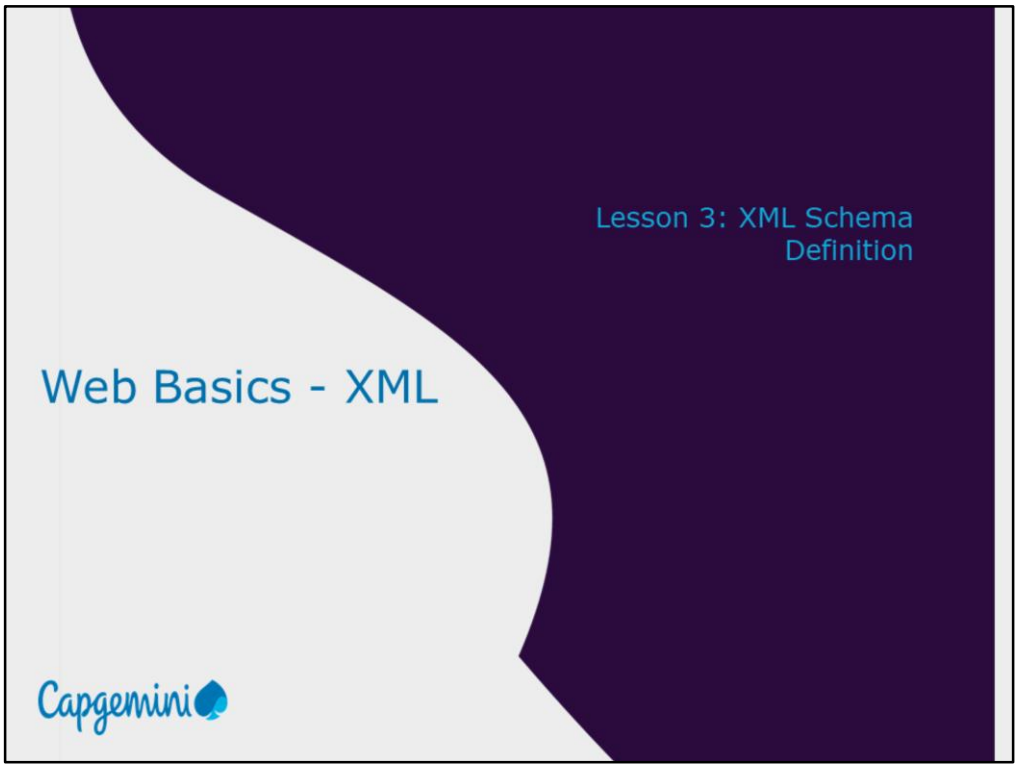
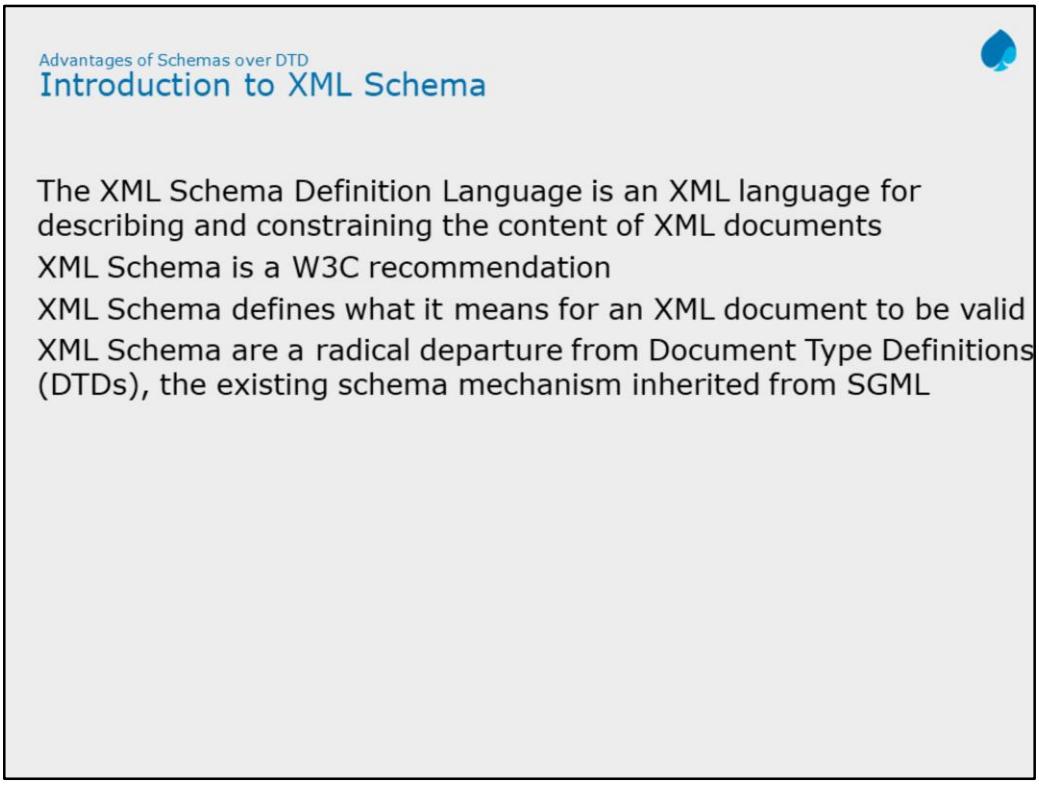

Lesson 3: XML Schema Definition
Web Basics - XML
Capgemini

Lesson Objectives

In this lesson, you will learn about:

- Advantages of Schema over DTD
- Method to write a schema definition for an XML file
- Data types used in schemas
- Simple and Complex type of elements
- Restrictions on XSD elements
- Indicator – Order, Occurrence, and Group



What You Should Already Know

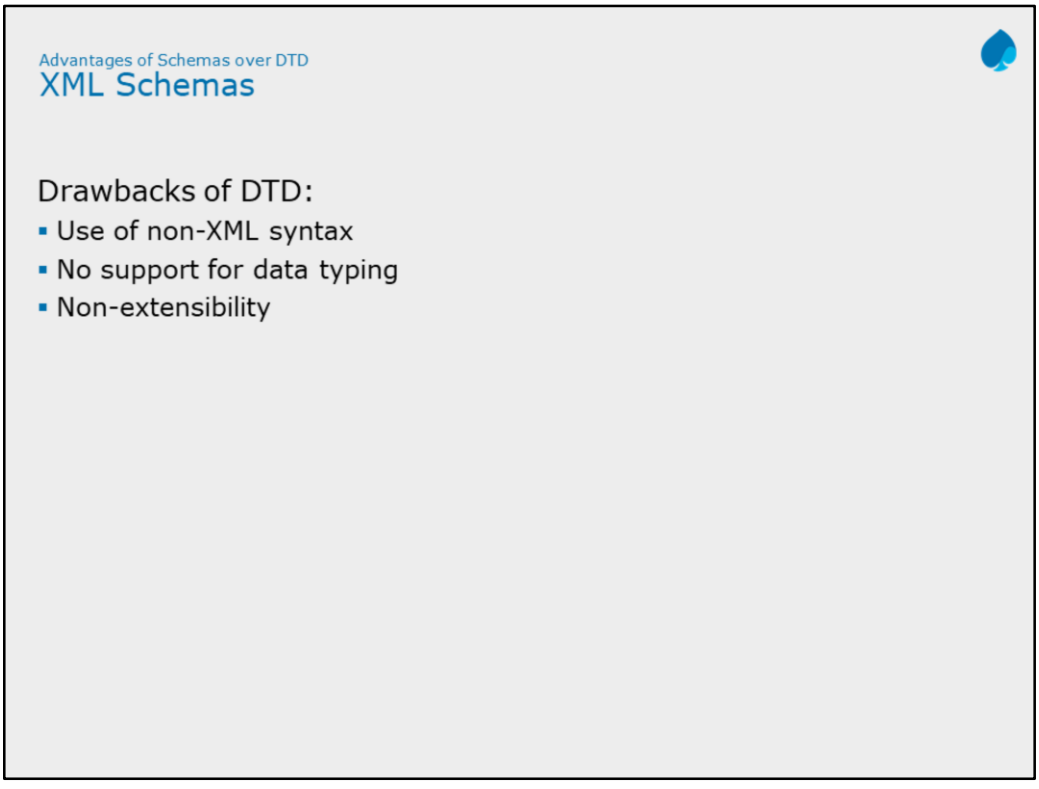
Before you continue, you should have a basic understanding of the following:

- HTML/ XHTML
- XML and XML Namespaces
- A basic understanding of DTD

Introduction to Schema:

An XML document is essentially a structured medium for storing information. In order to assess the validity of a XML document, you need to establish exactly to which structure the information within the document must adhere. This is accomplished with **schema**.

A schema describes the arrangement of **markup** and **character data** within a valid XML document. It describes the grammar, vocabulary, structure, datatypes, etc. of a XML document. A traditional schema solution is DTD. We are already familiar with DTD and XML namespaces.



Advantages of Schemas over DTD:

XML Data Modeling with DTDs:

We are familiar with DTD for XML document. DTDs originated from SGML and provide a standard mechanism for validating SGML documents.

XML is subset of SGML. Hence it also uses the same approach. However, DTDs have some drawbacks. So a new approach for modeling XML data has come up. It is called as **XML schema**.

DTDs rely on a specialized syntax for describing the structure of XML vocabularies. This is a drawback of DTD. The question arises, why is it necessary to learn a specialized syntax, when XML itself provides a suitable syntax for describing data of any kind.

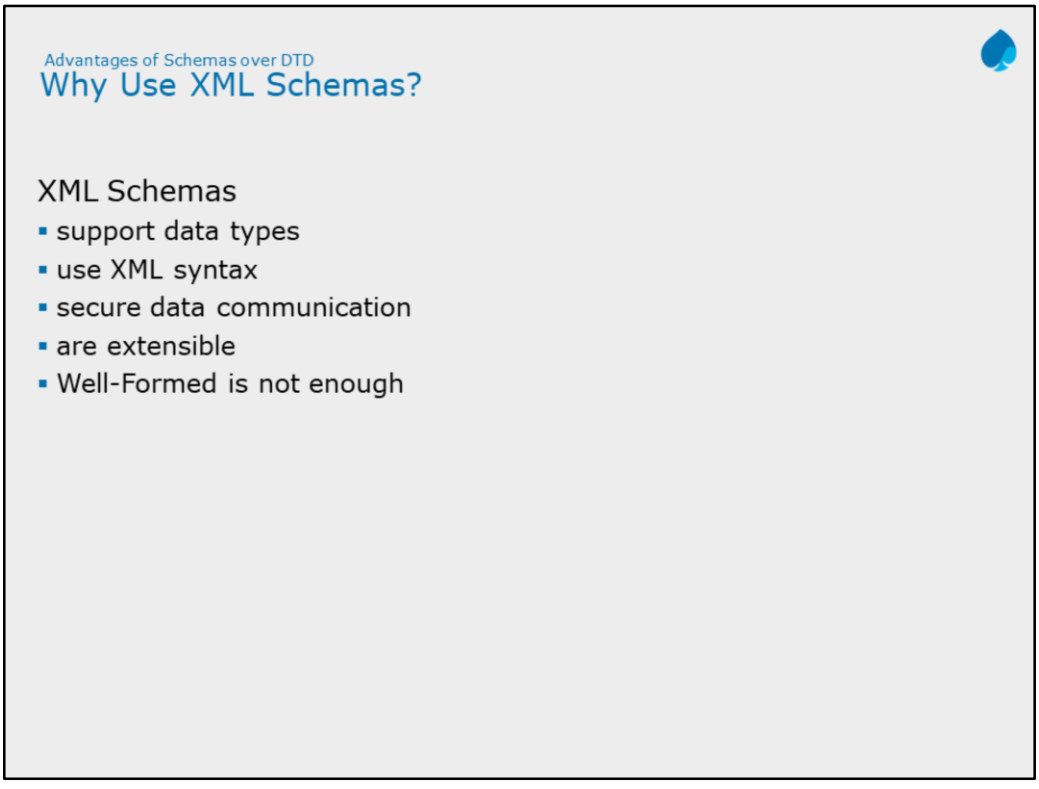
Disadvantages of DTD:

Good DTDs are difficult to write.

There is no provision for inheritance from one DTD to another.

DTDs do not provide support for namespaces.

It is limited in its descriptive powers.



Why Use XML Schemas?

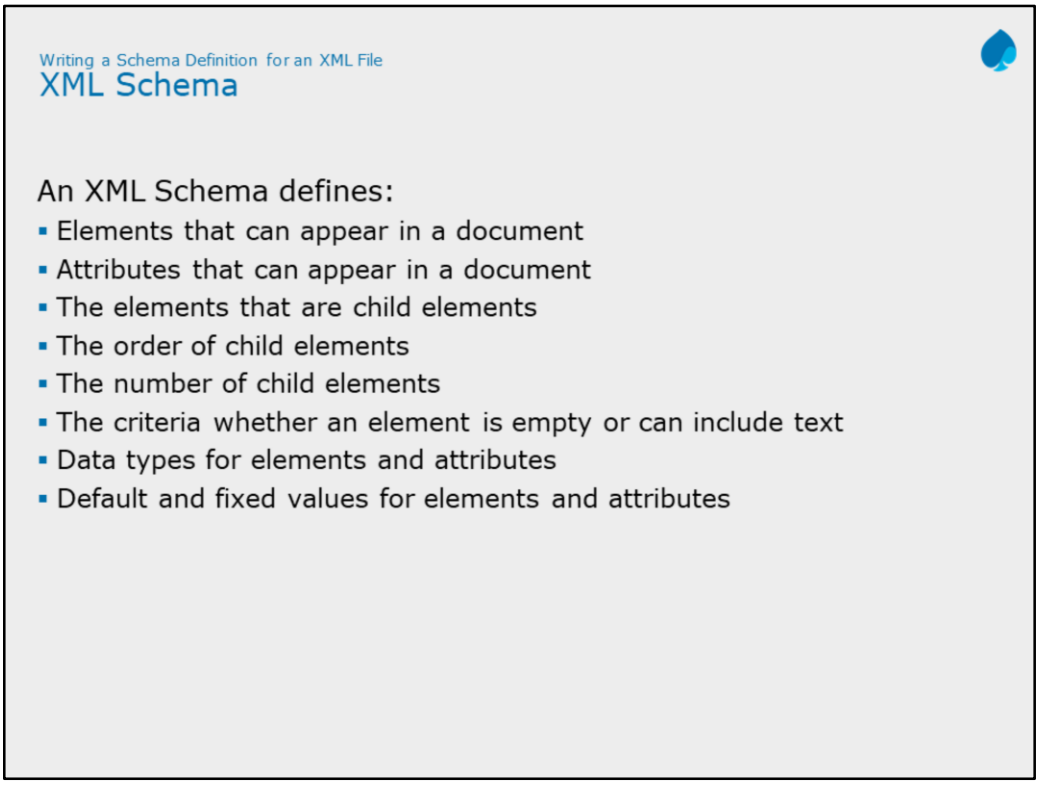
As mentioned earlier XML Schemas have advantages over using DTD. Let us now se some more reasons why we should use XML Schemas.

Support Data Types: XML Schemas have support for datatypes. This makes it simple to describe allowable document content, to validate the data correctness. It is also easy to work with data from database, defining restrictions on data and/or data formats. It also allows conversion of data between different data types.

Use of XML Syntax: When writing XML Schemas you follow XML syntax. Hence you can use the XML Editors and Parsers to work with the Schema files. In addition to this you also do not need to learn a different language.

Secure Data Communication**:** During data transfer it is essential that both dispatcher and receiver of the data have the same understanding about the transferred content. The dispatcher will have to depict the data in such a way that it is understood by the receiver**.**

Are extensible: XMl schemas can be inherited i.e one XML schema can be extended by another XML schema. You can also create your own datatypes from standard datatypes.



Advantages of Schemas over DTD:

Example:

Consider the following example:

<!ELEMENT pin-code #PCDATA>

Now, consider the following statement:

<pin-code>ABC-123444-hhh</pin-code>

It is both well-formed and valid even though ABC-123444-hhh certainly does not represent a pin code in any form.

The data-type constraints available in schemas can allow the schema designer to limit the content of the pin-code element to a six digits number, for example, 400090.

XML Schemas are the successors of DTDs.

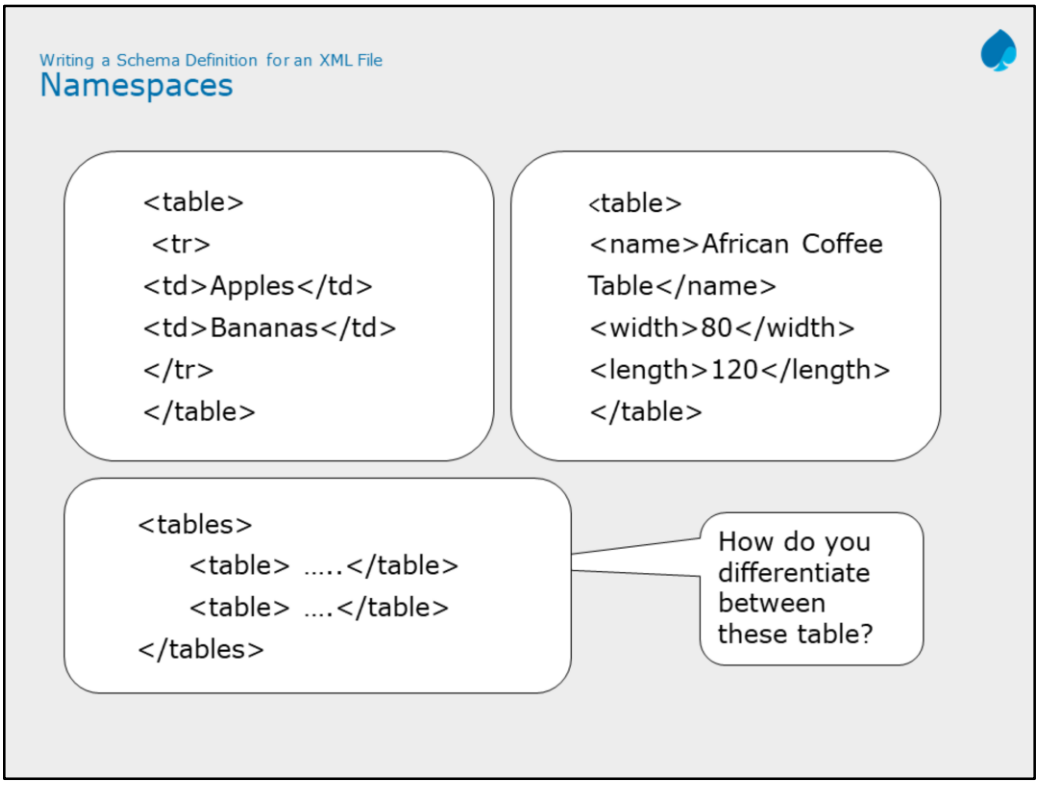
Writing a Schema Definition for an XML File

Namespaces

XML Namespaces provide a method to avoid element name conflicts

Name Conflicts: In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications

XML Namespaces provides a method to avoid element name conflicts



Namespaces:

If the XML fragments in the above slide were added together, then there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.

An XML parser will not know how to handle these differences.

Writing a Schema Definition for an XML File

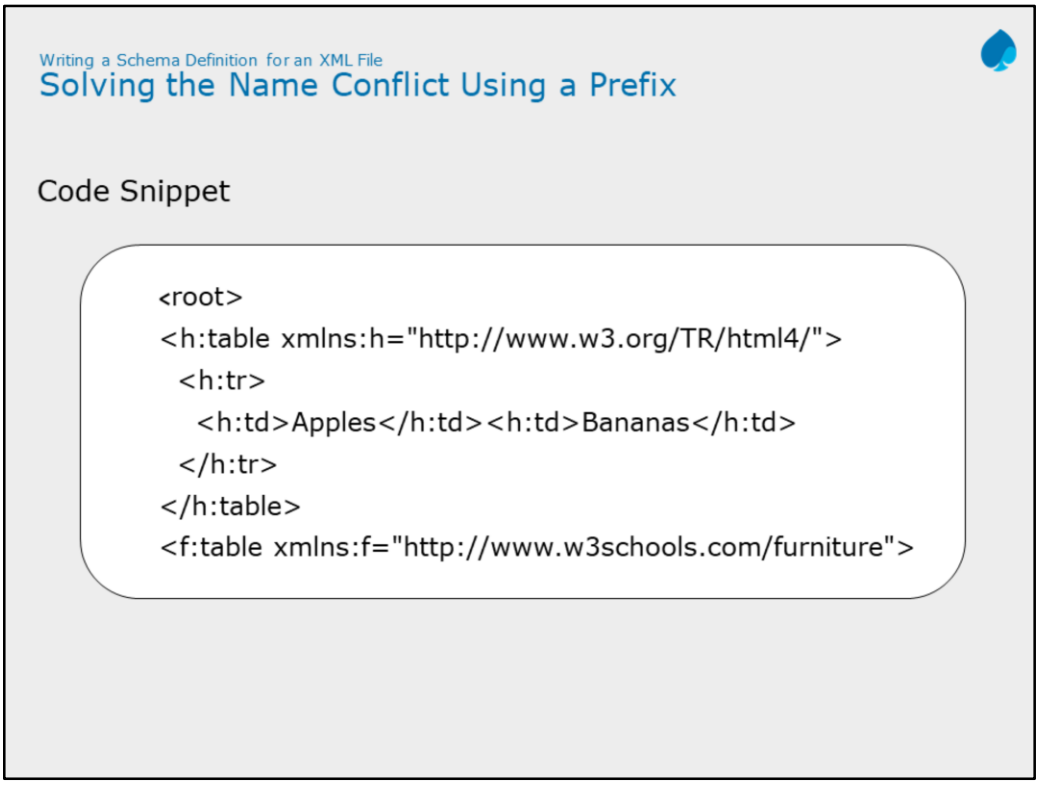
Namespaces

The namespace attribute is placed in the start tag of an element and has the following syntax:

```
xmlns:namespace-
prefix="namespace"
```

The W3C namespace specification states that the namespace itself should be an Uniform Resource Identifier (URI)

When a namespace is defined in the start tag of an element, all child elements with the same prefix are associated with the same namespace



Solving the Name Conflict using a Prefix:

In the example in the above slide, there will be no conflict because the two <table> elements have different names. This XML carries information about an HTML table, and a piece of furniture.

When using prefixes in XML, a so-called **namespace** for the prefix must be defined.

The namespace is defined by the **xmlns attribute** in the start tag of an element.

The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".

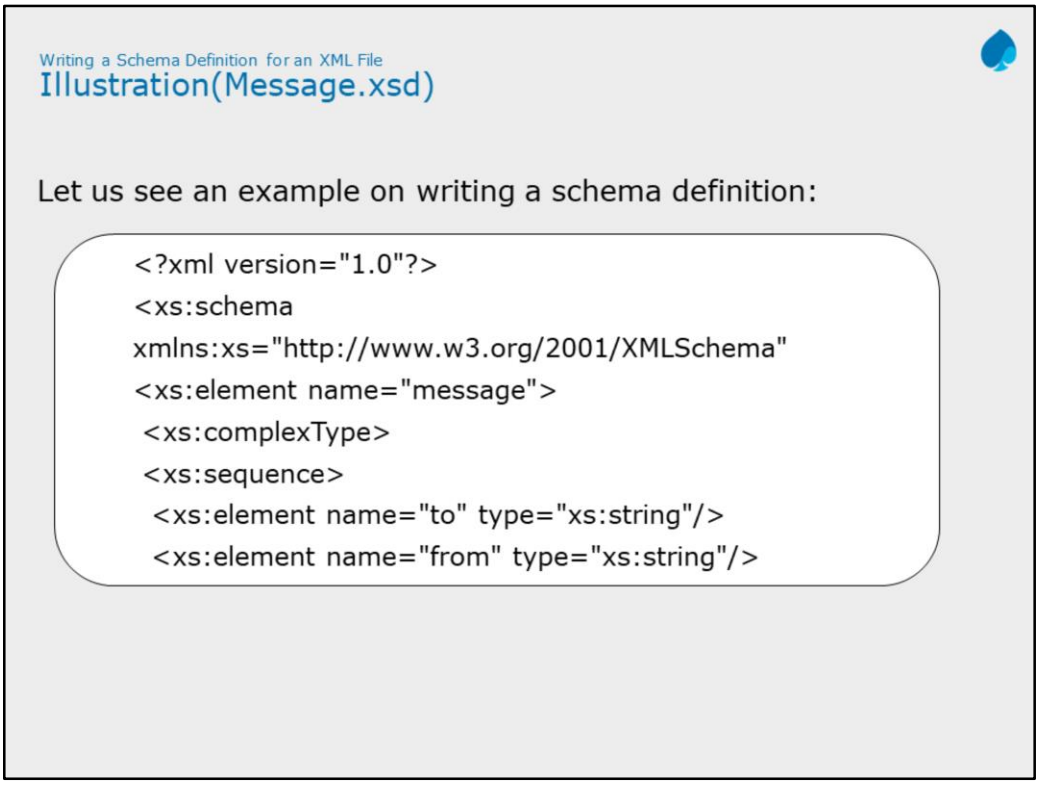
In the example on the above slide, the xmlns attribute in the <table> tag gives the h: and f: prefixes a qualified namespace.

When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

SoWriting a Schema Definition for an XML File lving the Name Conflict Using a Prefix

Code Snippet continued

```
<f:name>African  Coffee Table</f:name>
 <f:width>80</f:width><f:length>120</f:length>
 </f:table>
  </root>
```



Creating Schema Document:

The <schema> element is the root element of every XML Schema. The <schema> element contains some attributes. A schema declaration often looks like something as shown in the above slide.

xmlns:xs= http://www.w3.org/2001/XMLSchema

It implies that the elements and data types used in the schema are from "http://www.w3.org/2001/XMLSchema" namespace. It also signifies that any elements and datatypes referred from here should have the prefix “xs”.

Some more optional attributes:

targetNamespace="patniNamespace"

This value is a unique identifier. The value could be anything. Place this attribute at the top of the XSD means all entities are part of the namespace

xmlns=" http://www.w3.org/2001/XMLSchema "

It indicates that the default namespace is "http://www.w3.org/2001/XMLSchema".

elementFormDefault="qualified"

It signifies that any elements used by the XML instance document that were declared in this schema must be qualified by namespace.

Writing a Schema Definition for an XML File

Illustration(Message.xsd)

Code Snippet continued

```
<xs:element name="subject" type="xs:string"/>
<xs:element name="text" type="xs:string"/>
<xs:attribute name="priority" type="xs:string"
use="required"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Writing a Schema Definition for an XML File:

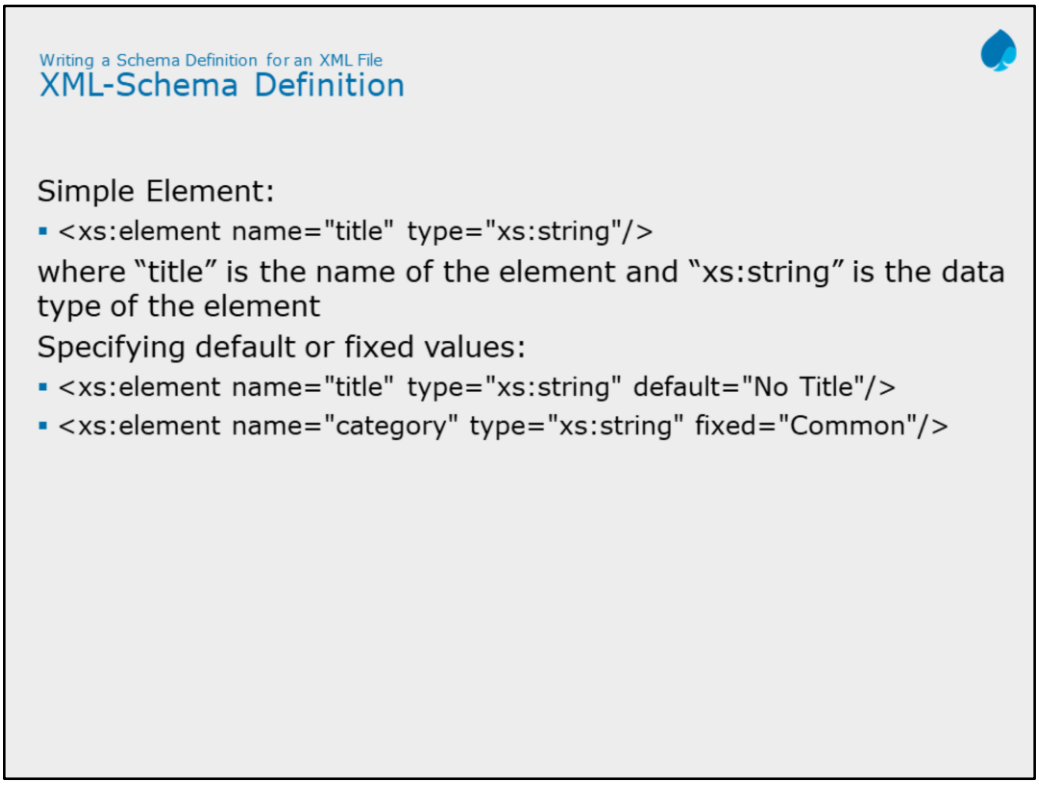
Using XSD in XML Document:

xmlns=" http://www.w3.org/2001/XMLSchema" indicates the default namespace declaration which tells the schema-validator that all the elements used in this XML document are declared in the "http://www.w3.org/2001/XMLSchema" namespace.

When you have the XML Schema Instance namespace available, that is "http://www.w3.org/2001/XMLSchema-instance ", you can use the schemaLocation attribute. This attribute has two values:

The first value is the namespace to use.

The second value is the location of the XML schema to use for that namespace "xsi:schemaLocation="message.xsd "



Writing a Schema Definition for an XML File:

What is a Simple Element?

A simple element is an XML element that can contain only text. It cannot contain any other element or attribute. The text can be of many different types. It can be one of the types that are included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.

You can also add restrictions to a data type in order to limit its content, and you can make it mandatory for the data to match a defined pattern.

Default and Fixed Values for Simple Elements:

Simple elements may have a default value or a fixed value specified.

A default value is automatically assigned to the element when no other value is specified. In the above example, default value is "No Title".

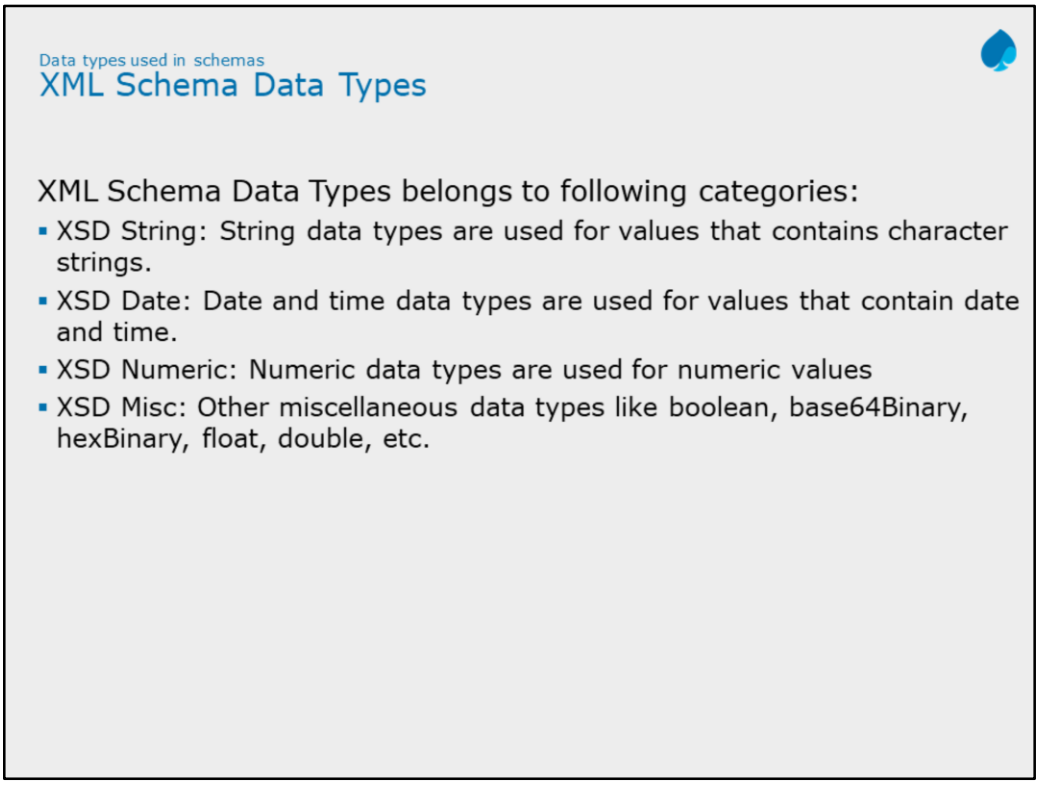
A fixed value is also automatically assigned to the element, and you cannot specify another value. In the above example, the fixed value is "common".

Here are some XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

Here are the corresponding simple element definitions:

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```



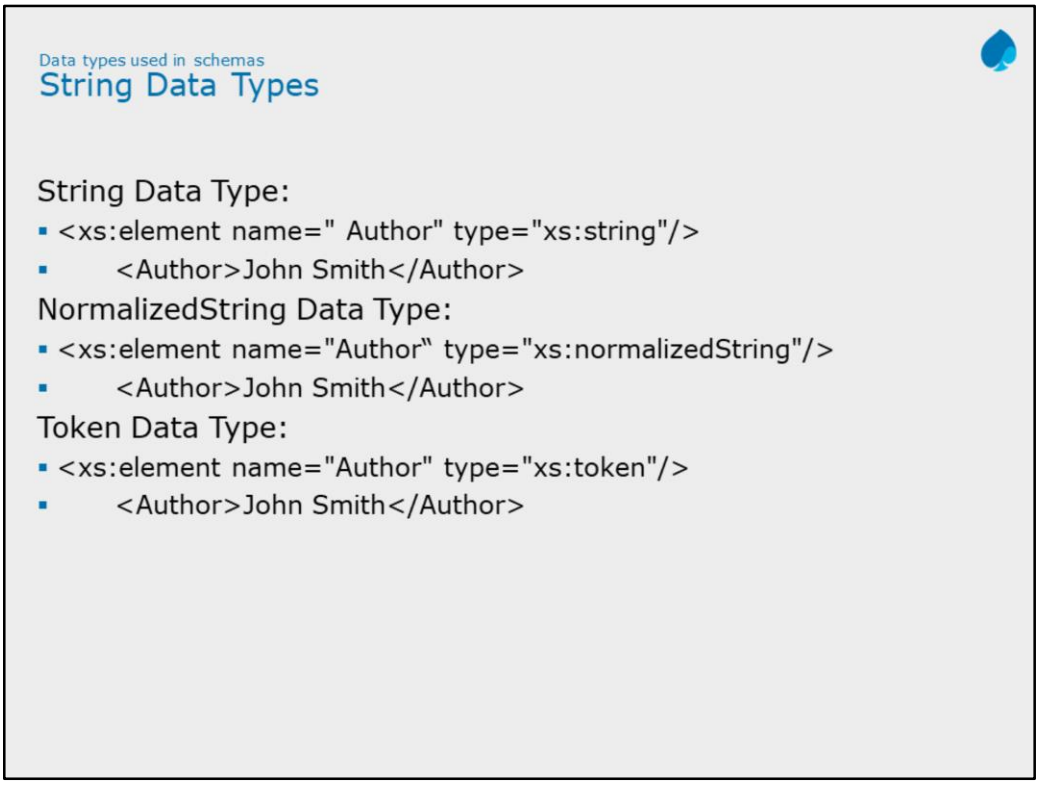
XML Schema Vocabulary:

XML Schema has support for data types.

With the support for data types it is easier:

- to describe permissible document content.
- to validate the correctness of data.
- to work with data from a database.
- to define data patterns (data formats).
- to convert data between different data types.

Apart from the above advantages, XML schemas are **extensible**. It implies that you can reuse your Schema in other Schemas and also reference multiple schemas from the same document.



String Data Types:

String Data Type:

The string data type can contain characters, line feeds, carriage returns, and tab characters.

NormalizedString Data Type

The normalizedString data type is derived from the String data type.

The normalizedString data type also contains characters, but the XML processor will remove line feeds, carriage returns, and tab characters.

Token Data Type

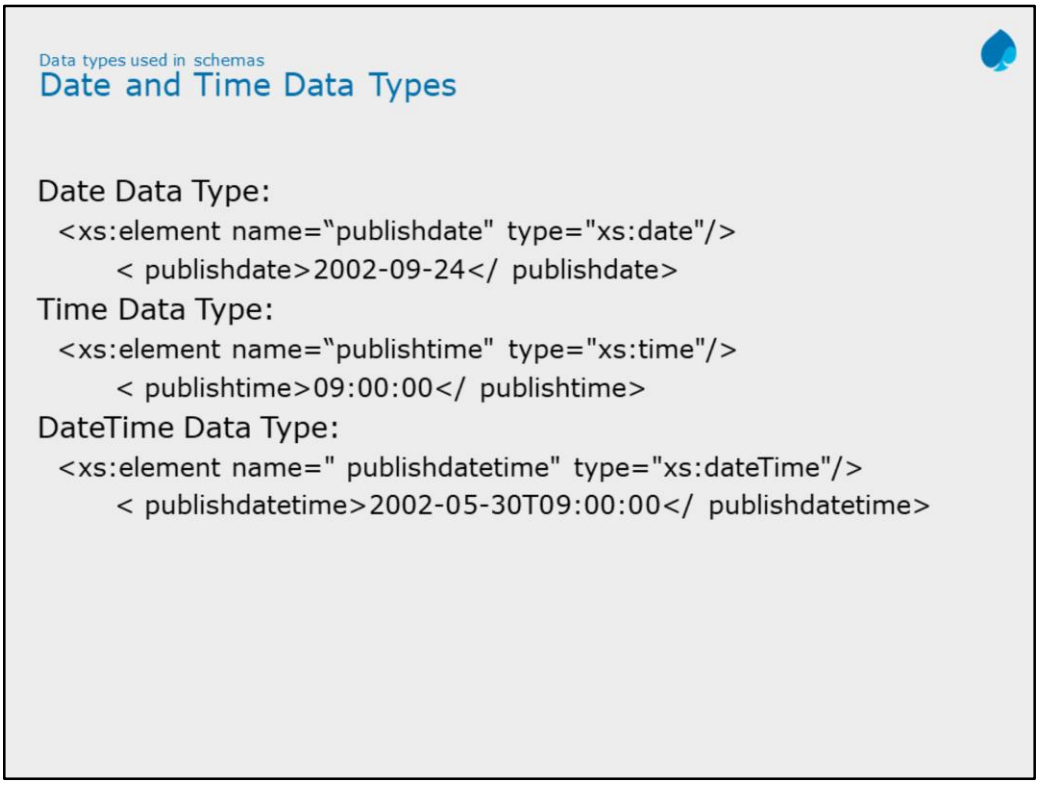
The token data type is also derived from the String data type.

The token data type also contains characters, but the XML processor will remove line feeds, carriage returns, tabs, leading and trailing spaces, and multiple spaces.

Restrictions on String Data Types

Restrictions that can be used with String data types:

- enumeration
- length
- maxLength
- minLength
- pattern
- whiteSpace



Date and Time Data Types:

Date Data Type:

The date data type is used to specify a date.

The date is specified in the following form "YYYY-MM-DD", where YYYY indicates the year, MM indicates the month, and DD indicates the day.

Time Data Type:

The time data type is used to specify a time.

The time is specified in the following form "hh:mm:ss", where hh indicates the hour, mm indicates the minute, and ss indicates the second.

DateTime Data Type:

The dateTime data type is used to specify a date and a time.

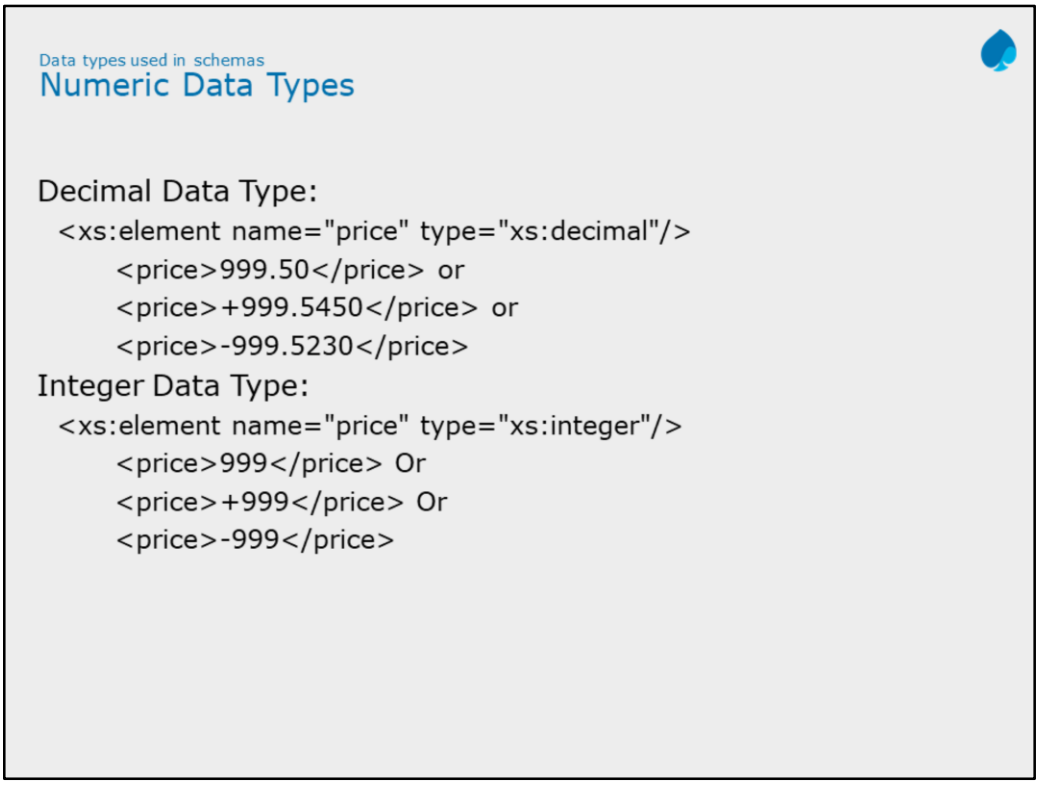
The dateTime is specified in the following form "YYYY-MM-DDThh:mm:ss", where YYYY indicates the year, MM indicates the month, DD indicates the day, T indicates the start of the required time section, hh indicates the hour, mm indicates the minute, and ss indicates the second.

Note: All components are required in all categories!

Restrictions on Date Data Types

Restrictions that can be used with Date data types:

Enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern whiteSpace



Numeric Data Types:

Decimal Data Type: The decimal data type is used to specify a numeric value.

Integer Data Type: The integer data type is used to specify a numeric value without a fractional component.

Numeric Data Types: All the data types below derive from the Decimal data type (except for decimal itself)!

Byte: A signed 8-bit integer

Decimal: A decimal valueintA signed 32-bit integer

Integer: An integer valuelongA signed 64-bit integer

negativeInteger: An integer containing only negative values (..,-2,-1)

nonNegativeInteger: An integer containing only non-negative values (0,1,2,..)

nonPositiveInteger: An integer containing only non-positive values (..,-2,-1,0)

positiveInteger: An integer containing only positive values (1,2,..)

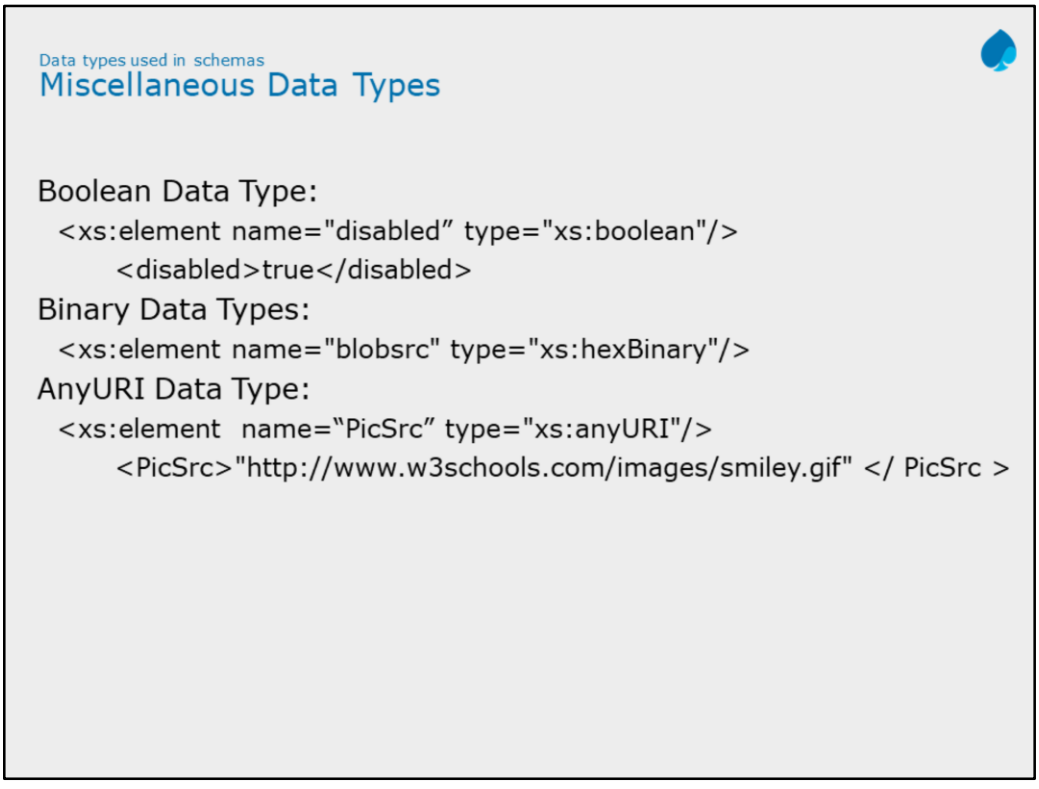
Short: A signed 16-bit integer

unsignedLong: An unsigned 64-bit integer

unsignedInt: An unsigned 32-bit integer

unsignedShort: An unsigned 16-bit integer

unsignedByte: An unsigned 8-bit integer



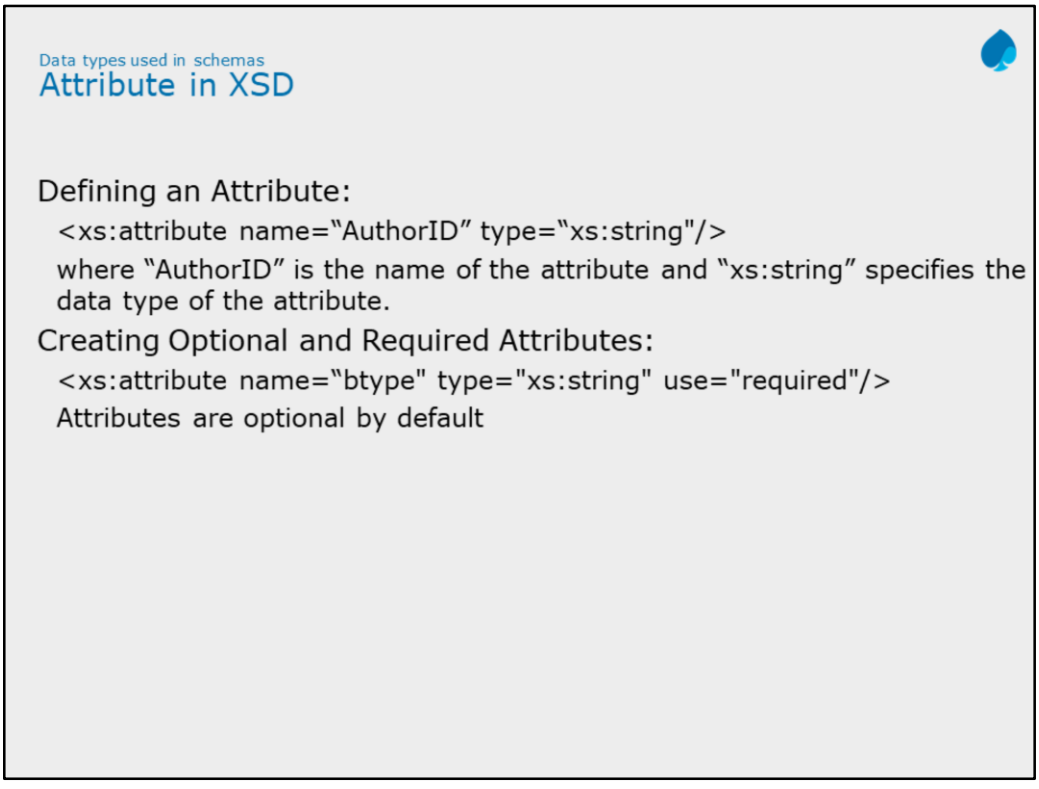
Miscellaneous Data Types:

Boolean Data Type: The boolean data type is used to specify a true or false value.

Binary Data Types: Binary data types are used to express binary-formatted data. We have two binary data types:

- base64Binary (Base64-encoded binary data)
- hexBinary (hexadecimal-encoded binary data)

AnyURI Data Type: The anyURI data type is used to specify a URI.



Attribute in XSD:

What is an Attribute?

Simple elements cannot have attributes. If an element has attributes, it is considered to be of complex type. However, the attribute itself is always declared as a simple type. This means that an element with attributes always has a complex type definition.

Example:

<Author AuthorID="A001">Smith</Author>

Following is a corresponding simple attribute definition:

<xs:attribute name="AuthorID" type="xs:string"/>

Default and Fixed Values for Attributes

Attributes may have a default value or a fixed value specified.

A default value is automatically assigned to the attribute when no other value is specified.

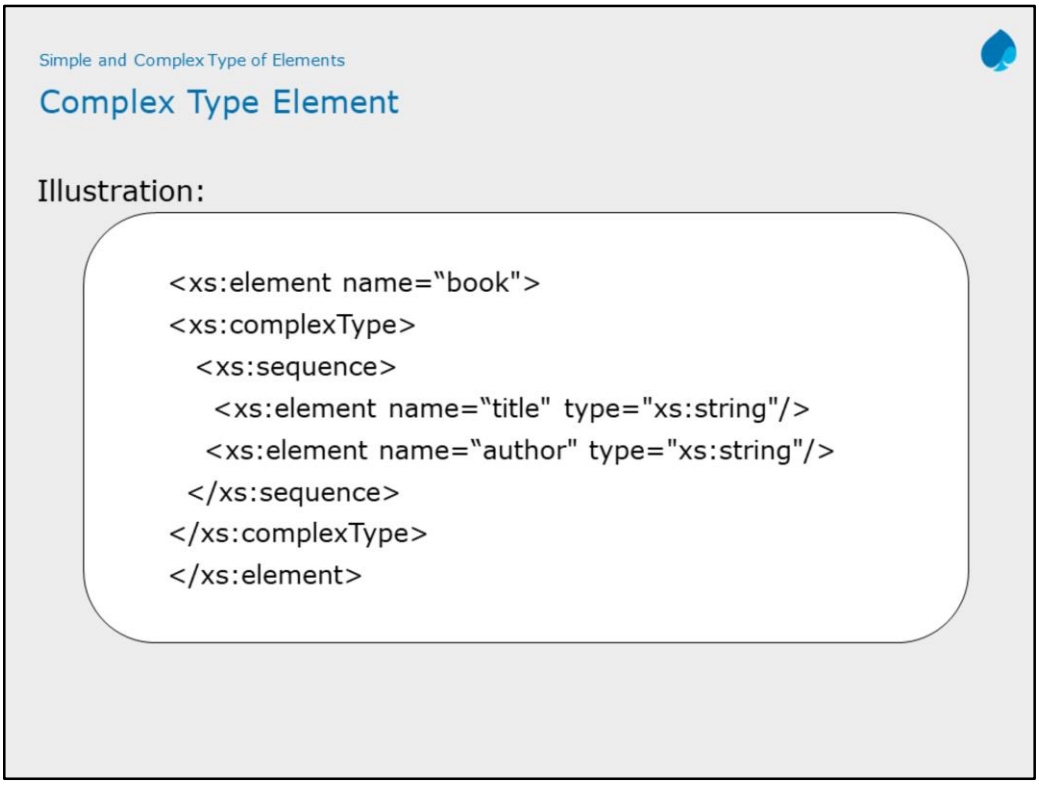
In the following example the default value is "UnKnown":

<xs:attribute name="AuthorID" type="xs:string" default=" UnKnown"/>

Optional and Required Attributes

Attributes are optional, by default. To specify that the attribute is required, use the "use" attribute:

<xs:attribute name="AuthorID" type="xs:string" use="required"/>



Complex Element:

Complex Element:

A complex element is an XML element that contains other elements and/or attributes.

There are four kinds of complex elements:

- Empty elements
- Elements that contain only other elements
- Elements that contain both – other elements and text
- Elements that contain text

Examples of Complex XML Elements:

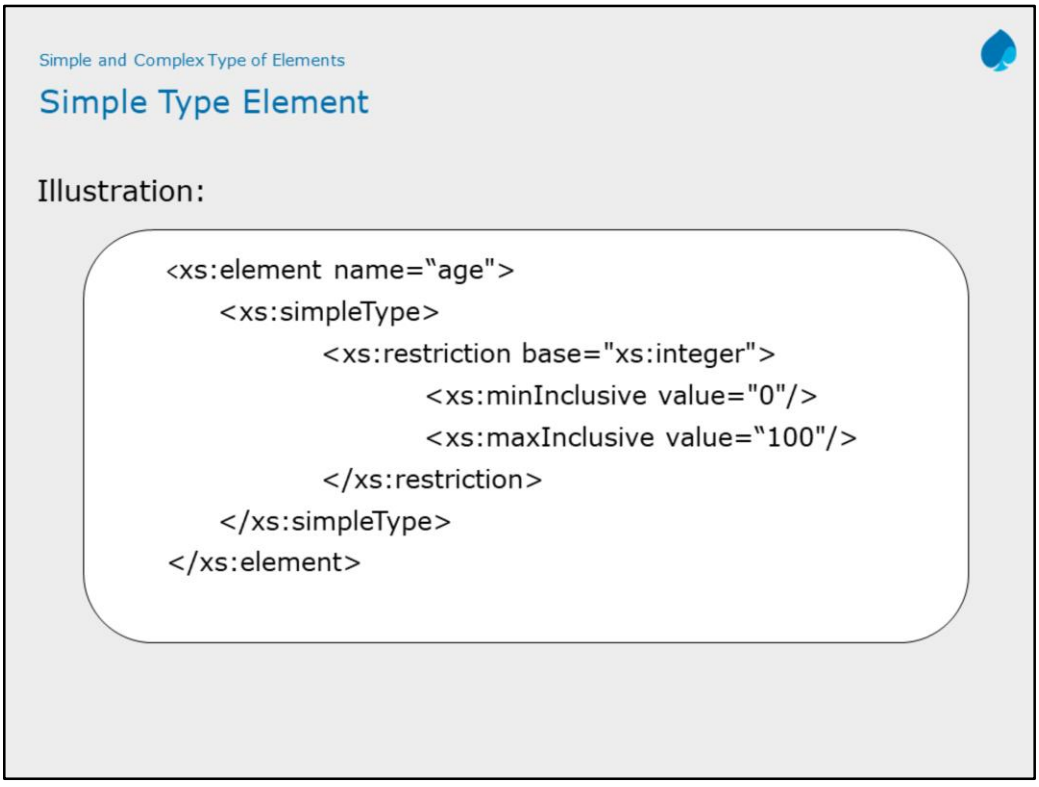
Example 1:

Consider a complex XML element, “product”, which is empty:

- <product pid="1345"/>

It can be declared as:

```
<xs:element name="product">
    <xs:complexType>
        <xs:attribute name="prodid" type="xs:positiveInteger"/>
    </xs:complexType>
</xs:element>
```



Simple Element:

Simple Element:

A simple element is an XML element that contain text

Examples of Simple XML Elements:

Example 1:

Consider a simple XML element, “EmpName”, which contains the name of employee:

- <EmpName>Smith</EmpName>

Restrictions on XSD Elements:

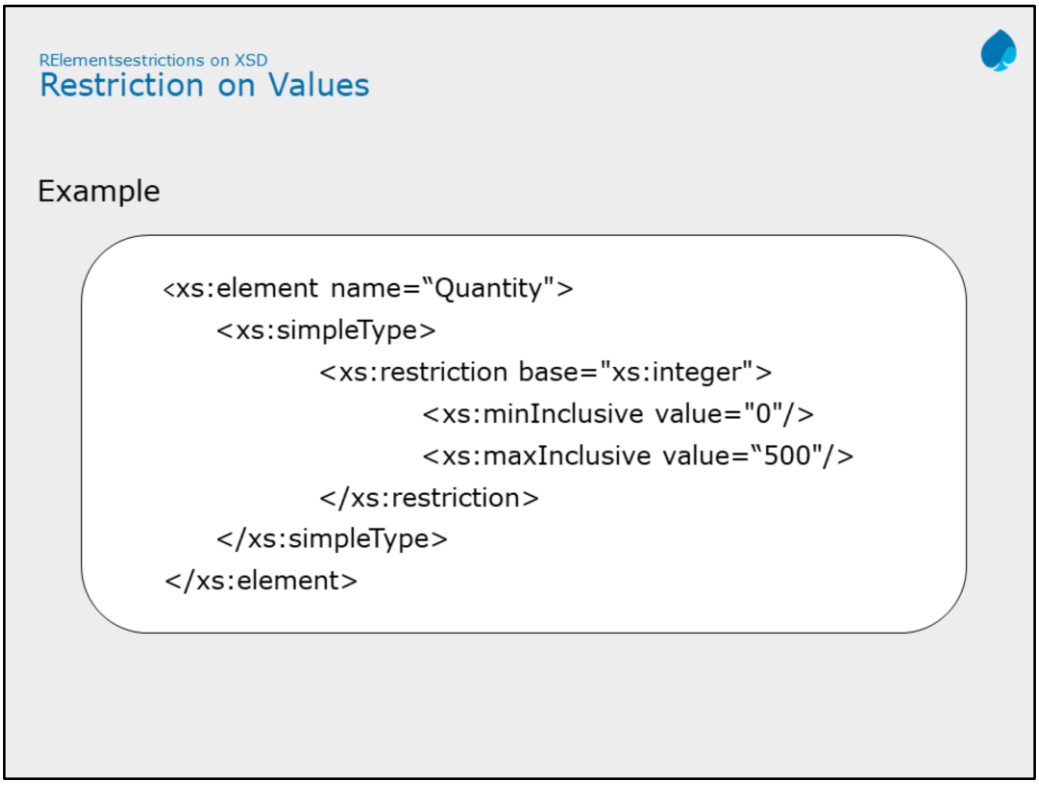
Restrictions on Content:

When an XML element or attribute has a datatype associated with , it puts a restriction on the element's or attribute's content. If an XML element is of type "xs:integer" and contains a string value "Nice Day", then the element will not validate.

With XML Schemas, you can also add your own restrictions to your XML elements and attributes. These restrictions are called **facets**.

Some restrictions that apply on XSD elements are as follows:

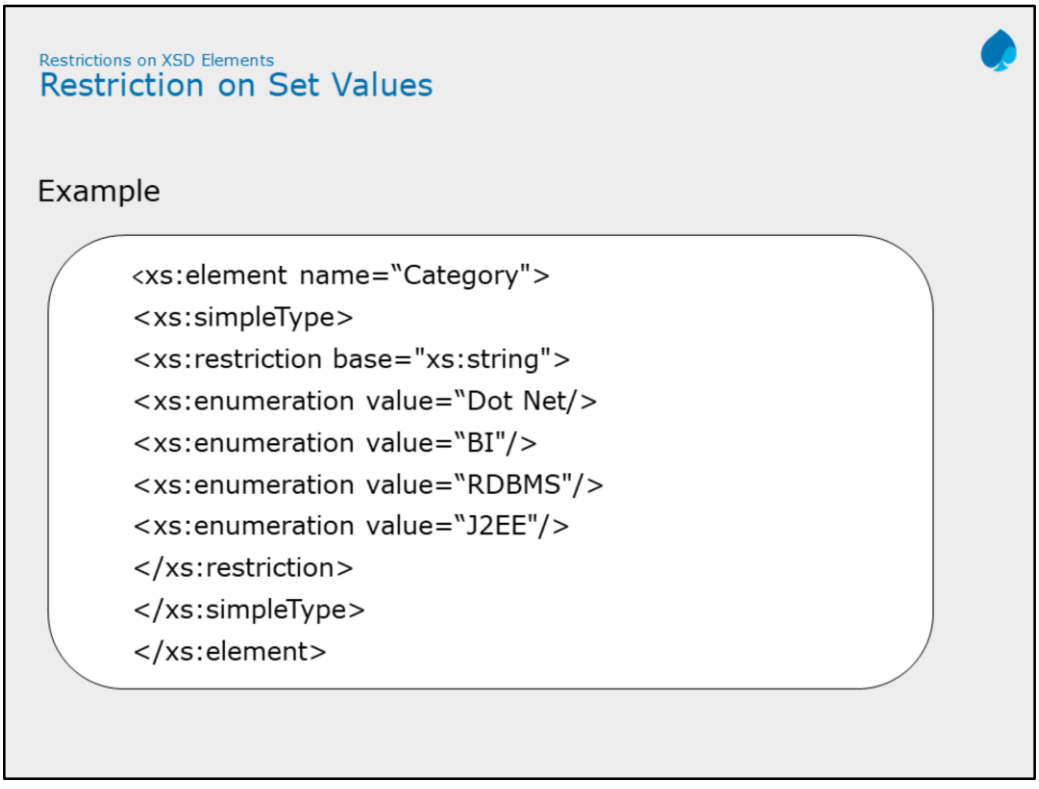
Constraint	Description
Enumeration	Defines a list of acceptable values
FractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero.
Length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero.
MaxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
MaxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
MaxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero.
MinExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
MinInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
MinLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero.
Pattern	Defines the exact sequence of characters that are acceptable
TotalDigits	Specifies the exact number of digits allowed. Must be greater than zero
WhiteSpace	Specifies the manner in which white space (line feeds, tabs, spaces, and carriage returns) is handled



Restrictions on XSD Elements:

Restrictions on Values:

The example in the above slide defines an element called “Quantity” with a restriction. The value of book “Quantity” cannot be lower than 0 or greater than 500.



Restriction on Set of Values:

According to the example shown on the above slide, the Element category can have only four possible values which are Dot Net, BI, RDBMs, and J2EE.

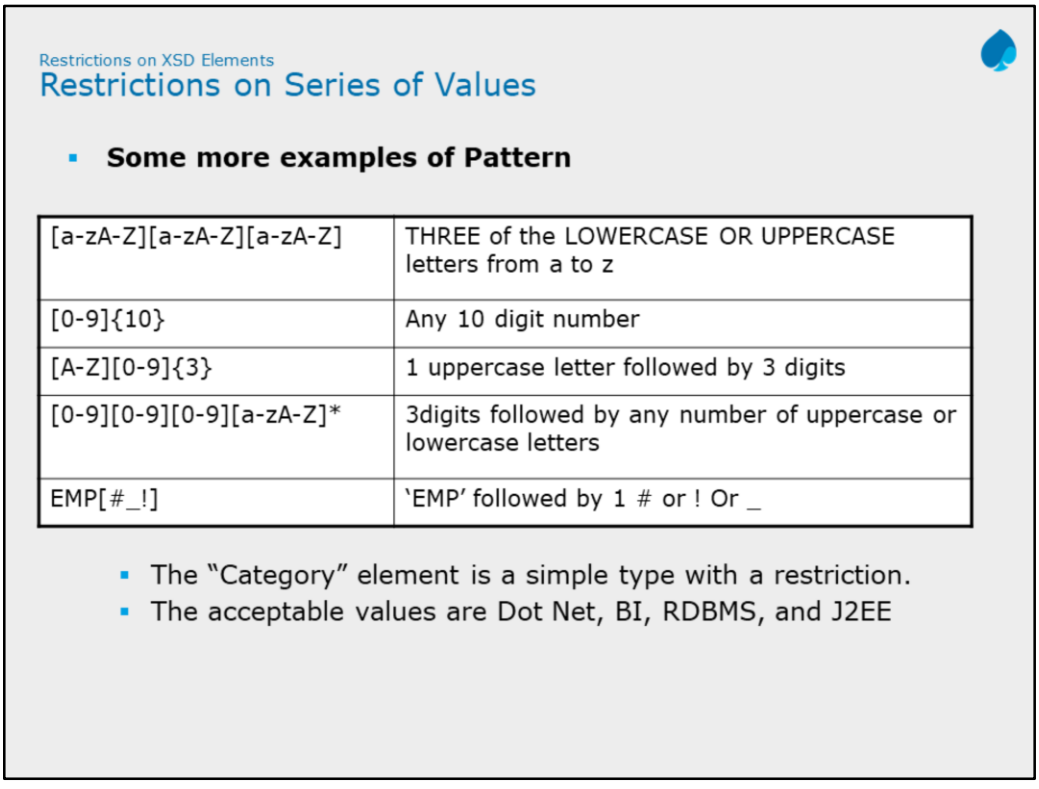
Restrictions on XSD Elements

Restrictions on Series of Values

To limit the content of an XML element to define a series of numbers or letters that can be used, we can use the pattern constraint.

```
<xs:element name="letter">
<xs:simpleType>
  <xs:restriction base="xs:string">
  <xs:pattern value="[a-z]"/>
</xs:restriction>
</xs:simpleType>
 </xs:element>
```

- The only acceptable value is ONE of the LOWERCASE letters from a to z
- The "Category" element is a simple type with a restriction.
- The acceptable values are Dot Net, BI, RDBMS, and J2EE



Restrictions on Series of Values

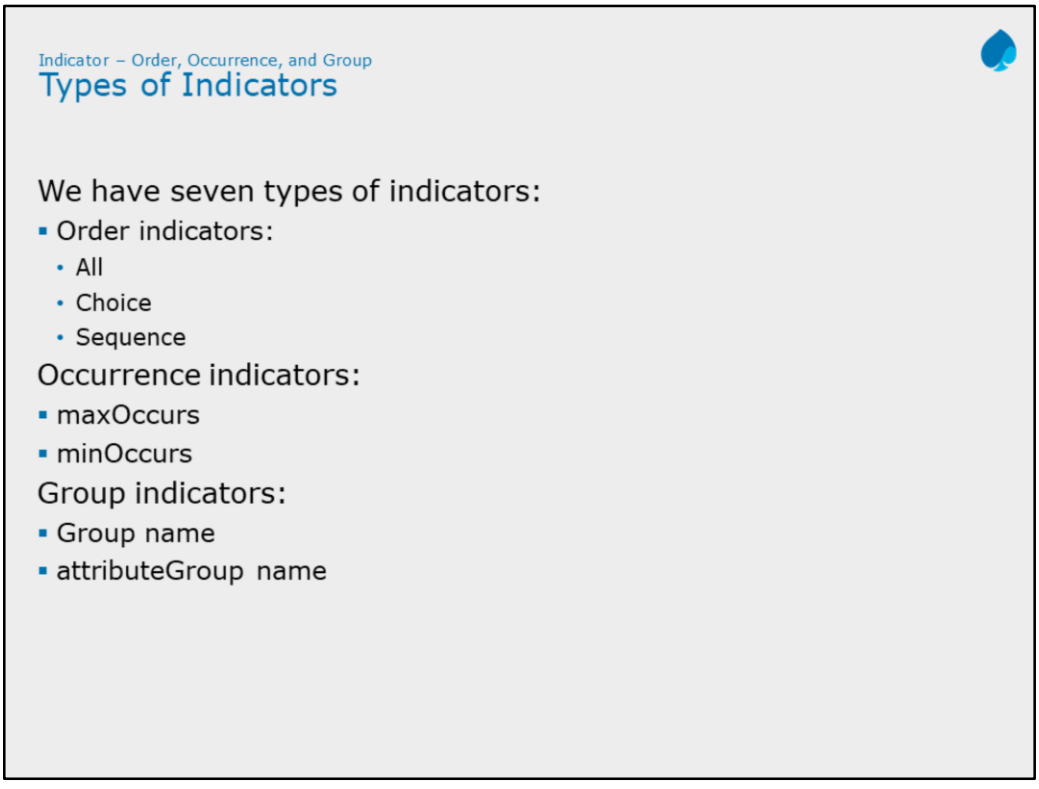
The above table shows how to give patterns in restrictions.

The next example defines an element called "gender" with a restriction. The only acceptable value is male or female:

```
<xs:element name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The next example defines an element called "password" with a restriction. There must be exactly eight characters in a row and those characters must be lowercase or uppercase letters from a to z, or a number from 0 to 9:

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```



Types of Indicators:

Indicators are specially used to control the occurrences of elements in different orders. Sometimes, we may want certain elements to occur only once, or certain elements may not be in a particular order, or certain elements may not be necessary at all (optional) and so on.

We can handle these kinds of issues by using **indicators**.

Order Indicators:

Order indicators are used to define how elements should occur.

Indicator – Order, Occurrence, and Group

All Indicator

The <all> indicator specifies, by default, that the child elements can appear in any order and that each child element must occur once and only once

```
<xs:element name="book">
 <xs:complexType>
 <xs:all>
<xs:element name="title" type="xs:string"/>
<xs:element name="author" type="xs:string"/>
</xs:all>
</xs:complexType>
</xs:element>
```

Indicator – Order, Occurrence, and Group:

In the example shown above, “book” element can have only “title & author” child elements which can occur in any sequence.

All Indicator:

When using the <all> indicator you can set the <minOccurs> indicator to 0 or 1 and the <maxOccurs> indicator can only be set to 1.

Indicator – Order, Occurrence, and Group

Choice Indicator

The <choice> indicator specifies that either one child element or another can occur

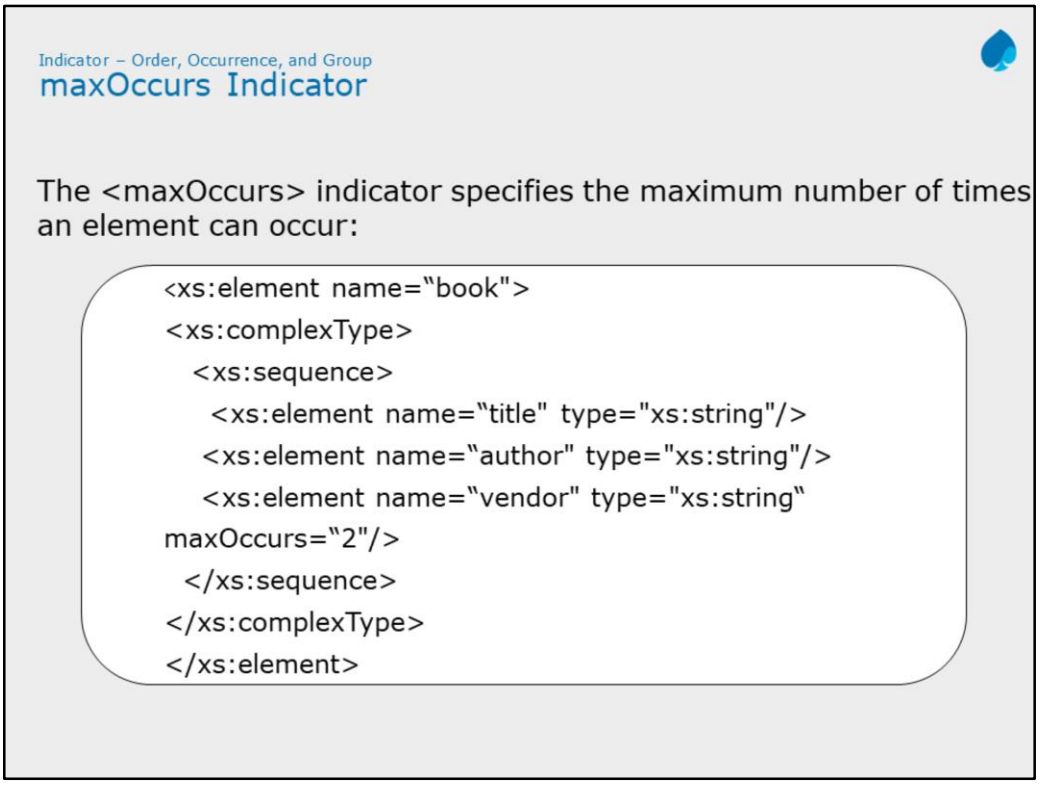
```
<xs:element name="person">
<xs:complexType>
<xs:choice>
<xs:element name="employee" type="employee"/>
<xs:element name="member" type="member"/>
</xs:choice>
</xs:complexType>
</xs:element>
```

Indicator – Order, Occurrence, and Group

Sequence Indicator

The <sequence> indicator specifies that the child elements must appear in a specific order

```
<xs:element name="book">
<xs:complexType>
  <xs:sequence>
    <xs:element name="title" type="xs:string"/>
   <xs:element name="author" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
```



Indicator – Order, Occurrence, and Group:

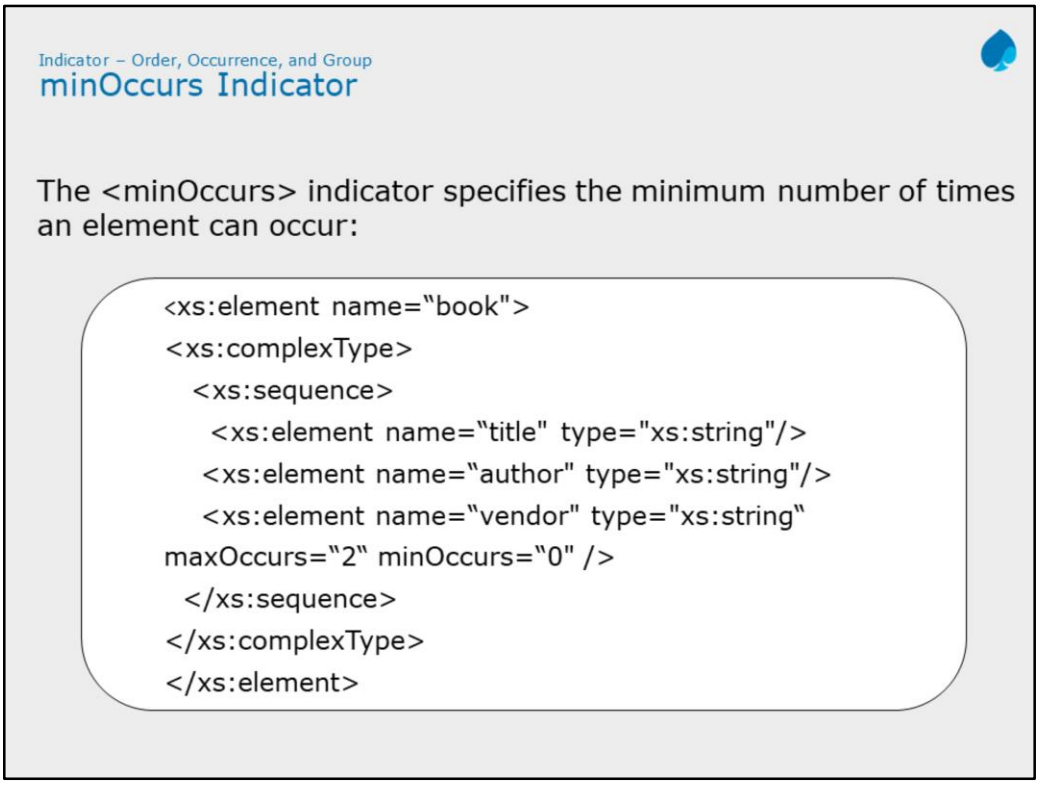
Occurrence Indicators:

Occurrence indicators are used to define how often an element can occur.

Note: For all “Order” and “Group” indicators (any, all, choice, sequence, group name, and group reference), the default value for maxOccurs and minOccurs is 1.

maxOccurs Indicator:

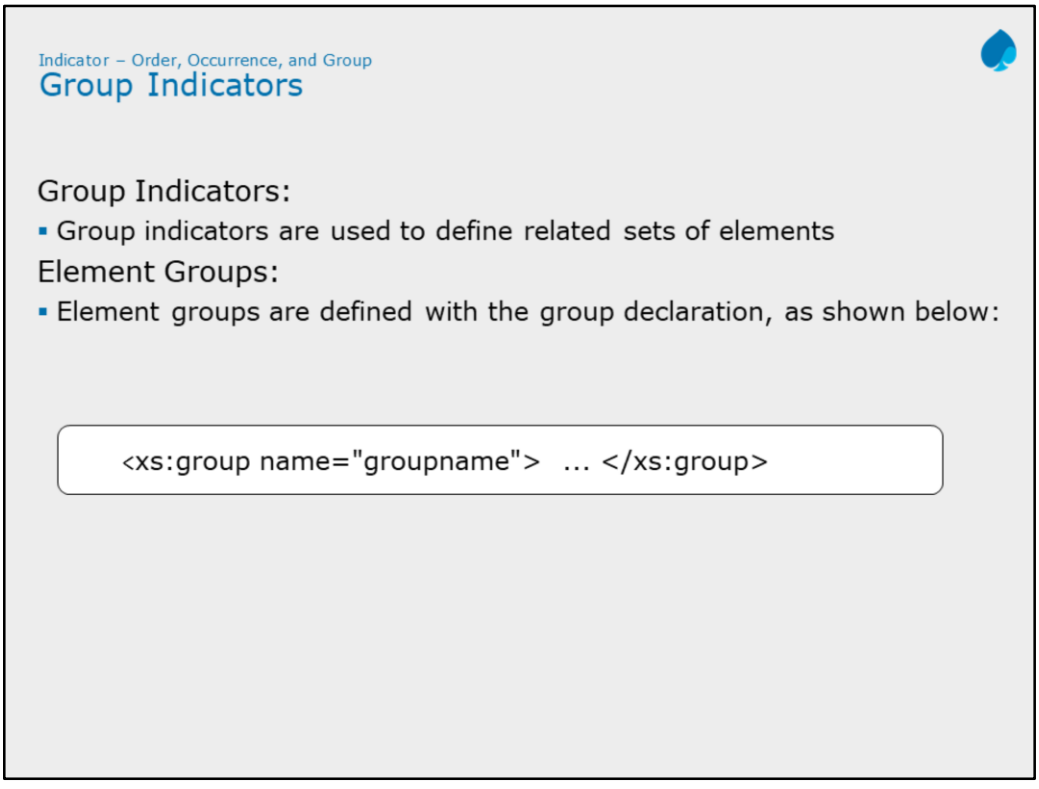
The <maxOccurs> indicator specifies the maximum number of times an element can occur.



Indicator – Order, Occurrence, and Group:
minOccurs Indicator:

The example in the above slide indicates that the “vendor” element can occur a minimum of zero times and a maximum of two times in the “book” element.

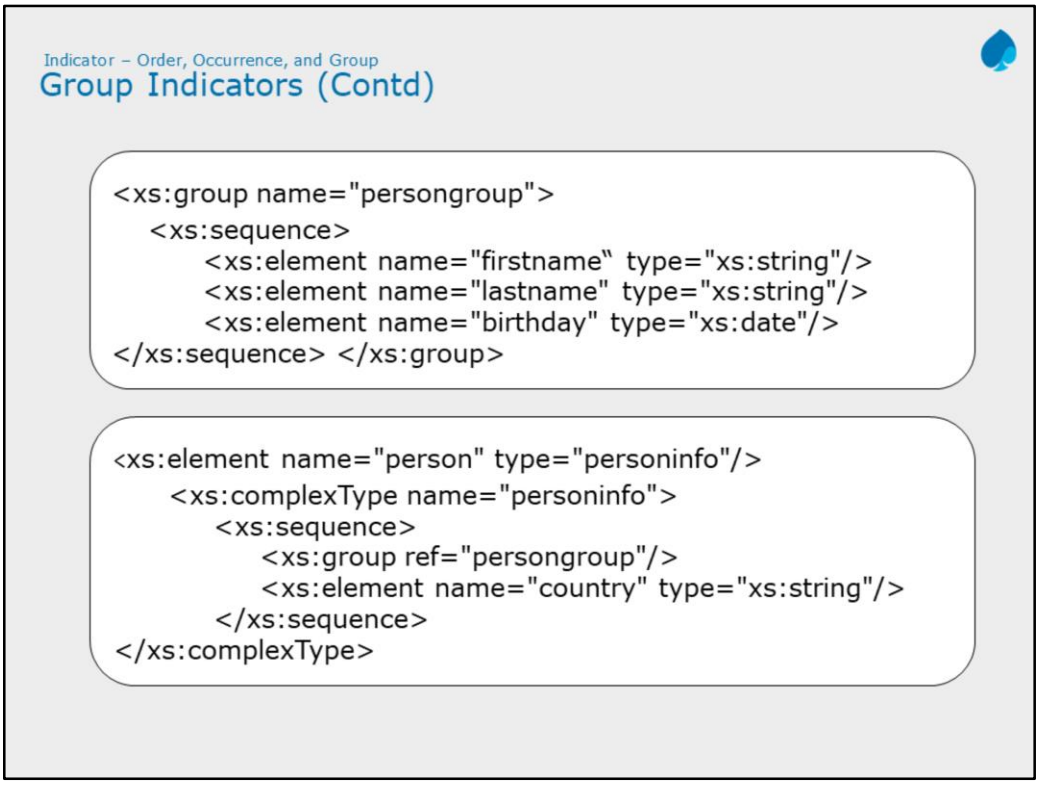
Tip: To allow an element to appear for an unlimited number of times, use the maxOccurs="unbounded" statement.



Indicator – Order, Occurrence, and Group:

Group Indicators:

You must define an all, choice, or sequence element inside the group declaration.



Group Indicators (Contd):

The example in the above slide defines a group named “persongroup”, that defines a group of elements that must occur in an exact sequence.

After you have defined a group, you can reference it in another group or complex type definition, as shown above.

Demo on XML-Schema Definition

Demo on:

- Run Validator
- Code to be validated
- Schema file
- Shiporder.xsd (schema File)
- Shiporder.xml
  (xml Document)

Summary

In this lesson, you have learnt:

- A schema describes the arrangement of markup and character data within a valid XML document
- The purpose of XML Schema and XML DTD is same
- Currently, only Microsoft IE5 supports XML Schema
- XML Schema vocabulary defines different elements

Review Question

Question 1: List any four valid datatypes in XML Schema: ___, ___, ___, and ___.

Question 2:The elements defined in a schema come from this namespace:

- Option 1 : sourceNamespace
- Option 2: targetNamespace
- Option 3: cannot be specified

Question 3: Choice is an Occurrence indicator.

- True/False