs:enumeration value="Childrens"/>
22.                              </xs:restriction>
23.                          </xs:simpleType>
24.                      </xs:element>
25.                  </xs:sequence>
26.              </xs:extension>
27.          </xs:complexContent>
28.      </xs:complexType>
     -------Lines 29 through 44 Omitted-------
45.      <xs:element name="Book">
46.          <xs:complexType>
47.              <xs:sequence>
48.                  <xs:element name="Title" type="xs:string"/>
49.                  <xs:element name="Author" type="PersonExtended"/>
50.                  <xs:element name="Illustrator" type="Person" minOccurs="0"/>
51.              </xs:sequence>
52.          </xs:complexType>
53.      </xs:element>
54.  </xs:schema>
```

Copy

No material changes have been made from `Book.xsd` to `Book4.xsd`. The `WinnieThePooh.xml` file would be valid according to all of these schemas; however the last one would be easier to maintain and build upon than the first.

EVALUATION COPY: Not to be used in class.

❖ 5.3.1. Abstract Types

When a type is made abstract, it cannot be used directly in an XML instance. One of its derived types must be used instead. The derived type is identified in the instance document using the `xsi:type` attribute. The schema below includes an abstract type with two derivations.

Evaluation Copy

EVALUATION COPY: Not to be used in class.

Demo 5.6: ReusingComponents/Demos/Animals.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.      <xs:complexType name="Measurement">
4.          <xs:simpleContent>
5.              <xs:extension base="xs:integer">
6.                  <xs:attribute name="units" type="xs:string"/>
7.              </xs:extension>
8.          </xs:simpleContent>
9.      </xs:complexType>
10.     <xs:element name="Weight" type="Measurement"/>
11.     <xs:element name="Name" type="xs:string"/>
12.     <!--Abstract Type-->
13.     <xs:complexType name="Animal" abstract="true">
14.         <xs:sequence>
15.             <xs:element ref="Name"/>
16.             <xs:element ref="Weight"/>
17.         </xs:sequence>
18.     </xs:complexType>
19.     <xs:complexType name="Dog">
20.         <xs:complexContent>
21.             <xs:extension base="Animal"/>
22.         </xs:complexContent>
23.     </xs:complexType>
24.     <xs:complexType name="Bird">
25.         <xs:complexContent>
26.             <xs:extension base="Animal">
27.                 <xs:sequence>
28.                     <xs:element name="WingSpan" type="Measurement"/>
29.                 </xs:sequence>
30.             </xs:extension>
31.         </xs:complexContent>
32.     </xs:complexType>
33.     <xs:element name="Animals">
34.         <xs:complexType>
35.             <xs:sequence>
36.                 <xs:element name="Animal" type="Animal" maxOccurs="unbounded"/>
37.             </xs:sequence>
38.         </xs:complexType>
39.     </xs:element>
40. </xs:schema>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

The `Animal` type is declared as abstract by setting the `abstract` attribute to `true`. It is extended by the `Dog` and `Bird` types. The `Dog` type doesn't actually modify the original type at all, but the `Bird` type addes a `WingSpan` element.

Note that the `Animal` element declared within the `Animals` element is of the abstract type `Animal`.

Let's now look at an instance document of this schema:

Demo 5.7: ReusingComponents/Demos/Animals.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Animals xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames ↵↵
              paceSchemaLocation="Animals.xsd">
3.       <Animal xsi:type="Dog">
4.           <Name>Rover</Name>
5.           <Weight units="pounds">80</Weight>
6.       </Animal>
7.       <Animal xsi:type="Bird">
8.           <Name>Tweetie</Name>
9.           <Weight units="grams">15</Weight>
10.          <WingSpan units="cm">20</WingSpan>
11.      </Animal>
12.  </Animals>
```

Evaluation Copy

Notice that each of the `Animal` elements includes an `xsi:type` attribute. If we were to remove that attribute, the instance would be become invalid because the `Animal` element is of an abstract type.

Conclusion

In this lesson, you have learned to use element and attribute groups as a way of making your code more modular. In the simple examples in this lesson, using groups did not save us much, but as schemas become bigger and more complicated, groups become an excellent way of making your schema code more modular.

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

LESSON 6
XML Schema Keys

EVALUATION COPY: Not to be used in class.

Topics Covered

☑ Uniqueness.

☑ Keys.

Introduction

Evaluation Copy

In this lesson, you will learn to require uniqueness and to work with keys.

EVALUATION COPY: Not to be used in class.

6.1. Uniqueness

XML Schema provides a mechanism for requiring that each element be unique among like elements.

This is best illustrated with an example:

EVALUATION COPY: Not to be used in class.

Demo 6.1: SchemaKeys/Demos/Unique.xsd

```
    -------Lines 1 through 16 Omitted-------
17.                 <xs:element name="Artists">
18.                     <xs:complexType>
19.                         <xs:sequence>
20.                             <xs:element name="Artist" maxOccurs="unbounded">
21.                                 <xs:complexType>
22.                                     <xs:simpleContent>
23.                                         <xs:extension base="xs:string">
24.                                             <xs:attribute name="aID"
          type="xs:string" use="required"/>
25.                                         </xs:extension>
26.                                     </xs:simpleContent>
27.                                 </xs:complexType>
28.                             </xs:element>
29.                         </xs:sequence>
30.                     </xs:complexType>
31.                 </xs:element>
32.                 <xs:element name="Lyrics">
33.                     <xs:complexType>
34.                         <xs:sequence>
35.                             <xs:element name="Stanza" maxOccurs="unbounded">
36.                                 <xs:complexType>
37.                                     <xs:sequence>
38.                                         <xs:element name="Line" type="xs:string"
          maxOccurs="unbounded"/>
39.                                     </xs:sequence>
40.                                  <xs:attribute name="Artist" type="xs:string"/>
41.                                 </xs:complexType>
42.                             </xs:element>
43.                         </xs:sequence>
44.                     </xs:complexType>
45.                 </xs:element>
46.             </xs:sequence>
47.         </xs:complexType>
48.         <xs:unique name="ArtistKey">
49.             <xs:selector xpath="Artists/Artist"/>
50.             <xs:field xpath="@aID"/>
51.         </xs:unique>
52.     </xs:element>
53. </xs:schema>
```

Evaluation Copy

The `Artist` element has an `aID` attribute, which we would like to be able to use to uniquely identify the artist. The XML Schema `xs:unique` element is used to enforce this. It takes two children:

EVALUATION COPY: Not to be used in class.

- `xs:selector` - takes an `xpath` attribute which holds an XPath 1.0 expression[3] referencing the elements affected by this constraint.
- `xs:field` - takes an `xpath` attribute which holds an XPath 1.0 expression specifying the part of the selected elements that must be unique.

In the example above, the `selector` XPath identifies all `Artist` elements that are children of an `Artists` element. The `field` XPath identifies the `aID` attribute as the part of the `Artist` element that must be unique.

In the XML instance below, each `Artist` must have a unique `aID` attribute. Try making them the same and validating.

Demo 6.2: SchemaKeys/Demos/Unique.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Song xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames ↵↵
              paceSchemaLocation="Unique.xsd">
3.       <Title Type="duet">The Girl Is Mine</Title>
4.       <Year>1983</Year>
5.       <Length>Medium</Length>
6.       <Artists>
7.           <Artist aID="MJ">Michael Jackson</Artist>
8.           <Artist aID="PM">Paul McCartney</Artist>
9.       </Artists>
     -------Lines 10 through 41 Omitted-------
42.  </Song>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

6.2. Keys

XML Schema also provides a mechanism for keys and key references - that is, for creating a relationship between elements through the value of an attribute or contained element. The `xs:key` and `xs:keyref` elements are used to create such a relationship.

3. The `xpath` attribute only supports a subset of XPath 1.0. See `https://www.w3.org/TR/2004/PER-xmlschema-1-20040318/structures.html#coss-identity-constraint` for details.

EVALUATION COPY: Not to be used in class.

Demo 6.3: SchemaKeys/Demos/Keys.xsd

```
    -------Lines 1 through 16 Omitted-------
17.                 <xs:element name="Artists">
18.                     <xs:complexType>
19.                         <xs:sequence>
20.                             <xs:element name="Artist" maxOccurs="unbounded">
21.                                 <xs:complexType>
22.                                     <xs:simpleContent>
23.                                         <xs:extension base="xs:string">
24.                                             <xs:attribute name="aID"
          type="xs:string" use="required"/>
25.                                         </xs:extension>
26.                                     </xs:simpleContent>
27.                                 </xs:complexType>
28.                             </xs:element>
29.                         </xs:sequence>
30.                     </xs:complexType>
31.                 </xs:element>
32.                 <xs:element name="Lyrics">
33.                     <xs:complexType>
34.                         <xs:sequence>
35.                             <xs:element name="Stanza" maxOccurs="unbounded">
36.                                 <xs:complexType>
37.                                     <xs:sequence>
38.                                         <xs:element name="Line" type="xs:string"
          maxOccurs="unbounded"/>
39.                                     </xs:sequence>
40.                                   <xs:attribute name="Artist" type="xs:string"
          use="required"/>
41.                                 </xs:complexType>
42.                             </xs:element>
43.                         </xs:sequence>
44.                     </xs:complexType>
45.                 </xs:element>
46.             </xs:sequence>
47.         </xs:complexType>
48.         <xs:key name="ArtistKey">
49.             <xs:selector xpath="Artists/Artist"/>
50.             <xs:field xpath="@aID"/>
51.         </xs:key>
52.         <xs:keyref name="ArtistKeyRef" refer="ArtistKey">
53.             <xs:selector xpath="Lyrics/Stanza"/>
54.             <xs:field xpath="@Artist"/>
55.         </xs:keyref>
56.     </xs:element>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

```
57.   </xs:schema>
```

Like the `xs:unique` element, the `xs:key` and `xs:keyref` elements each contain `xs:selector` and `xs:field` child elements.

The `xs:key` element is used to identify the elements being referenced by the elements specified by the `xs:keyref` element.

Evaluation Copy

In the example above, the `Artist` attribute of the `Stanza` element must point to an `Artist` element's `aID` attribute, which must be unique.

In the XML instance below, each `Artist` must have a unique `aID` attribute and each `Stanza` element must have an `Artist` attribute with the same value as one of the `Artist`'s `aID` attributes. Try making changing the value of a `Stanza`'s `Artist` attribute to something arbitrary and validating.

EVALUATION COPY: Not to be used in class.

Demo 6.4: SchemaKeys/Demos/Keys.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Song xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames ↵↵
           paceSchemaLocation="Keys.xsd">
3.       <Title Type="duet">The Girl Is Mine</Title>
4.       <Year>1983</Year>
5.       <Length>Medium</Length>
6.       <Artists>
7.           <Artist aID="MJ">Michael Jackson</Artist>
8.           <Artist aID="PM">Paul McCartney</Artist>
9.       </Artists>
10.      <Lyrics>
11.          <Stanza Artist="MJ">
12.              <Line>Every night she walks right in my dreams</Line>
13.              <Line>Every night she walks right in my dreams</Line>
     -------Lines 14 through 19 Omitted-------
20.          </Stanza>
21.          <Stanza Artist="PM">
22.              <Line>I don't understand the way you think</Line>
23.              <Line>Saying that she's yours not mine</Line>
     -------Lines 24 through 31 Omitted-------
32.          </Stanza>
33.          <Stanza Artist="MJ">
34.              <Line>I know she'll tell you I'm the one for her</Line>
35.              <Line>'Cause she said I blow her mind</Line>
     -------Lines 36 through 39 Omitted-------
40.          </Stanza>
41.      </Lyrics>
42.  </Song>
```

Evaluation Copy

Conclusion

In this lesson, you have learned to work with unique values and keys.

EVALUATION COPY: Not to be used in class.

LESSON 7

Tying It All Together: XML Schema

EVALUATION COPY: Not to be used in class.

Topics Covered

☑ Practicing.

Evaluation Copy

Introduction

In this lesson, you will practice all the concepts you have learned thus far.

EVALUATION COPY: Not to be used in class.

7.1. Tying it all Together

EVALUATION COPY: Not to be used in class.

Exercise 8: Tying It All Together

60 to 120 minutes

In this exercise, you will write a schema for one of the two documents shown below. You will then give your schema to another student, who will mark up the document as a valid XML instance of your schema. Likewise, you will mark up the document according to someone else's schema. Save both documents in the `TyingItTogetherXsd` folder. Validate your document. If there are errors, make necessary changes until it is valid.

Evaluation Copy

❖ E8.1. Business Letter

If you are likely to be working with *document-centric* content (e.g, books, manuals, or product documentation), you should build a schema for the business letter shown below.

EVALUATION COPY: Not to be used in class.

Exercise Code 8.1: TyingItTogetherXsd/Exercises/BusinessLetter.txt

```
1.   September 8, 2008
2.
3.   Joshua Woodlock
4.   Woodlock & Woodlock
5.   291 Broadway Ave.
6.   New York, NY 10007
7.   United States
8.
9.   Dear Mr. Woodlock:
10.
11.  Along with this letter, I have enclosed the following items:
12.
13.      - two original, execution copies of the Webucator Master Services Agreement
14.      - two original, execution copies of the Webucator Premier Support for Devel ↵↵
               opers Services Description between Woodlock & Woodlock and Webucator,
                Inc.
15.
16.  Please sign and return all four original, execution copies to me at your earliest
               convenience.  Upon receipt of the executed copies, we will immediately
               return a fully executed, original copy of both agreements to you.
17.
18.  Please send all four original, execution copies to my attention as follows:
19.
20.      Webucator, Inc.
21.      4933 Jamesville Rd.
22.      Jamesville, NY 13078  USA
23.      Attn: Bill Smith
24.
25.  If you have any questions, feel free to call me at 800-555-1000 x123 or e-mail
               me at bsmith@webucator.com.
26.
27.  Best regards,
28.
29.  Bill Smith
30.  VP, Operations
```

Evaluation Copy

❖ E8.2. Transaction Log

If you are likely to be working with *data-centric* content (e.g, more structured data that maps to a database), you should build a schema for the transaction log described below.

EVALUATION COPY: Not to be used in class.

Overview

A networking website has a feature that allows people to make connections through other connections they have made in the past. A member can search the member list and on finding someone with whom (s)he would like to connect, (s)he can ask a mutual connection to pass on a message to that person.

Jeremy Coleman is a member of this website. Jeremy is looking for a new sales manager and, in searching the networking website, finds a woman named Melanie Littleton who seems to fit the bill. Seeing that Liza is connected with Fred Harris, an associate of his, Jeremy writes a message to Liza and a separate message to Jeremy asking him to forward the message to Liza.

Evaluation Copy

The system keeps a log of this transaction and stores it in an XML format.

Your job is to create a schema to represent this data. A person or computer program reading an XML instance of your schema should be able to figure out the flow of the messages and the role of each individual involved.

Challenge

Separate the schema definition into multiple files. For example, place all type definitions in a `Types.xsd` file and import that into your main schema.

EVALUATION COPY: Not to be used in class.

Exercise Code 8.2: TyingItTogetherXsd/Exercises/TransactionLog.txt

```
1.  CONTACT INFORMATION
2.  ===========
3.  Jeremy Coleman's contact information
4.  Jeremy H. Coleman
5.  President
6.  Atlanta Sales Corp
7.  210 Peachtree Street NW
8.  Atlanta, GA 30303
9.  jcoleman@atlantasalescorp.com
10. 404-555-1111
11.
12. Fred Harris' contact information
13. Fred Allen Harris
14. COO
15. Farout Ferris Wheels
16. 1333 Eastern Ave
17. Takoma Park, MD 20912
18. 301-555-2323
19. fred.harris@faroutwheels.com
20.
21. Melanie Littleton's contact information
22. Melanie Littleton
23. Vice President, Sales
24. Saving Sinking Ships, Inc.
25. 2207 7th Ave
26. New York, NY 10027
27. (212) 555-3331
28. littleton@sssships.com
29.
30. MESSAGE 1: Jeremy's Message to Liza
31. ===========
32. Dear Ms. Littleton,
33.
34. I found your resume on this site and I would be interested in speaking with you
            about a possible job opening. Please visit our website at www.at ↵↵
            lantasalescorp.com and let me know if you would be interested in
            speaking.
35.
36. Please do not use this system to contact me as I rarely check it. Rather email
            me at jcoleman@atlantasalescorp.com.
37.
38. Thanks!
39.
40. Jeremy
41.
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

```
42.  MESSAGE 2: Jeremy's Message to Fred
43.  ===========
44.  Hi Fred,
45.
46.  Hope all is well with you! Would you mind passing this message along to Melanie
          Littleton.
47.
48.  She looks like she'd make a great salesperson to replace Freddy!
49.
50.  Take care!
51.
52.  Jeremy
```

Evaluation Copy

Conclusion

Congratulations! You have come a long way.

EVALUATION COPY: Not to be used in class.

LESSON 8
Annotating XML Schemas

EVALUATION COPY: Not to be used in class.

Topics Covered

☑ Annotating XML schemas.

☑ Creating HTML documentation from XML schemas.

Introduction

In this lesson, you will learn to annotate XML schemas, and to create HTML documentation from XML schemas.

EVALUATION COPY: Not to be used in class.

8.1. Overview

One of the nice features of XML Schema is that comments about the schema itself can be made within built-in XML elements. This makes it possible to run a transformation against a schema to builds documentation in HTML or some other human-readable format.

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

8.2. Annotating a Schema

The `xs:annotation` element is used to document a schema. It can take two elements: `xs:documentation` and `xs:appInfo`, which are used to provide human-readable and machine-readable notes, respectively.

The `xs:annotation` element can go at the beginning of most schema constructions, including `xs:schema`, `xs:element`, `xs:attribute`, `xs:simpleType`, `xs:complexType`, `xs:group`, and `xs:attributeGroup`.

Evaluation Copy

Both the `xs:documentation` and `xs:appInfo` elements can contain any content, including undeclared elements and attributes. This allows the schema author to insert elements (e.g, HTML elements) to structure or format the documentation.

The sample below shows an annotated XML schema.

EVALUATION COPY: Not to be used in class.

Demo 8.1: AnnotatingXMLSchemas/Demos/Book.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.    <xs:group name="GroupName">
4.      <xs:annotation>
5.        <xs:documentation>
6.          This group can be used with any element that represents a <strong>per
              son</strong>.
7.          It provides for FirstName, MiddleName (?), and LastName.
8.        </xs:documentation>
9.      </xs:annotation>
10.     <xs:sequence>
11.       <xs:element name="FirstName" type="xs:string"/>
12.       <xs:element name="MiddleName" type="xs:string" minOccurs="0"/>
13.       <xs:element name="LastName" type="xs:string"/>
14.     </xs:sequence>
15.   </xs:group>
16.   <xs:attributeGroup name="AttGroupPerson">
17.     <xs:annotation>
18.       <xs:documentation>
19.         This attribute group can be used with any element that represents a
20.         person.  It provides for Title (?) and BirthYear (?).
21.       </xs:documentation>
22.     </xs:annotation>
23.     <xs:attribute name="Title">
24.       <xs:annotation>
25.         <xs:documentation>
26.           This optional attribute provides the title of the person in question.

27.           There is no default value.
28.         </xs:documentation>
29.       </xs:annotation>
30.       <xs:simpleType>
31.         <xs:restriction base="xs:string">
32.           <xs:enumeration value="Mr."/>
33.           <xs:enumeration value="Ms."/>
34.           <xs:enumeration value="Dr."/>
35.         </xs:restriction>
36.       </xs:simpleType>
37.     </xs:attribute>
38.     <xs:attribute name="BirthYear" type="xs:gYear"/>
39.   </xs:attributeGroup>
40.   <xs:element name="Book">
41.     <xs:annotation>
42.       <xs:documentation>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

```
43.          Root Element.  Contains the Title, Author, and Illustrator elements.
44.        </xs:documentation>
45.      </xs:annotation>
46.      <xs:complexType>
47.        <xs:sequence>
48.          <xs:element name="Title" type="xs:string"/>
49.          <xs:element name="Author">
50.            <xs:annotation>
51.              <xs:documentation>
52.                The Author element contains the elements defined in the GroupName
53.                element group followed by the Specialty element and the attributes
54.                defined in the AttGroupPerson attribute group.
55.              </xs:documentation>
56.            </xs:annotation>
57.            <xs:complexType>
58.              <xs:sequence>
59.                <xs:group ref="GroupName"/>
60.                <xs:element name="Specialty">
61.                  <xs:simpleType>
62.                    <xs:restriction base="xs:string">
63.                      <xs:enumeration value="Mystery"/>
64.                      <xs:enumeration value="Humor"/>
65.                      <xs:enumeration value="Horror"/>
66.                      <xs:enumeration value="Childrens"/>
67.                    </xs:restriction>
68.                  </xs:simpleType>
69.                </xs:element>
70.              </xs:sequence>
71.              <xs:attributeGroup ref="AttGroupPerson"/>
72.            </xs:complexType>
73.          </xs:element>
74.          <xs:element name="Illustrator" minOccurs="0">
75.            <xs:annotation>
76.              <xs:documentation>
77.                The Illustrator element contains the elements defined in the
78.                GroupName element group and the attributes defined in the
79.                AttGroupPerson attribute group.
80.              </xs:documentation>
81.            </xs:annotation>
82.            <xs:complexType>
83.              <xs:sequence>
84.                <xs:group ref="GroupName"/>
85.              </xs:sequence>
86.              <xs:attributeGroup ref="AttGroupPerson"/>
```

Evaluation Copy

EVALUATION COPY: Not to be used in class.

```
87.            </xs:complexType>
88.          </xs:element>
89.        </xs:sequence>
90.      </xs:complexType>
91.    </xs:element>
92.  </xs:schema>
```

❖ 8.2.1. Transforming an XML Schema for Documentation

The `Annotation.xsl` file in the `Annotation/Demos` folder is an XSLT document that can be applied to an annotated schema to create human-readable HTML documentation. The screenshot below shows what it produces when applied to `Annotation/Demos/Books.xsd`.

Schema Documentation

group: GroupName

This group can be used with any element that represents a person. It provides for FirstName, MiddleName (?), and LastName.

attributeGroup: AttGroupPerson

This attribute group can be used with any element that represents a person. It provides for Title (?) and BirthYear (?).

attribute: Title

This optional attribute provides the title of the person in question. There is no default value.

element: Book

Root Element. Contains the Title, Author, and Illustrator elements.

element: Author

The Author element contains the elements defined in the GroupName element group followed by the Specialty element and the attributes defined in the AttGroupPerson attribute group.

element: Illustrator

The Illustrator element contains the elements defined in the GroupName element group and the attributes defined in the AttGroupPerson attribute group.

EVALUATION COPY: Not to be used in class.

Exercise 9: Annotating an XML Schema

10 to 15 minutes

In this exercise, you will annotate the schema you created in the "Tying It All Together" exercise (see page 95).

1. Open the schema you created in the "Tying It All Together" (see page 95) and save it in the `AnnotatingXMLSchemas/Exercises` directory.
2. Annotate the schema as you see fit.
3. Transform the XML document against `Annotation.xsl` to see the documentation.

Evaluation Copy

Conclusion

In this lesson, you have learned to annotate XML schemas and to use the annotation to output HTML documentation.

EVALUATION COPY: Not to be used in class.

LESSON 9
Namespaces

EVALUATION COPY: Not to be used in class.

Topics Covered

☑ The purpose of namespaces.

☑ Target namespaces.

☑ Default namespace.

☑ Qualifying namespaces.

Introduction

In this lesson, you will learn the purpose of namespaces, to declare target namespaces, to set a default namespace, and to qualify namespaces for differentiation.

Evaluation Copy

EVALUATION COPY: Not to be used in class.

9.1. Overview

Namespaces are used to group elements and attributes that relate to each other in some special way. Namespaces are held in a unique URI (Uniform Resource Identifier). Note that, although it is possible that an XML schema is kept at this URI, it is not required. This can be a bit confusing. It is important to understand that a namespace is a set of rules that can be enforced by an application in whatever way the application wishes.

As an example, modern HTML editors understand the `https://www.w3.org/1999/xhtml` namespace. It is unlikely that these editors ever visit the URI that holds the XHTML namespace. Instead, these applications have built-in functionality to support the namespace. The main reason a URI is used is

EVALUATION COPY: Not to be used in class.

to provide a unique variable name to hold the namespace. Namespace authors should use URIs that they own to prevent conflicts with each other.

EVALUATION COPY: Not to be used in class.

9.2. Purpose of Namespaces

As described above, one purpose of namespaces is to provide a unique identifier for a group of element and attribute declarations.

Another purpose is to allow instance documents to be made up of a combination of such groups without having name conflicts. For example, we could hold the book schema and song schema we have worked on throughout this course in separate namespaces. Now suppose you wanted to use both schemas to create a book of songs. Both songs and books can have `Title` elements. This could potentially be a source of confusion as an application might not understand which `Title` element to apply. By specifying which namespace the `Title` elements come from, the confusion is removed.

EVALUATION COPY: Not to be used in class.

9.3. Target Namespaces

A schema can be used to populate a namespace. So far, we have not created namespaces with our schemas. That is why the root elements of our XML instances all include the `xsi:noNamespaceSchemaLocation` attribute.

To populate a namespace with an XML schema, set the `targetNamespace` attribute of the `xs:schema` element to a URI. You must also include a `xmlns` attribute, so that global elements declared in the target namespace can be referenced within the schema.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://webucator.com/Artist"
xmlns="http://webucator.com/Artist">
```


EVALUATION COPY: Not to be used in class.

Demo 9.1: Namespaces/Demos/Artist.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema targetNamespace="https://www.webucator.com/Artist"
3.      xmlns:xs="http://www.w3.org/2001/XMLSchema"
4.      xmlns="https://www.webucator.com/Artist">
5.      <xs:element name="Title" type="xs:string"/>
6.      <xs:element name="FirstName" type="xs:string"/>
7.      <xs:element name="LastName" type="xs:string"/>
8.      <xs:element name="Name">
9.          <xs:complexType>
10.             <xs:sequence>
11.                 <xs:element ref="Title"/>
12.                 <xs:element ref="FirstName"/>
13.                 <xs:element ref="LastName"/>
14.             </xs:sequence>
15.         </xs:complexType>
16.     </xs:element>
17.     <xs:element name="Artist">
18.         <xs:complexType>
19.             <xs:sequence>
20.                 <xs:element ref="Name"/>
21.             </xs:sequence>
22.             <xs:attribute name="BirthYear" type="xs:gYear" use="required"/>
23.         </xs:complexType>
24.     </xs:element>
25. </xs:schema>
```

Evaluation Copy

This schema would be invalid if the `xmlns="http://www.webucator.com/Artist"` attribute were removed. That's because the `Name` and `Artist` element declarations have child elements that reference elements declared in this schema. We can only reference elements that are declared globally in namespaces used in the document (as indicated by the `xmlns` attributes).

Instance documents of this XML schema would take the `xmlns` and `xsi:schemaLocation` attributes. Again, the `xmlns` attribute allows global elements declared in the specified namespace to be used in this instance. The `xsi:schemaLocation` attribute is used to point to the schema associated with a namespace. Its value is the namespace name and the path to the schema separated by a space.

EVALUATION COPY: Not to be used in class.

Demo 9.2: Namespaces/Demos/MichaelJackson.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <Artist BirthYear="1958"
3.       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.       xmlns="https://www.webucator.com/Artist"
5.       xsi:schemaLocation="https://www.webucator.com/Artist Artist.xsd">
6.       <Name>
7.           <Title>Mr.</Title>
8.           <FirstName>Michael</FirstName>
9.           <LastName>Jackson</LastName>
10.      </Name>
11.  </Artist>
```

EVALUATION COPY: Not to be used in class.

9.4. Default Namespaces

Evaluation Copy

Let's consider the `Artist.xsd` schema for a moment. The first thing to remember is that the XML schema document is itself an instance document. The `xmlns` attributes are used to indicate what namespaces are associated with this instance.

The `xs:schema` element looks like this:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.webucator.com/Artist"
  targetNamespace="http://www.webucator.com/Artist">
```

The `xmlns:xs` attribute indicates that global elements, attributes, types, and groups in the XML Schema namespace can be used in this document, but they must be qualified with the `xs:` prefix.

Note that you could change this prefix to `bob:` or `jill:`, though `xs:` and `xsd:` are the most commonly used for the XML Schema namespace.

The second `xmlns` attribute indicates that global elements, attributes, types, and groups in the Artist namespace can be used in this document without a prefix (i.e, unqualified). The Artist namespace then is the default namespace. There can only be one default namespace in an instance document.

EVALUATION COPY: Not to be used in class.

Now let's look at the `MichaelJackson.xml` instance document.

```
<Artist BirthYear="1958"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.webucator.com/Artist"
  xsi:schemaLocation="http://www.webucator.com/Artist Artist.xsd">
```

It uses two namespaces: http://www.w3.org/2001/XMLSchema-instance, which we will discuss later, and http://www.webucator.com/Artist.

Notice that none of the elements or attributes in the document is qualified (i.e, prefixed). That is because they all belong to the Artist namespace, which is the default namespace.

The sample code below shows how this same document would look without a default namespace.

Demo 9.3: Namespaces/Demos/MichaelJacksonQualified.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <art:Artist BirthYear="1958"
3.              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.              xmlns:art="https://www.webucator.com/Artist"
5.              xsi:schemaLocation="https://www.webucator.com/Artist Artist.xsd">
6.      <art:Name>
7.          <art:Title>Mr.</art:Title>
8.          <art:FirstName>Michael</art:FirstName>
9.          <art:LastName>Jackson</art:LastName>
10.     </art:Name>
11. </art:Artist>
```

Evaluation Copy

Generally, it only makes sense to use qualifiers when using more than one namespace.

EVALUATION COPY: Not to be used in class.

9.5. Locally Declared Elements and Attributes

By default, locally declared elements and attributes in an instance document do not need to be qualified. This can be changed in the schema by including the `elementFormDefault` and `attributeFormDefault` attributes of the `xs:schema` element with the value of `"qualified"`.

EVALUATION COPY: Not to be used in class.

First, let's look at the `ArtistLocal.xsd` schema, which has been modified to make all elements but the `Artist` element locally declared.

Demo 9.4: Namespaces/Demos/ArtistLocal.xsd

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xs:schema targetNamespace="https://www.webucator.com/Artist"
3.      xmlns:xs="http://www.w3.org/2001/XMLSchema"
4.      xmlns="https://www.webucator.com/Artist"
5.      elementFormDefault="unqualified"
6.      attributeFormDefault="unqualified">
7.      <xs:element name="Artist">
8.          <xs:complexType>
9.              <xs:sequence>
10.                 <xs:element name="Name">
11.                     <xs:complexType>
12.                         <xs:sequence>
13.                             <xs:element name="Title" type="xs:string"/>
14.                             <xs:element name="FirstName" type="xs:string"/>
15.                             <xs:element name="LastName" type="xs:string"/>
16.                         </xs:sequence>
17.                     </xs:complexType>
18.                 </xs:element>
19.             </xs:sequence>
20.             <xs:attribute name="BirthYear" type="xs:gYear" use="required"/>
21.         </xs:complexType>
22.     </xs:element>
23. </xs:schema>
```

Evaluation Copy

Notice that the `elementFormDefault` and `attributeFormDefault` attributes are set to `"unqualified"`.

Now let's take a look at an instance document of this schema.

EVALUATION COPY: Not to be used in class.

Demo 9.5: Namespaces/Demos/MichaelJacksonLocal.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <art:Artist BirthYear="1958"
3.       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.       xmlns:art="https://www.webucator.com/Artist"
5.       xsi:schemaLocation="https://www.webucator.com/Artist ArtistLocal.xsd">
6.       <Name>
7.           <Title>Mr.</Title>
8.           <FirstName>Michael</FirstName>
9.           <LastName>Jackson</LastName>
10.      </Name>
11.  </art:Artist>
```

When using unqualified locals, it is not valid to use a default namespace. The schema processor must know that these elements are locally declared within a specific namespace. If a default namespace were used, the schema processor would not be able to differentiate between locally declared and globally declared elements. Therefore, we use the `art:` prefix to qualify the Artist namespace.

Evaluation Copy

EVALUATION COPY: Not to be used in class.

9.6. Qualified Locals

If the `elementFormDefault` and `attributeFormDefault` attributes in the `xs:schema` element are set to `"qualified"` all locals must be qualified with a prefix.

EVALUATION COPY: Not to be used in class.

Demo 9.6: Namespaces/Demos/ArtistLocalQualified.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3.                     xmlns="https://www.webucator.com/Artist"
4.                     targetNamespace="https://www.webucator.com/Artist"
5.                     elementFormDefault="qualified"
6.                     attributeFormDefault="qualified">
7.       <xs:element name="Artist">
8.           <xs:complexType>
9.               <xs:sequence>
10.                  <xs:element name="Name">
11.                      <xs:complexType>
12.                          <xs:sequence>
13.                              <xs:element name="Title" type="xs:string"/>
14.                              <xs:element name="FirstName" type="xs:string"/>
15.                              <xs:element name="LastName" type="xs:string"/>
16.                          </xs:sequence>
17.                      </xs:complexType>
18.                  </xs:element>
19.              </xs:sequence>
20.              <xs:attribute name="BirthYear" type="xs:gYear" use="required"/>
21.          </xs:complexType>
22.      </xs:element>
23.  </xs:schema>
```

Evaluation Copy

Demo 9.7: Namespaces/Demos/MichaelJacksonLocalQualified.xml

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <art:Artist art:BirthYear="1958"
3.       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.       xmlns:art="https://www.webucator.com/Artist"
5.       xsi:schemaLocation=
6.           "https://www.webucator.com/Artist ArtistLocalQualified.xsd">
7.       <art:Name>
8.           <art:Title>Mr.</art:Title>
9.           <art:FirstName>Michael</art:FirstName>
10.          <art:LastName>Jackson</art:LastName>
11.      </art:Name>
12.  </art:Artist>
```

The result of qualifying all locals is that instance authors do not have to differentiate between local and global declarations. They simply prefix all elements and attributes with a qualifier. This has two major advantages over using unqualified locals.

EVALUATION COPY: Not to be used in class.

- Clarity - it is easy to tell which namespace each element belongs to.
- Flexibility - the schema author can mix global and local declarations without worrying that the instance author will get confused. As both local and global declarations require prefixes, the instance author doesn't need to know how an element or attribute is declared.

EVALUATION COPY: Not to be used in class.

9.7. The XMLSchema-instance Namespace

The XMLSchema-instance namespace contains only four attributes. Here is a simplified version of the schema.

Demo 9.8: Namespaces/Demos/XMLSchema-instance.xsd

```
1.   <?xml version='1.0'?>
2.   <xs:schema targetNamespace="http://www.w3.org/2001/XMLSchema-instance"
3.             xmlns:xs="http://www.w3.org/2001/XMLSchema">
4.     <xs:attribute name="nil"/>
5.    <xs:attribute name="type"/>
6.    <xs:attribute name="schemaLocation"/>
7.    <xs:attribute name="noNamespaceSchemaLocation"/>
8.   </xs:schema>
```

Evaluation Copy

By specifying that an XML document uses the XMLSchema-instance namespace, the instance author gets access to the four attributes declared above. We have now used all four of these attributes.

- `xsi:nil` is used to specify that an element has no value. It is covered in Nil Values (see page 40).
- `xsi:schemaLocation` is used to specify the location of a schema for a particular namespace.
- `xsi:noNamespaceSchemaLocation` is used to specify the location of a schema when no namespace is used.
- `xsi:type` is infrequently used to specify that the element in the instance is of a different type than the one declared in the schema for that element.

EVALUATION COPY: Not to be used in class.

EVALUATION COPY: Not to be used in class.

9.8. Using Multiple Namespaces

Often it makes sense to use multiple namespaces for a single instance document. As an example, take a look at the following document.

Demo 9.9: Namespaces/Demos/TheGirlIsMine.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <Song    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.              xmlns="https://www.webucator.com/Song"
4.              xmlns:art="https://www.webucator.com/Artist"
5.              xsi:schemaLocation="https://www.webucator.com/Song Song.xsd">
6.      <Title>The Girl Is Mine</Title>
7.      <Year>1983</Year>
8.      <Artists>
9.          <art:Artist BirthYear="1958">
10.             <art:Name>
11.                 <art:Title>Mr.</art:Title>
12.                 <art:FirstName>Michael</art:FirstName>
13.                 <art:LastName>Jackson</art:LastName>
14.             </art:Name>
15.         </art:Artist>
16.         <art:Artist BirthYear="1942">
17.             <art:Name>
18.                 <art:Title>Mr.</art:Title>
19.                 <art:FirstName>Paul</art:FirstName>
20.                 <art:LastName>McCartney</art:LastName>
21.             </art:Name>
22.         </art:Artist>
23.     </Artists>
24. </Song>
```

Evaluation Copy

The default namespace is the Song namespace. The Artist namespace is qualified with the `art:` prefix. Locally declared elements (there are none) and attributes (e.g, `BirthYear`) are unqualified.

Let's look at `Song.xsd`.

EVALUATION COPY: Not to be used in class.

Demo 9.10: Namespaces/Demos/Song.xsd

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3.                     xmlns:art="https://www.webucator.com/Artist"
4.                     xmlns="https://www.webucator.com/Song"
5.                     targetNamespace="https://www.webucator.com/Song">
6.       <xs:import namespace="https://www.webucator.com/Artist"
7.           schemaLocation="Artist.xsd"/>
8.       <xs:element name="Title" type="xs:string"/>
9.       <xs:element name="Year" type="xs:gYear"/>
10.      <xs:element name="Artists">
11.          <xs:complexType>
12.              <xs:sequence>
13.                  <xs:element ref="art:Artist" maxOccurs="unbounded"/>
14.              </xs:sequence>
15.          </xs:complexType>
16.      </xs:element>
17.      <xs:element name="Song">
18.          <xs:complexType>
19.              <xs:sequence>
20.                  <xs:element ref="Title"/>
21.                  <xs:element ref="Year"/>
22.                  <xs:element ref="Artists"/>
23.              </xs:sequence>
24.          </xs:complexType>
25.      </xs:element>
26.  </xs:schema>
```

Evaluation Copy

By importing the Artist namespace with `xs:import` and specifying that elements in that namespace can be referenced with the `xmlns:art` attribute of `xs:schema`, elements and attributes in the Artist namespace are accessible to this schema.

Conclusion

Namespaces can be tricky. Often the best approach is to start by mapping prefixes to all namespaces and then switching to a default namespace only if one namespace is clearly predominant and it becomes a bother to continue to type the same prefix over an over again. Generally, though, you get used to it, and a good XML editor will help out.

EVALUATION COPY: Not to be used in class.