

IBM XML certification success, Part 1: Prepare for IBM XML certification with XML basics

Review DTDs, schemas, Web services, and security

Skill Level: Introductory

[Pradeep Chopra \(authors@whizlabs.com\)](mailto:authors@whizlabs.com)

Cofounder
WHIZlabs Software

[Hari Vignesh Padmanaban \(authors@whizlabs.com\)](mailto:authors@whizlabs.com)

Software engineer
Invensys Foxboro

18 Mar 2005

This is the first part of a three-part tutorial series designed specifically for those interested in taking the IBM Certified Solution Developer Exam for XML and Related Technologies. Here, authors Pradeep Chopra and Hari Vignesh Padmanaban help you prepare for the exam with explanations, examples, practice questions, and tips that cover the following topics: XML basics, Document Type Definitions (DTDs), W3C XML Schema, Web services, and security.

Section 1. Overview

About this tutorial

This tutorial is the first part of a three-part tutorial series designed specifically for those interested in taking the [IBM Certified Solution Developer exam for XML and Related Technologies](#).

The first two parts cover XML technologies that are fundamental to the exam, while

the third part covers the objectives of the actual exam in detail.

In all the three tutorials, each section of the main lesson includes:

- An example that illustrates the theory covered in the section
- An exercise for testing your skills
- An exam tip that relates to the topics covered in the section

Note: The XML topics covered in each section of the tutorial pertain specifically to the XML certification exam and are not intended to be a comprehensive overview of XML or its related technologies.

About the XML certification exam

The XML certification exam offered by IBM tests the user's knowledge of XML and various XML-related technologies.

The exam covers the following topics:

- XML
- Web services
- XPath
- XSLT
- SAX
- DOM
- XSL-FO
- DTDs
- W3C XML Schema
- XML security

It is one of only a few certifications available for XML. Unlike other certifications, the XML exam requires you to not only understand these technologies, but also be able to apply them to solve real-world situations. As a result, most of the questions in the XML exam are scenario-based questions in which you are expected to choose the most appropriate solution for a given scenario. This makes it more challenging than other exams.

In particular, preparation for this exam should emphasize the importance of

; anyone seriously considering XML certification should have broad experience with XML-related products *and* technologies.

For more information on the exam, visit the [information page for the IBM XML certification exam](#) where you'll find:

- Details on the **job role** and **target audience** for whom the certification was built
- **Recommended prerequisites** for the knowledge and skills one should possess before considering this certification
- The **test objectives** and the **skills** measured by the exam
- Recommended **educational resources** to prepare you for the test, based on the test objectives
- A **pre-assessment/sample test** to gauge your readiness for the actual exam

Note: *This tutorial was produced by an independent (non-IBM) organization, and is not necessarily endorsed by the IBM Certification Team.*

For general information about the IBM Professional Certification Program, visit <http://www-03.ibm.com/certify/>.

Section 2. An introduction to XML

Elements and attributes

Elements and their attributes form the basic building blocks of XML files.

Elements

Elements are the main constituents of an XML document; most of the data in XML documents is contained within elements. An element is represented in an XML document using a start tag (<>) and an end tag (</>). For example, an element named `Book` can be represented in an XML file as follows:

```
<Book></Book>
```

The start tag has the format `<ELEMENT_NAME>` and the end tag has the format

```
</ELEMENT_NAME>.
```

Element structure

Everything between the start and end tags of an element is considered to be the **content** of the element. The content of an element can be character data (text), no content (empty element), or other elements (known as child elements). Child elements may, in turn, contain their own child elements, and so on. Elements can also contain comments, processing instructions, or entity references.

You can represent empty elements with a single special tag, as follows:

```
<Book/> , <Name/>
```

These two definitions are equivalent to:

```
<Book></Book> and <Name></Name>
```

Attributes

An element can contain any number of attributes. Attributes give you more information about the data an element represents. Attributes are specified as name-value pairs and they appear within the start tag of the element. Attribute values are enclosed within quotes. An attribute value may contain text characters, entity references, or character references.

It is important to remember that attribute names are unique within the same element. In a single element, no two attributes can have the same name.

For example:

```
<Book isbn="s675-098-098"></Book>
```

is valid, but

```
<Book isbn="s675-098-098" isbn="b453-432-111"></Book>
```

is *not* valid.

Well-formed XML

XML code that follows certain rules is called **well-formed XML**. These rules are defined in the Worldwide Web Consortium's (W3C) XML specifications. An XML document is said to be well formed if:

- For each element defined in the XML document, every start tag has a corresponding end tag.
- All the XML elements must nest properly within each other (for example, `<book><name></book></name>` is *not* allowed, but `<book><name></name></book>` is).
- In each element, no two attributes have the same name.
- Markup characters are properly specified.
- The document contains only one **root element** -- meaning, an element that is present only once in the document and does not appear as a child of any other element.

Namespaces

In XML, namespaces are defined to avoid naming conflicts. In many applications it is possible for two elements to have the same name but contain different data. In these cases, it is necessary to have some sort of naming system that will enable XML parsers to distinguish between the different elements.

For example, consider the following XML document in which two elements are named `Book` under an element called `Books`.

```
<Books>
  <!--One Book element may contain -->
  <Book>
    <Name>Understanding Namespaces</Name>
    <Author>Whizlabs</Author>
    <ISBN> s677-898-765-098</ISBN>
    <Price>78.00</Price>
  </Book>

  <!-- while the other Book element may contain-->
  <Book>
    <Chapters>
      <Chapter1>Namespaces</Chapter1>
      <Chapter2>Schemas</Chapter2>
    </Chapters>
  </Book>
</Books>
```

The first `Book` element contains information about the book such as its name and author, while the second `Book` element contains information about the content of the book.

When you use namespaces, you can solve this conflict. With namespaces, you can define unique names within the XML document. You define a namespace by assigning a prefix to a unique Uniform Resource Identifier (URI), and then using that prefix to define specific elements. In this way, you can define elements with the

same element name using different prefixes that are associated with different URIs.

Define a prefix and associate it with a URI using a special attribute called `xmlns`:

```
<Books xmlns:BookInfo = "http://www.booksforsale.com/BookInformation"
       xmlns:BookContent="http://www.booksforsale.com/BookContent" >
```

Now, you can prefix both book elements with `BookInfo` or with `BookContent` to make them belong to different namespaces! Hence, even if both of the following `Book` elements are present in the same element, they are different as they now belong to different namespaces.

```
<BookInfo:Book>
  <BookInfo:Name>Understanding Namespaces</BookInfo:Name>
  <BookInfo:Author>Whizlabs</BookInfo:Author>
  <BookInfo:ISBN> s677-898-765-098</BookInfo:ISBN>
  <BookInfo:Price> s677-898-765-098</BookInfo:Price>
</BookInfo:Book>

<BookContent:Book>
  <BookContent:Chapters>
    <BookContent:Chapter1>Namespaces</BookContent:Chapter1>
    <BookContent:Chapter2>Schemas</BookContent:Chapter2>
  </BookContent:Chapters>
</BookContent:Book>
```

Such a prefixed name is known as a **qualified name**, or **QName**. As you can see, it is made up of two parts: the prefix, known as the **namespace prefix**, and the element name, known as the **local part**. QNames are used in most XML parsers to extract elements that belong to a particular namespace.

Default namespaces and scope

For a namespace definition, a prefix is *optional*. All elements that are defined without a prefix and appear within the element containing the namespace declaration belong to that **default namespace**.

A namespace declaration applies to the element that contains the definition as well as its child elements, unless it is overridden by another namespace declaration within the element definition.

Example

```
<Books xmlns:BookInfo="http://www.booksforsale.com/BookInformation"
       xmlns:BookContent="http://www.booksforsale.com/BookContent"
       xmlns ="http://www.booksforsale.com/BookDefault" >

  <Book>
    <BookInfo:Name>Understanding Namespaces</BookInfo:Name>
    <Author>Whizlabs</Author>
```

```

<BookInfo:ISBN>s677-898-765-098</BookInfo:ISBN>
<BookContent:Price>s677-898-765-098</BookContent:Price>
<Publisher
  xmlns="http://www.booksforsale.com/Publishers">Whizlabs</Publisher>
</Book>
</Books>

```

In the above example, the following are the element names and the namespaces they belong to:

Element	Namespace
<Book>	http://www.booksforsale.com/BookDefault
<Name>	http://www.booksforsale.com/BookInformation
<Author>	http://www.booksforsale.com/BookDefault
<ISBN>	http://www.booksforsale.com/BookInformation
<Price>	http://www.booksforsale.com/BookContent
<Publisher>	http://www.booksforsale.com/Publishers

Attributes

As with elements, you can also qualify attributes by assigning them a prefix that's mapped to a namespace declaration. But attributes behave differently from elements when it comes to the application of namespaces. If an attribute is not qualified with a prefix, it does not belong to *any* namespace, so default namespace declarations do not apply to attributes. And attributes cannot be duplicated, even with different namespace prefixes.

For example, consider the following declaration:

```
<book xmlns="http://www.BookPress.com" id="S567"/>
```

Even though the `book` element is defined to belong to the default namespace, `http://www.BookPress.com`, the `id` attribute *does not* belong to that namespace. To specify the namespace for `id` as `http://www.BookPress.com`, it should be done as follows:

```
<book xmlns:bookPress="http://www.BookPress.com" bookPress:id="S567"/>
```

Summary of XML

XML documents contain **elements** and **attributes**, which have to conform to rules established by the W3C in order to be valid XML documents. It is also possible to have a shared vocabulary through **namespaces**. Namespaces can be default and apply within the scope of the elements in which they are defined. Attributes cannot belong to any default namespace.

Example of XML

We present the examples discussed in this section in a more elegant way, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Books xmlns:BookInfo="http://www.booksforsale.com/BookInformation"
       xmlns:BookContent="http://www.booksforsale.com/BookContent"
       xmlns="http://www.booksforsale.com/BookDefault">

  <BookInfo:Book>
    <Name>Understanding Namespaces</Name>
    <Author>Whizlabs</Author>
    <ISBN> s677-898-765-098</ISBN>
    <Price> s677-898-765-098</Price>
  </BookInfo:Book>

  <BookContent:Book>
    <Chapters>
      <Chapter1>Namespaces</Chapter1>
      <Chapter2>Schemas</Chapter2>
    </Chapters>
  </BookContent:Book>

</Books>
```

Exercise for XML

Which of the following XML files is well-formed?

```
<Movies>
  <Movie name="Mask">
    <Actor>Jim Carrey</Actor>
    <Type>Comedy</Type>
  <Movie name ="Terminator"></Movie>
  <Actor>Arnold </Actor>
  <Type>Action</Type>
  <Movie name="Speed" />
</Movies>
```

```
<Movie name="Mask">
  <Actor>Jim Carrey</Actor>
  <Type>Comedy</Type>
```

```
</Movie>
<Movie name="Terminator">
<Actor>Arnold </Actor>
<Type>Action</Type>
</Movie>
<Movie name="Speed" />
```

```
<Movies>
<Movie name="Mask">
<Actor>Jim Carrey</Actor>
<Type>Comedy</Type>
</Movie>
<Movie name="Terminator">
<Actor>Arnold</Actor>
<Type>Action</Type>
</Movie>
<Movie name="Speed" />
</Movies>
```

```
<Movies>
<Movie name="Mask">
<Actor></Movie>Jim Carrey</Actor>
<Type>Comedy</Type>
<Movie name="Terminator">
<Actor>Arnold</Actor>
<Type>Action</Type>
</Movie>
<Movie name="Speed" />
</Movies>
```

Correct choice: C

Explanation:

Choice C is the correct answer. It follows all of the rules specified by the W3C for well-formedness.

Choice A is incorrect as two of the `<Movie>` tags do not have corresponding closing `</Movie>` tags.

Choice B is incorrect as the XML document does not have a root element.

Choice D is incorrect as the first `<Movie>` tag is improperly closed. Improper nesting of start and end elements is not allowed by the W3C.

Exam tip for XML

This section is designed to help you with questions in the XML exam that focus on

the concept of well-formed XML and namespaces. It is important to understand the rules required for the XML to be well formed, as well as the concept of default namespaces and scope of namespaces.

Section 3. DTDs

DTD basics

Document Type Definitions (DTDs) define the constraints of the structure of XML documents.

DTDs are not written in XML. They have their own syntax. DTDs are not based on XML; they are not extensible and do not support strong data typing.

Although they are rapidly being replaced by W3C XML Schemas (discussed in the next section), DTDs are still used in many XML applications, and XML certification exams require you to have a solid understanding of them. You can write a DTD in a separate file and link it to an XML document, or write a DTD in the same document and reference it from the XML document as explained in this section.

Note: Strong data typing indicates support for strong data types (such as `string`, `float`, `boolean`, `decimal`, and `dateTime`). A language is weakly typed if it accepts expressions that are not safe (for example, accepting a string where an integer is expected). Strong data typing ensures that expressions are safe and the data is valid.

DTD linking

The **Document Type Declaration (DOCTYPE)** links a DTD to an XML document. You can include a DTD *in* an XML document (internal subset) or the DTD can exist as a *separate* document (external subset). In both cases, you have to use the DOCTYPE declaration to define the DTD, and the syntax of that declaration depends on where it is defined.

For example, an internal DOCTYPE declaration has the following syntax:

```
<!DOCTYPE ROOT_ELEMENT_NAME [  
<!-- DTD DECLARATIONS -->  
>
```

Whereas an external DOCTYPE declaration has either of the following syntaxes:

```
<!DOCTYPE ROOT_ELEMENT_NAME SYSTEM "SYSTEM_ID">
```

or

```
<!DOCTYPE ROOT_ELEMENT_NAME PUBLIC "PUBLIC_ID"  
"SYSTEM_ID">
```

Where the value of:

- ROOT_ELEMENT_NAME is the name of the root element in the XML file
- SYSTEM_ID or PUBLIC_ID is the location of the DTD

The main difference between the system identifier (SYSTEM_ID) and the public identifier (PUBLIC_ID) is that the public identifier is *location independent*. For example, the PUBLIC_ID might be mapped to a particular location on the local area network (LAN), and that mapping might be changed in the future to point to a different location. Most DTDs use the SYSTEM_ID to specify the location of the DTD. Also, the PUBLIC_ID is followed by a backup SYSTEM_ID in an external DOCTYPE declaration.

Examples of internal and external DOCTYPE declarations are:

```
<!DOCTYPE Books SYSTEM "Books.dtd">
```

and

```
<!DOCTYPE html  
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Where the value of:

- ROOT_ELEMENT_NAME is html
- PUBLIC_ID is "-//W3C//DTD XHTML 1.0 Strict//EN
- SYSTEM_ID is
http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"

In the external declaration type, http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd is used only if "-//W3C//DTD XHTML 1.0 Strict//EN is not available.

DTD declarations: Elements

As we mentioned earlier, elements are the fundamental building blocks of any XML document. In a DTD, the element declaration (`ELEMENT`) defines an element. With the element declaration, you can specify the type of elements allowed in an XML document (content model) and the number of times an element can occur in a document (cardinality); the element declaration also provides information about what child elements an element can contain.

An element declaration has the following syntax:

```
<!ELEMENT ELEMENT_NAME CONTENT_MODEL>
```

The content model describes the type of the element. The following table shows the various content model types that can be specified for the element declaration.

Content model	Description
ANY	Any type of child is allowed within the element.
EMPTY	No children are allowed within the element.
(#PCDATA)	Only text is allowed within the element.
(Child1, Child2, Child3) (also known as a sequence list)	Child elements appear in the order specified within the element.
(Child1 Child2 Child3) (also known as a choice list)	Only one of the several mutually exclusive child elements may appear within the element.

You can also specify a **mixed content** model in which child elements and text can be mixed in any order. This type is specified using the following syntax:

```
<!ELEMENT ELEMENT_NAME (#PCDATA | CHILD_ELEMENT_1 | CHILD_ELEMENT_2) >
```

DTD declarations: Cardinalities

Occurrence modifiers or **cardinality operators** control the number of times that a particular element may occur in an XML document. Most of the exam questions related to DTDs require a strong understanding of these cardinality operators. You can use the following cardinality operators:

Operator	Description
None	Only one instance of the element is allowed.
*	Element can appear zero or more times in the

	document.
+	Element can occur one or more times in the document.
?	Element can occur zero or one times in the document.

Here's an example:

```
<!ELEMENT BookDetails (Book*)>
<!ELEMENT Book (Name, Price, ISBN, Rating?, (NonTechnical|Technical))>
```

The above DTD declaration implies that the `BookDetails` element can contain zero or more `Book` elements, and that each `Book` element:

- Must contain `Name`, `Price`, and `ISBN` elements (only one instance of each)
- Might contain an optional `Rating` element (and if present, only one instance of the element)
- Must contain either a `NonTechnical` or a `Technical` element, but not both (only one instance of the either element)

DTD declarations: Attributes (basics)

In a DTD, declare attributes separately for each element. Define attributes using the `ATTLIST` declaration, which makes it possible to restrict the type of the attribute as well as specify an **occurrence constraint**. To achieve the constraint on the attribute specification and value, specify a default or use a keyword that specifies the constraints on the attribute value.

Eight attribute types are allowed in a DTD:

- `CDATA`
- `ID`
- `IDREF`
- `IDREFS`
- `ENTITY`
- `ENTITIES`
- `NMTOKEN`

- NMTOKENS

You can have a fixed value on an attribute, make the attribute specification mandatory or optional, or supply a default value.

The syntax for defining an attribute is as follows:

```
<!ATTLIST ELEMENT_NAME ATTRIBUTE_1_NAME TYPE (DEFAULT_VALUE | KEYWORD)
        ATTRIBUTE_2_NAME TYPE (DEFAULT_VALUE | KEYWORD) >
```

DTD declarations: Attributes (types and occurrence constraints)

Before you define an attribute, it is important to know the exact data type it will represent. The various attribute types and their descriptions are:

Attribute type	Description
CDATA	Character data
ID	A name that has to be unique in the document
IDREF	A reference to an ID value in the XML document
IDREFS	A list of IDREF values delimited by space
ENTITY	The name of an unparsed entity
ENTITIES	A list of ENTITY values delimited by space
NMTOKEN	A name that is a valid XML name
NMTOKENS	A list of NMTOKEN values delimited by space

The various occurrence constraint options for an attribute are:

Keyword	Constraints
Default attribute value	Contains the default value of the attribute
#REQUIRED	Indicates that a value for the attribute is required and must be specified
#IMPLIED	Attribute is optional
#FIXED "value"	Attribute is optional, but if a value is specified it must match the given value

DTD declarations: Entities

Entities are variables that define content that is repeated in multiple XML

documents. With an entity, you construct the *logical parts* of an XML document. You can use this set of information repeatedly in creating XML documents. You classify entities in two ways -- according to where they are used, and according to the content they contain.

Based on where they are used, you can classify entities into **general entities** (those referenced in XML) and **parameter entities** (those referenced in DTD documents). Based on their content, you can classify entities into **parsed entities** (well-formed XML content) and **unparsed entities** (non-XML data). Moreover, parsed entities can be **internal** (replacement content is included in the declaration) or **external** (replacement content is located externally). Every unparsed entity *must* have an associated notation.

Basic formats for the different entity types are:

Entity	Format
Internal parameter entity	<code><!ENTITY % NAME 'value'></code>
External parameter entity	<code><!ENTITY % NAME SYSTEM 'SYSTEM_ID'></code>
Internal general entity	<code><!ENTITY NAME 'value'></code>
External general entity	<code><!ENTITY NAME SYSTEM 'SYSTEM_ID'></code>
Unparsed general entity	<code><!ENTITY NAME SYSTEM 'SYSTEM_ID' NDATA NOTATION_NAME></code>

DTD declarations: Notation

If an XML document contains unparsed entities, you need a mechanism that tells the parser how to handle that non-XML data. A **notation** declaration in a DTD gives information about handling unparsed entities. The basic syntax of a notation is:

```
<!NOTATION DATA_TYPE SYSTEM 'SYSTEM_ID'>
<!NOTATION DATA_TYPE PUBLIC 'PUBLIC_ID' 'SYSTEM_ID'>
```

An example of the basic syntax of a notation is:

```
<!NOTATION jpeg SYSTEM 'http://www.whizlabs.com/JPG_EDITOR.exe'>
<!ENTITY Image SYSTEM 'cool.jpg' NDATA jpeg>
```

Summary of DTD

We have shown you how to define a DTD, how to link the DTD with an XML file, and the various DTD declarations that control the structure of the XML file. For the exam,

it's important to remember the different content models, cardinality operators, and attribute types, and how you can apply them in modeling the structure of an XML document.

Example of DTD

The first code listing below (Class.dtd) defines a DTD for the XML document that appears below it (Class.xml). Try to come up with your own DTD for the Class.xml file and see if it matches Class.dtd.

```
<!-- DTD File - Class.dtd -->

<!ELEMENT Class (Students*)>
<!ELEMENT Students (Student)*>
<!ELEMENT Student (Name, Phone)>
<!ELEMENT Name (First, Middle?, Last)>
<!ELEMENT Phone (Home, (Cell | Mobile)?)>
<!ELEMENT First (#PCDATA)>
<!ELEMENT Middle (#PCDATA)>
<!ELEMENT Last (#PCDATA)>
<!ELEMENT Home (#PCDATA)>
<!ELEMENT Cell (#PCDATA)>
<!ELEMENT Mobile (#PCDATA)>
<!ATTLIST Student
    StudentID ID #REQUIRED
    Country CDATA #REQUIRED
    School NMTOKEN #FIXED 'MIT'>
```

```
<!-- XML File - Class.dtd-->

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Class SYSTEM "Class.dtd">
<Class>
<Students>
    <Student StudentID='s34567' Country='USA'>
        <Name>
            <First> John </First>
            <Last> Peter </Last>
        </Name>
        <Phone>
            <Home>6175557865</Home>
            <Mobile>1232345678</Mobile>
        </Phone>
    </Student>
    <Student StudentID='s3467' Country='Canada'>
        <Name>
            <First> Maxwell</First>
            <Last> Longwood</Last>
        </Name>
        <Phone>
            <Home>6575556177</Home>
            <Mobile>2345678678</Mobile>
        </Phone>
    </Student>
</Students>
</Class>
```

Exercise for DTDs

Which of the following XML documents' contents will validate successfully against the DTD given below?

```
<!ELEMENT root (root1, root2?)*>
<!ELEMENT root1 (root2*)>
<!ELEMENT root2 (root3+)>
<!ELEMENT root3 (root1?)>
```

Assume the following declaration in the XML documents:

```
<!DOCTYPE root SYSTEM "F:\WHIZLABS\ARTICLE\ARTICLE-1\root.dtd">
```

```
<root></root>
```

```
<root>
<root1>
  <root2/>
  <root2/>
</root1>
<root2/>
</root>
```

```
<root>
  <root1>
    <root2>
      <root3/>
    </root2>
    <root2>
      <root3/>
    </root2>
  </root1>
  <root2>
    <root3/>
  </root2>
</root>
```

```
<root>
  <root1>
    <root2>
      <root3/>
    </root2>
    <root2>
      <root3/>
    </root2>
  </root1>
  <root2>
    <root3/>
  </root2>
</root>
```

```
                </root2>
            </root1>
        <root3/>
    </root>
```

Correct choices: A and C

Explanation:

Choice A is correct since the * operator specifies that it is possible for root to have 0 or more instances of root1 and root2.

Choice C is correct since there are no elements that violate the DTD model.

Choice B is incorrect since the root2 element should contain one or more instances of root3, but is empty in the XML document.

Choice D is incorrect since the root element cannot have root3 as its immediate child element. root3 can be present only under root2. And root2 can then be present under root.

Exam tip for DTDs

Since most of the XML world is shifting toward W3C XML Schema, it is enough if you know just the basics of DTDs. You should be able to define a DTD based on the XML document requirements, and choose one or more XML files that conform to a given DTD (similar to the exercise in [Exercise for DTDs](#)). It is also important to know the advantages and disadvantages of DTDs as compared to XML Schema in order to answer certain architecture questions. Remember that DTDs support only weak data typing, are not extensible, have no DOM support, and have limited support for namespaces.

Section 4. W3C XML Schema

Introduction

W3C XML Schema (Schema) is an XML-based technology that is considered a replacement for DTDs. Just like DTDs, schemas are used for defining the constraints of an XML document. But unlike DTDs, they provide strong data typing and support for namespaces -- and since they are based on XML, they are also

extensible.

Schema definition

A schema is defined in a separate file and generally stored with the .xsd extension. Every schema definition has a `schema` root element that belongs to the `http://www.w3.org/2001/XMLSchema` namespace. The `schema` element can also contain optional attributes.

For example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
```

This indicates that the elements used in the schema come from the `http://www.w3.org/2001/XMLSchema` namespace.

Schema linking

An XML file links to its corresponding schema using the `schemaLocation` attribute of the schema namespace. You have to define the schema namespace in order to use the `schemaLocation` attribute. All of these definitions appear in the root element of the XML document.

The syntax is:

```
<ROOT_ELEMENT
  SCHEMA_NAMESPACE_DEFINITION
  SCHEMA_LOCATION_DEFINITION >
```

And here's an example of it in use:

```
<Books
  xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
  xs:schemaLocation="http://www.booksforsale.com Books.xsd">
```

Schema elements

A schema file contains definitions for element and attributes, as well as data types for elements and attributes. It is also used to define the structure or the content model of an XML document. Elements in a schema file can be classified as either

simple or complex -- defined in [Schema elements: Simple type](#) and [Schema elements: Complex types](#).

Schema elements: Simple type

A **simple type** element is an element that cannot contain any attributes or child elements; it can only contain the data type specified in its declaration. The syntax for defining a simple element is:

```
<xs:element name="ELEMENT_NAME" type="DATA_TYPE" default/fixed="VALUE" />
```

Where `DATA_TYPE` is one of the built-in schema data types (see below).

You can also specify default or fixed values for an element. You do this with either the `default` or `fixed` attribute and specify a value for the attribute. **Note:** Specifying a fixed or default attribute is optional.

An example of a simple type element is:

```
<xs:element name="Author" type="xs:string" default="Whizlabs"/>
```

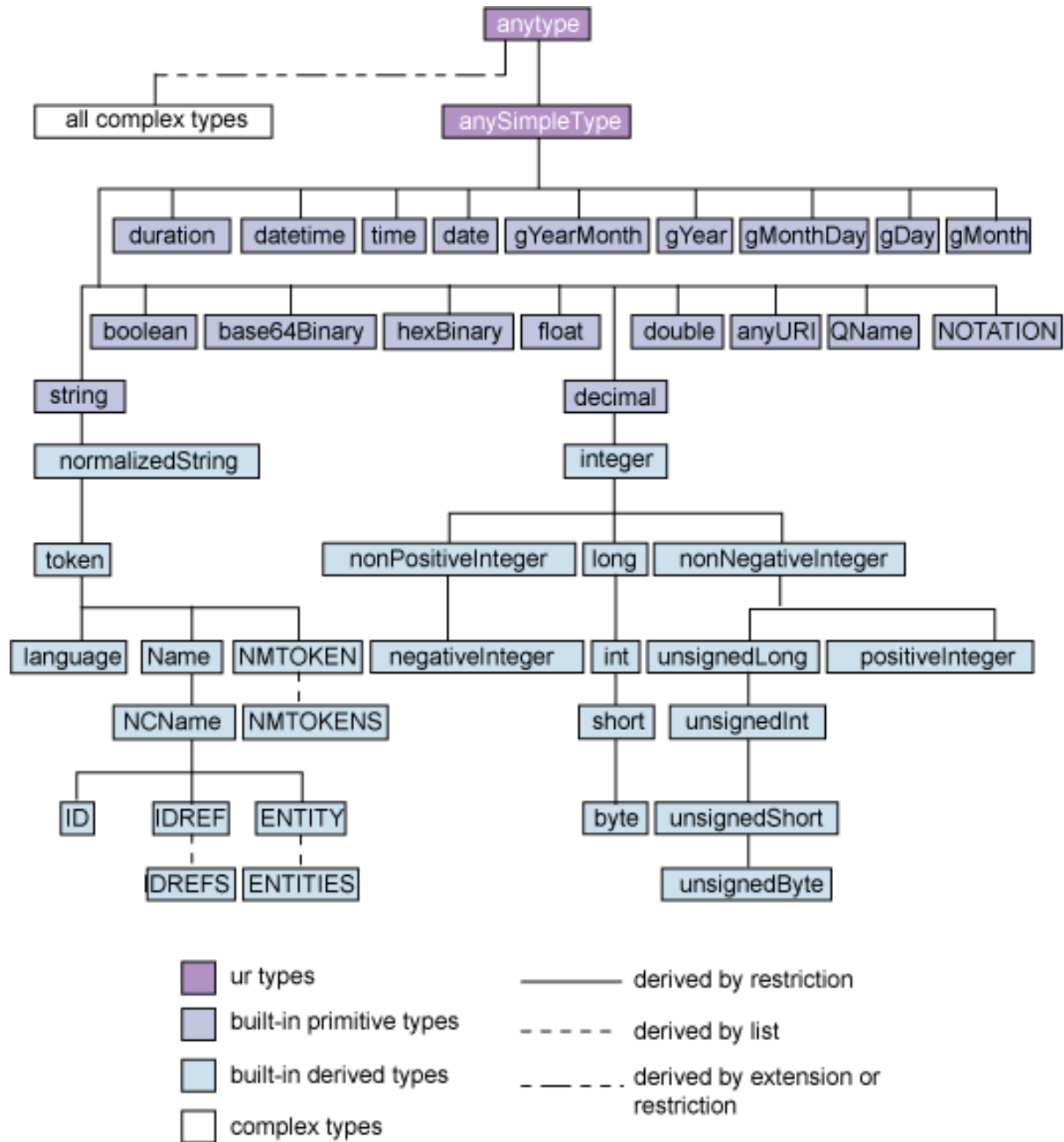
All attributes are simple types, so they are defined in the same way that simple elements are defined. For example:

```
<xs:attribute name="title" type="xs:string" />
```

Schema data types

All data types in schema inherit from `anyType`. This includes both **simple** and **complex data types**. You can further classify simple types into **built-in-primitive** types and **built-in-derived** types. A complete hierarchical diagram from the XML Schema Datatypes Recommendation (see [Resources](#)) is shown below.

Built-in datatype hierarchy



Copyright 2003 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved.
<http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231>

Schema elements: Complex types

Complex types are elements that either:

- Contain other elements
- Contain attributes
- Are empty (empty elements)
- Contain text

To define a complex type in a schema, use a `complexType` element. You can specify the order of occurrence and the number of times an element can occur (cardinality) by using the **order** and **occurrence** indicators, respectively. (See [Occurrence and order indicators](#) for more on these indicators.)

For example:

```
<xs:element name="Book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Name" type="xs:string" />
      <xs:element name="Author" type="xs:string" maxOccurs="4"/>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="Price" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

In this example, the order indicator is `xs:sequence`, and the occurrence indicator is `maxOccurs` in the `Author` element name.

Occurrence and order indicators

Occurrence indicators specify the number of times an element can occur in an XML document. You specify them with the `minOccurs` and `maxOccurs` attributes of the element in the element definition.

As the names suggest, `minOccurs` specifies the minimum number of times an element can occur in an XML document while `maxOccurs` specifies the maximum number of times the element can occur. It is possible to specify that an element might occur any number of times in an XML document. This is determined by setting the `maxOccurs` value to `unbounded`. The default values for both `minOccurs` and `maxOccurs` is 1, which means that by default an element or attribute can appear exactly one time.

Order indicators define the order or sequence in which elements can occur in an XML document. Three types of order indicators are:

- **All:** If All is the order indicator, then the defined elements can appear in any order and must occur only once. Remember that both the `maxOccurs` and `minOccurs` values for All are *always* 1.
- **Sequence:** If Sequence is the order indicator, then the elements must appear in the order specified.
- **Choice:** If Choice is the order indicator, then any one of the elements specified must appear in the XML document.

Example:

```
<xs:element name="Book">
  <xs:complexType>
    <xs:all>
      <xs:element name="Name" type="xs:string" />
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="Authors" type="authorType"/>
      <xs:element name="Price" type="priceType"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:complexType name="authorType">
  <xs:sequence>
    <xs:element name="Author" type="xs:string" maxOccurs="4"/>
  </xs:sequence>
</xs:complexType >

<xs:complexType name="priceType">
  <xs:choice>
    <xs:element name="dollars" type="xs:double" />
    <xs:element name="pounds" type="xs:double" />
  </xs:choice>
</xs:complexType >
```

In the above example, the `<xs:all>` indicator specifies that the `Book` element, if present, must contain *only one* instance of each of the following four elements: `Name`, `ID`, `Authors`, `Price`.

The `xs:sequence` indicator in the `authorType` declaration specifies that elements of this particular type (`Authors` element) contain at least one `Author` element and can contain up to four `Author` elements.

The `xs:choice` indicator in the `priceType` declaration specifies that elements of this particular type (`Price` element) can contain either a `dollars` element or a `pounds` element, but not both.

Restriction

A main advantage of schema is that you have the ability to control the value of XML attributes and elements. A **restriction**, which applies to all of the simple data

elements in a schema, allows you to define your own data type according to the requirements by modifying the facets available for a particular simple type. To achieve this, use the `restriction` element defined in the schema namespace.

W3C XML Schema defines 12 facets for simple data types. The following list includes each facet, along with its effect on the data type value and an example.

- `enumeration` - Value of the data type is constrained to a specific set of values.

```
<xs:simpleType name="Subjects">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Biology"/>
    <xs:enumeration value="History"/>
    <xs:enumeration value="Geology"/>
  </xs:restriction>
</xs:simpleType>
```

- `maxExclusive` - Numeric value of the data type is less than the value specified.

`minExclusive` - Numeric value of the data type is greater than the value specified.

```
<xs:simpleType name="id">
  <xs:restriction base="xs:integer">
    <xs:maxExclusive value="101"/>
    <xs:minExclusive value="1"/>
  </xs:restriction>
</xs:simpleType>
```

- `maxInclusive` - Numeric value of the data type is less than or equal to the value specified.
- `minInclusive` - Numeric value of the data type is greater than or equal to the value specified.

```
<xs:simpleType name="id">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="100"/>
  </xs:restriction>
</xs:simpleType>
```

- `maxLength` - Specifies the maximum number of characters or list items

allowed in the value.

`minLength` - Specifies the minimum number of characters or list items allowed in the value.

`pattern` - Value of the data type is constrained to a specific sequence of characters that are expressed using regular expressions.

```
<xs:simpleType name="nameFormat">
  <xs:restriction base="xs:string">
    <xs:minLength value="3"/>
    <xs:maxLength value="10"/>
    <xs:pattern value="[a-z][A-Z]*"/>
  </xs:restriction>
</xs:simpleType>
```

- `length` - Specifies the exact number of characters or list items allowed in the value.

```
<xs:simpleType name="secretCode">
  <xs:restriction base="xs:string">
    <xs:length value="5"/>
  </xs:restriction>
</xs:simpleType>
```

- `whiteSpace` - Specifies the method for handling white space. Allowed values for the value attribute are `preserve`, `replace`, and `collapse`.

```
<xs:simpleType name="FirstName">
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="preserve"/>
  </xs:restriction>
</xs:simpleType>
```

- `fractionDigits` - Constrains the maximum number of decimal places allowed in the value.

`totalDigits` - The number of digits allowed in the value.

```
<xs:simpleType name="reducedPrice">
  <xs:restriction base="xs:float">
    <xs:totalDigits value="4"/>
    <xs:fractionDigits value="2"/>
  </xs:restriction>
</xs:simpleType>
```

Extension

The extension element defines complex types that might derive from other complex or simple types. If the base type is a simple type, then the complex type can only add attributes. If the base type is a complex type, then it is possible to add attributes and elements. To derive from a complex type, you have to use the `complexContent` element in conjunction with the `base` attribute of the extension element.

Extensions are particularly useful when you need to reuse complex element definitions in other complex element definitions. For example, it is possible to define a `Name` element that contains two child elements (`First` and `Last`) and then reuse it in other complex element definitions.

An example of extensions is:

```
<!--Base element definition -->
<xs:complexType name="Name">
  <xs:sequence>
    <xs:element name="First"/>
    <xs:element name="Last"/>
  </xs:sequence>
</xs:complexType>

<!-- Customer element that reuses it -->
<xs:complexType name="Customer">
  <xs:complexContent>
    <xs:extension base="Name">
      <xs:sequence>
        <xs:element name="phone" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Student element that reuses it -->
<xs:complexType name="Student">
  <xs:complexContent>
    <xs:extension base="Name">
      <xs:sequence>
        <xs:element name="school" type="xs:string"/>
        <xs:element name="year" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Import and include

The `import` and `include` elements help to construct a schema from multiple documents and namespaces.

The `import` element brings in a schema from a different namespace, while the `include` element brings in a schema from the same namespace. When you use `include`, the target namespace of the included schema must be the *same* as the target namespace of the including schema. In the case of `import`, the target namespace of the included schema must be *different* from the target namespace of the including schema.

The syntax for `import` is:

```
<xs:import id="ID_DATATYPE" namespace="anyURI_DATATYPE"
schemaLocation="anyURI_DATATYPE" />
```

The syntax for `include` is:

```
<xs:include id="ID_DATATYPE" schemaLocation="anyURI_DATATYPE" />
```

Summary of schema

W3C XML Schema has become the de facto standard for defining the structure of an XML document and for checking the validity of XML documents. Using schema, it is possible to define:

- Elements (simple and complex)
- Attributes
- Facets for XML elements
- The structure of a document (order indicators)
- The allowable number of elements (occurrence indicators) in an XML document

Example of schema

The following schema defines an XML document that contains a `Books` root element containing an unlimited number of `Book` elements; each `Book` element contains `Name`, `Author`, `ID`, and `Price` elements.

Note: Each `Book` can contain a maximum of *four* `Author` elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
```

```

<xs:element name="Books">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Book">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Name" type="xs:string"/>
            <xs:element name="Author" type="xs:string" maxOccurs="4"/>
            <xs:element name="ID" type="xs:string"/>
            <xs:element name="Price" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

```

It is also possible to define a complex type separately and then refer to it using the `type` attribute of the element definition. So in the above example, the `Book` element can be defined separately as a complex type (named `BookInfo`), and this type can then be referred to by any element that uses the `type` attribute. So the modified version of the above example, along with a facet for the name type, would be:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="Books">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Book" type="BookInfo" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="BookInfo">
    <xs:sequence>
      <xs:element name="Name" type="nameFormat"/>
      <xs:element name="Author" type="xs:string" maxOccurs="4"/>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="Price" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="nameFormat">
    <xs:restriction base="xs:string">
      <xs:minLength value="3"/>
      <xs:maxLength value="10"/>
      <xs:pattern value="[a-z][A-Z]*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>

```

Exercises for schema

1. In W3C XML Schema language, which of the following represents an

integer that is either zero or negative?

- A. `xsd:negativeInteger`
- B. `non xsd:positiveInteger`
- C. `xsd:nonNegativeInteger`
- D. None of these

Correct choice: D

Explanation:

Choice D is the correct answer.

Choice A is incorrect because `xsd:negativeInteger` represents an integer strictly less than zero.

Choice B is incorrect because there is no data type called `non xsd:positiveInteger`.

Choice C is incorrect because `xsd:nonNegativeInteger` represents an integer greater than or equal to zero. (`xsd:nonPositiveInteger` represents an integer less than or equal to zero.)

2. Which of the following XML documents are *not* valid according to the schema given below?

```
<xsd:element name="Movie" type="MovieType"/>
<xsd:complexType name="MovieType">
  <xsd:sequence>
    <xsd:element name="Name" type="xsd:string"/>
    <xsd:choice maxOccurs="2">
      <xsd:element name="Actor" type="xsd:string"/>
      <xsd:element name="Actress" type="xsd:string"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
```

```
<Movie>
  <Name>
    The Mask
  </Name>
  <Actor>
    Jim Carrey
  </Actor>
  <Actress>
    Cameron Diaz
  </Actress>
```

```
</Movie>
```

```
<Movie>
  <Name>
    The Mask
  </Name>
</Movie>
```

```
<Movie>
  <Name>
    The Mask
  </Name>
  <Actor>
    Jim Carrey
  </Actor>
  <Actor>
    Jim Carrey
  </Actor>
</Movie>
```

```
<Movie>
<Name>
  The Mask
</Name>
<Actress>
  Cameron Diaz
</Actress>
<Actor>
  Jim Carrey
</Actor>
</Movie>
```

- E. None of these
Correct choice: B

Explanation:

Choice B is invalid according to the provided schema, which requires one `Name` element and one or two elements of the type `Actor` or `Actress`. This can be two `Actor` elements, two `Actress` elements, or one `Actor` element and one `Actress` element, in any order. Hence choices A, C, and D are perfectly valid according to the schema. Choice B is invalid because it doesn't contain any `Actor` or `Actress` elements, which are necessary as the default value of `minOccurs` is 1.

Exam tip for schema

Most of the questions in the various sections of the XML exam revolve around the use and application of W3C XML Schema, so it is important to know both the syntax and application of this technology. It is also important to know the advantages of this technology over DTDs. In addition, be aware of the various built-in schema data types and be able to construct a schema based on XML document requirements.

Section 5. Web services

Introduction

A **Web service** is an application on the Web that provides some sort of service to any other application. This might range from finding out the price of a book to a complex financial transaction. Web services are also used by various companies to integrate disparate applications that run on different platforms and use different technologies. Web services are based on HTTP protocol and use XML for defining, building, and transporting messages.

Web services are composed of the following core technologies:

- Simple Object Access Protocol ([SOAP](#))
- Web Services Description Language ([WSDL](#))
- Universal Description, Discovery, and Integration ([UDDI](#))

SOAP

Simple Object Access Protocol (SOAP) is the protocol that transports messages in Web services. In a Web service, SOAP is often used to initiate a conversation with a UDDI service, which provides a SOAP-based API. The requests to the Web service are also constructed and sent through SOAP.

SOAP syntax

Since a SOAP message is essentially an XML file, it can contain the following elements:

- Envelope
- Header (optional)
- Body
- Fault (present only for error messages)

All four of these elements must belong to the SOAP envelope namespace (<http://schemas.xmlsoap.org/soap/envelope/>) and use the SOAP encoding namespace (<http://schemas.xmlsoap.org/soap/encoding/>).

Envelope

The `Envelope` element is the root element of any SOAP message. Generally, it contains the definition for the required envelope namespace and the `encodingStyle` attribute, which defines the data types used in the document. For example:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
```

Header

The `Header` element is optional, and if present it must be the first child of the `Envelope` element.

A SOAP message is essentially constructed by a `sender` and sent to a `receiver`. Before the SOAP message reaches the `receiver`, it might pass through one or more nodes (SOAP intermediaries). The `Header` element can contain additional information regarding security, transaction, or other details. Depending upon this information, a SOAP intermediary might or might not be allowed to process the SOAP message.

A SOAP `Header` element can contain two attributes, `actor` and `mustUnderstand`:

Attribute	Value	Description
<code>actor</code>	URI	Indicates the location of the receiver that should process the SOAP message.
<code>mustUnderstand</code>	1 0	Indicates that the receiver should understand the <code>Header</code> element. If it is not able to understand the information contained in the <code>Header</code> , then the receiver returns an error and does not process the message. Indicates that the receiver need not understand the <code>Header</code>

element.

Body

This element contains the message that the receiver will process. The SOAP `body` might be a Remote Procedure Call (RPC), an error message (the SOAP `fault` element), or other namespace-qualified elements. When the SOAP message carries a fault message, the `body` element contains only the `fault` element.

Fault

This is the only element from the default envelope namespace that can be present inside the `body` element. When `fault` is present, no other elements can be present in the `body` element. `fault` can have four child elements:

Element name	Description
<code>faultcode</code> (mandatory)	<p>The fault code indicates the place where the fault has occurred and can be any one of these four values:</p> <ul style="list-style-type: none"> <code>VersionMismatch</code> -- This error is generated if the namespace of the <code>envelope</code> element does not belong to http://schemas.xmlsoap.org/soap/envelope/. <code>MustUnderstand</code> -- This error is generated when the receiver is not able to understand a <code>Header</code> element that has to be understood. <code>Client</code> -- This error indicates that something is wrong with the SOAP message sent by the client. <code>Server</code> -- This indicates that the server has a problem. This might be a temporary shutdown of a particular service. In this case, the SOAP message might be sent again.
<code>faultstring</code> (mandatory)	This element contains a description of the fault that has occurred.
<code>faultfactor</code> (optional)	This element contains information about the point that caused the fault.
<code>detail</code> (optional)	This element is present if the fault is a result of the receiver being unable to process the <code>body</code> element. It gives additional information that can help in resolving the problem.

WSDL

Web Services Description Language (WSDL) is used to describe the Web services offered by a company. You can think of it as an interface to the real services. A SOAP request is constructed based on the information obtained through WSDL, which is written in XML and essentially contains the following information:

- All available functions (public functions)
- Data type information for the responses and requests
- Binding Information regarding the transport protocol
- Information regarding the address of the service

The following elements, which make up the structure of a WSDL document, make this information available:

Element name	Description
types	This defines various data types that are used by the Web service.
message	This can contain 0 or more message parts. The message parts might refer to message parameters or responses.
portType	This defines the functions (operations) to be supported. The operations can be any of the following four types: one-way, request-response, solicit-response, notification.
binding	This element specifies the transport protocol details and the data formats for the operations and messages.
service	This defines the location (address) of the Web service.

Based on the definitions above, the skeletal structure of a WSDL document is:

```
<definitions>
  <types>
    <!-- data types definitions -->
  </types>
  <message>
    <!-- messages definition-->
  </message>
  <portType>
    <!-- supported operations definition -->
  </portType>
```

```
<binding>
  <!-- protocol and data formats definition -->
</binding>

<service>
  <!-- service location definition -->
</service>

</definitions>
```

UDDI

Universal Description, Discovery, and Integration (UDDI) is a protocol used to find Web services. Written in XML, UDDI acts as a repository for the numerous Web services that are available for various companies.

The UDDI **publish** and **inquiry** APIs allow you to interact with a UDDI registry. The information in UDDI is stored in the form of specific data structures. UDDI supports the following five core data structures:

- Business entity
- Business service
- Binding template
- tModel
- Publisher assertion

To use a Web service, a client first finds the appropriate service from the UDDI. It then gets the WSDL of that particular Web service, and constructs a SOAP message based on the information provided in the WSDL.

Summary of Web services

Web services is a technology that uses three XML-based technologies:

- Simple Object Access Protocol (SOAP)
- Web Services Description Language (WSDL)
- Universal Description, Discovery, and Integration (UDDI)

SOAP is now the standard protocol for sending XML messages. WSDL is the language that describes the Web service being provided. UDDI allows users to find and use various Web services that are available.

Example of Web services

Here's an example of a SOAP message:

```
<soap:envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">

  <soap:header>
    <id:transaction xmlns:id="http://www.whizlabs.com/transaction/"
      soap:mustUnderstand="1">
      67
    </id:transaction>
  </soap:header>

  <soap:Body xmlns:info="http://www.whizlabs.com/priceInfo">
    <info:GetSimulatorPrice>
      <info:SimulatorName>IBM</info:SimulatorName>
    </info:GetSimulatorPrice>
  </soap:Body>

</soap:envelope>
```

And here's a SOAP fault example:

```
<soap:envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">

  <soap:body>

    <soap:fault>
      <faultcode>soap:Client</faultcode>
      <faultstring>Invalid Parameter</faultstring>
      <faultfactor>http://www.whizlabs.com/simulators/price.php</faultfactor>

      <detail>
        <e:errorInfo xmlns:e="http://www.whizlabs.com/errors">
          <errorCode>56</errorCode>
          <description>Invalid parameter for simulator</description>
        </e:errorInfo>
      </detail>

    </soap:fault>

  </soap:body>

</soap:envelope>
```

Exercises for Web services

1. Which of the following is an XML-based service IDL that defines the Web services interface and its implementation characteristics?

A. SOAP

- B. WSDL
- C. UDDI
- D. XPath
- E. tModel

Correct choice: B

Explanation:

WSDL (Web Service Description Language) is the standard for describing a Web service. It is an XML-based service IDL that defines the Web service interface and its implementation characteristics.

SOAP (Simple Object Access Protocol) is a protocol for initiating conversations with a UDDI service. SOAP allows applications to invoke object methods, or functions, residing on remote servers.

UDDI (Universal Description, Discovery, and Integration) is used for finding Web services.

XPath is used for finding elements in an XML document.

tModel is a data structure that is supported by UDDI.

2. Which of the following is a protocol for initiating conversations with a UDDI service?

- A. WSDL
- B. UDDI
- C. SOAP
- D. None of these

Correct choice: C

Explanation:

SOAP is a protocol for initiating conversations with a UDDI service. SOAP allows applications to invoke object methods, or functions, residing on remote servers.

WSDL is the standard for describing a Web service. It is an XML-based

service IDL that defines the Web service interface and its implementation characteristics.

UDDI is used for finding Web services.

Exam tip for Web services

The XML exam tests you on the bare basics of Web services. You only need to have a basic understanding of how XML, SOAP, WSDL, and UDDI technologies fit together. Most of the questions in the exam are fairly simple and similar to the exercise questions in [Exercises for Web services](#).

Section 6. Security

XML security basics

XML security deals with the encryption of data in XML documents. You can apply this encryption to a single element, a group of elements, or the full XML document.

In many cases, you might need to encrypt only a portion of an XML document, as the same document can be used by different applications. You must allow some of these applications to process the data while you prevent others from reading the critical data.

The encrypted data is contained within an `EncryptedData` element.

EncryptedData

This element derives from the `EncryptedType` element, and is the core element in XML encryption. Its `CipherData` child element contains the encrypted data. It's important to know the structure of the `EncryptedData` element in order to encrypt information in an XML document. This structure is:

```
<EncryptedData Id? Type? MimeType? Encoding?>
  <EncryptionMethod/?>
  <ds:KeyInfo>
    <EncryptedKey?>
    <AgreementMethod?>
    <ds:KeyName?>
    <ds:RetrievalMethod?>
    <ds:*?>
  </ds:KeyInfo?>
```

```

<CipherData>
  <CipherValue>?
  <CipherReference URI??>?
</CipherData>
<EncryptionProperties>?
</EncryptedData>

```

Where:

- A question mark ? denotes zero or one occurrence
- An asterisk * denotes zero or more occurrences

The namespace `ds` refers to `http://www.w3.org/2000/09/xmldsig#`, which is the XML Signature namespace (see [Resources](#)) containing the schema definitions.

EncryptedData explained

Each element that constitutes the `EncryptedData` element is explained as follows:

Element name	Description
<code>EncryptionMethod</code>	<p>This element describes the encryption algorithm that is applied to the cipher data. This is specified in the element's <code>Algorithm</code> attribute. When this is not specified, the receiver should know the encryption algorithm. You can specify these nine categories of algorithms:</p> <ul style="list-style-type: none"> • Block encryption • Stream encryption • Key transport • Key agreement • Symmetric key wrap • Message digest • Message authentication • Canonicalization • Encoding
<code>ds:KeyInfo</code>	<p>This element contains information about the encryption key and has these child elements:</p> <ul style="list-style-type: none"> • <code>EncryptedKey</code> -- This element

	<p>transports the encryption keys from the sender to the receiver.</p> <ul style="list-style-type: none"> • <code>AgreementMethod</code> -- This element identifies the keys and computational procedures that are used to obtain a shared encryption key. • <code>ds:KeyName</code> -- This is a property of the key and is necessary to decrypt the <code><CipherData></code>. • <code>ds:RetrievalMethod</code> -- If several signatures in the same document use a key that's verified by the same certificate, then each signature's <code>KeyInfo</code> can use a single <code>ds:RetrievalMethod</code> to get to the certificate.
<p><code>CipherData</code></p>	<p>This element contains the encrypted data. It has two child elements, namely <code>CipherValue</code> and <code>CipherReference</code>. <code>CipherData</code> can contain the encrypted data in the <code>CipherValue</code> child element, or provide a reference to the encrypted data through the <code>CipherReference</code> child element.</p>
<p><code>EncryptionProperties</code></p>	<p>Use this element to specify additional information with regard to the <code>EncryptedData</code> or <code>EncryptedKey</code> elements. This can include date, time, or any other information.</p>

Summary of security

To encrypt XML documents like any other document, use the `EncryptedData` element. The `CipherData` element contains the encrypted data in its `CipherValue` child element, or it contains a reference to the encrypted data through the `CipherReference` child element.

Example of XML security

Consider the following XML document which contains information about a business transaction. The information includes:

- The customer

- The product
- Credit card information
- Address information

```
<TransactionInfo xmlns='http://www.whizlabs.com/trans'>
  <Customer>Peter John</Customer>
  <Products>
    <Product>
      <Name>XML Exam Simulator</Name>
      <Price>90</Price>
    </Product>
  </Products>
  <Address>
    <Street>75 Bandeny Street</Street>
    <AptNo>765</AptNo>
    <City>Boston</City>
    <State>MA</State>
    <ZipCode>02134</ZipCode>
  </Address>
  <CreditCard >
    <Name>Peter</Name>
    <Number>4019 2445 0277 5567</Number>
    <Provider>Example Bank</Provider>
    <Expiration>
      <month>02</month>
      <year>2005</year>
    </Expiration>
  </CreditCard>
</TransactionInfo>
```

You might process this document through different applications that do not require to know all the information present in the document. For example, the shipping department only needs to extract the `Address` element information and does not need access to the more sensitive credit card information. So it's possible to encrypt the credit card information using the `EncryptedData` element. The `CreditCard` element is then replaced by the `EncryptedData` element:

```
<EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
  xmlns='http://www.w3.org/2001/04/xmlenc#' >
  <CipherData>
    <CipherValue>A23B45C56</CipherValue>
  </CipherData>
</EncryptedData>
```

So the modified structure of the above XML document is:

```
<TransactionInfo xmlns='http://www.whizlabs.com/trans'>
  <Customer>Peter John</Customer>
```

```
<Products>
  <Product>
    <Name>XML Exam Simulator</Name>
    <Price>90</Price>
  </Product>
</Products>

<Address>
  <Street>75 Bandeny Street</Street>
  <AptNo>765</AptNo>
  <City>Boston</City>
  <State>MA</State>
  <ZipCode>02134</ZipCode>
</Address>

<EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
  xmlns='http://www.w3.org/2001/04/xmlenc#' >
  <CipherData>
    <CipherValue>A23B45C56</CipherValue>
  </CipherData>
</EncryptedData>

</TransactionInfo>
```

Exercise for XML security

In an encrypted XML file, which one of the following elements is used to transport the encryption keys from the sender to the receiver?

- A. EncryptionMethod
- B. CipherValue
- C. EncryptionProperties
- D. EncryptedKey

Correct choice: D

Explanation:

The `EncryptedKey` element transports the encryption keys from the sender to the receiver.

`EncryptionMethod` specifies the algorithm that will be applied to the cipher data.

`CipherValue` contains the encrypted data.

`EncryptionProperties` provides additional information about the encrypted data.

Exam tip for security

The exam has few if any questions pertaining to XML security. Even though it's possible to skip this topic for the exam, basic knowledge of XML encryption (including the structure of the `EncryptedData` element and its usage) will help you answer any questions regarding XML security.

Section 7. Wrap up

Summary

This part of the tutorial series covered the following technologies:

- XML
- Document Type Definitions (DTDs)
- W3C XML Schema
- Web services
- Security

You'll find very few questions regarding well-formedness and valid XML. For most questions regarding DTDs, you want to remember the limitations of DTDs.

Most of the questions in the XML exam are based on schema, so pay extra attention to the section dedicated to that topic. And do remember the advantages of schema over DTDs.

The exam asks few if any questions about XML security. For questions about Web services, remember the core technologies that comprise them: SOAP, WSDL, and UDDI.

You also need to understand how all these technologies work apart from memorizing their syntax. To answer scenario-based questions, know the advantages and disadvantages of one technology over the other. Most of the time, you will be asked to choose the one best solution from a set of possible solutions. So it's important to understand the limitations and advantages of all these technologies.

The second part of this tutorial series will cover the remaining XML technologies

required for the exam -- namely XPath, XSLT, XLink, XPointer, XSL-FO, CSS, SAX, and DOM.

Resources

- Read [Part 2](#) (May 2005) and [Part 3](#) (August 2005) of this three-part developerWorks tutorial series on preparing for the IBM XML certification exam.
- Visit [the Worldwide Web Consortium \(W3C\) site](#), which contains the specifications for XML and many of its related technologies. This is probably the best place to get in-depth information on XML.
- Check out the [developerWorks XML zone](#) for a wide range of articles, tutorials, columns, forums and other resources that cover extensible technologies in detail. While you're there, take a look at the developerWorks [SOA and Web services zone](#) for more on those topics.
- Start with the basics. Read Doug Tidwell's popular "[Introduction to XML](#)" tutorial here on developerWorks (August 2002).
- Confused by all the XML standards out there? Uche Ogbuji's developerWorks article series on XML standards can help you sort through it all:
 - [Part 1](#) -- The core standards (January 2004)
 - [Part 2](#) -- XML processing standards (February 2004)
 - [Part 3](#) -- The most important vocabularies (February 2004)
 - [Part 4](#) -- Detailed cross-reference of the most important XML standards (March 2004)
- Read the [XML Schema Datatypes Recommendation](#).
- Find out more about [XML Signature](#) at the W3C.
- Visit [www.w3schools.org](#) for good tutorials about various XML technologies.
- Try [www.zvon.org](#), another good site that offers tutorials on these technologies.
- Take a closer look at namespaces -- read [Ronald Bourret's XML Namespaces FAQ](#) as well as Uche Ogbuji's developerWorks article "[Use XML namespaces with care](#)" (April 2004).
- Read Kurt Cagle's "[XSD Schema Tricks and Tips](#)" for more on W3C XML Schema.
- Check out the [XML Certification forum](#) at JavaRanch.
- Find out more about the [IBM Certified Developer in XML and Related Technologies exam](#). This page includes:
 - Details on the job role and target audience for whom the certification was built

- Recommended prerequisites for the knowledge and skills one should possess before considering this certification
 - The test objectives and the skills measured by the exam
 - Recommended educational resources to prepare you for the test, based on the test objectives
 - A pre-assessment/sample test to gauge your readiness for the actual exam
-
- Check out this [introductory article on XML certification](#) by the author of this tutorial.
 - Practice and assess your knowledge level using the [Whizlabs XML Exam Simulator](#).
 - Read [Professional XML, 2nd Edition](#) , the prescribed book for the XML Certification Exam that covers these technologies to the extent required by the exam.
 - Also pick up the [XML Bible](#) by Elliotte Rusty Harold. This is a good book to get you started on XML technologies.
 - Cover all the relevant topics except security and XSL-FO in [Essential XML Quick Reference](#) .

About the authors

Pradeep Chopra

Pradeep Chopra is the cofounder of [Whizlabs Software](#), a global leader in IT skill assessment and certification exam preparation. A graduate of the Indian Institute of Technology (Delhi), Pradeep has consulted individuals and organizations across the globe on the values and benefits of IT certifications.

Hari Vignesh Padmanaban

Hari Vignesh Padmanaban is the author of the [Whizlabs XML Simulator for XML](#). He has MCP, SCJP, SCWCD, SCBCD, Object Oriented Analysis and Design using UML, and XML and Related Technologies certifications to his credit. He is currently working as a software engineer for [Invensys Foxboro](#).