

XML Document Structure

Objectives

After completing this lesson, you should be able to do the following:

- Describe the benefits of XML
- Create an XML declaration
- Assemble the components of an XML document
- Declare and apply XML Namespaces
- Validate XML documents by using XML Schemas
- Create XML Schemas



Course Roadmap

**Application Development
Using Webservices [SOAP
and Restful]**



Lesson 1: Introduction to Web Services



Lesson 2: Creating XML Documents



You are here!



Lesson 3: Processing XML with JAXB



Lesson 4: SOAP Web Services Overview



Lesson 5: Creating JAX-WS Clients

Course Roadmap

Application Development Using Webservices [SOAP and Restful]



Lesson 6: Exploring REST Services



Lesson 7: Creating REST Clients



Lesson 8: Bottom Up JAX Web Services



Lesson 9: Top Down JAX Web Services



Lesson 10: Implementing JAX RS Web Services

Course Roadmap

**Application Development
Using Webservices [SOAP
and Restful]**



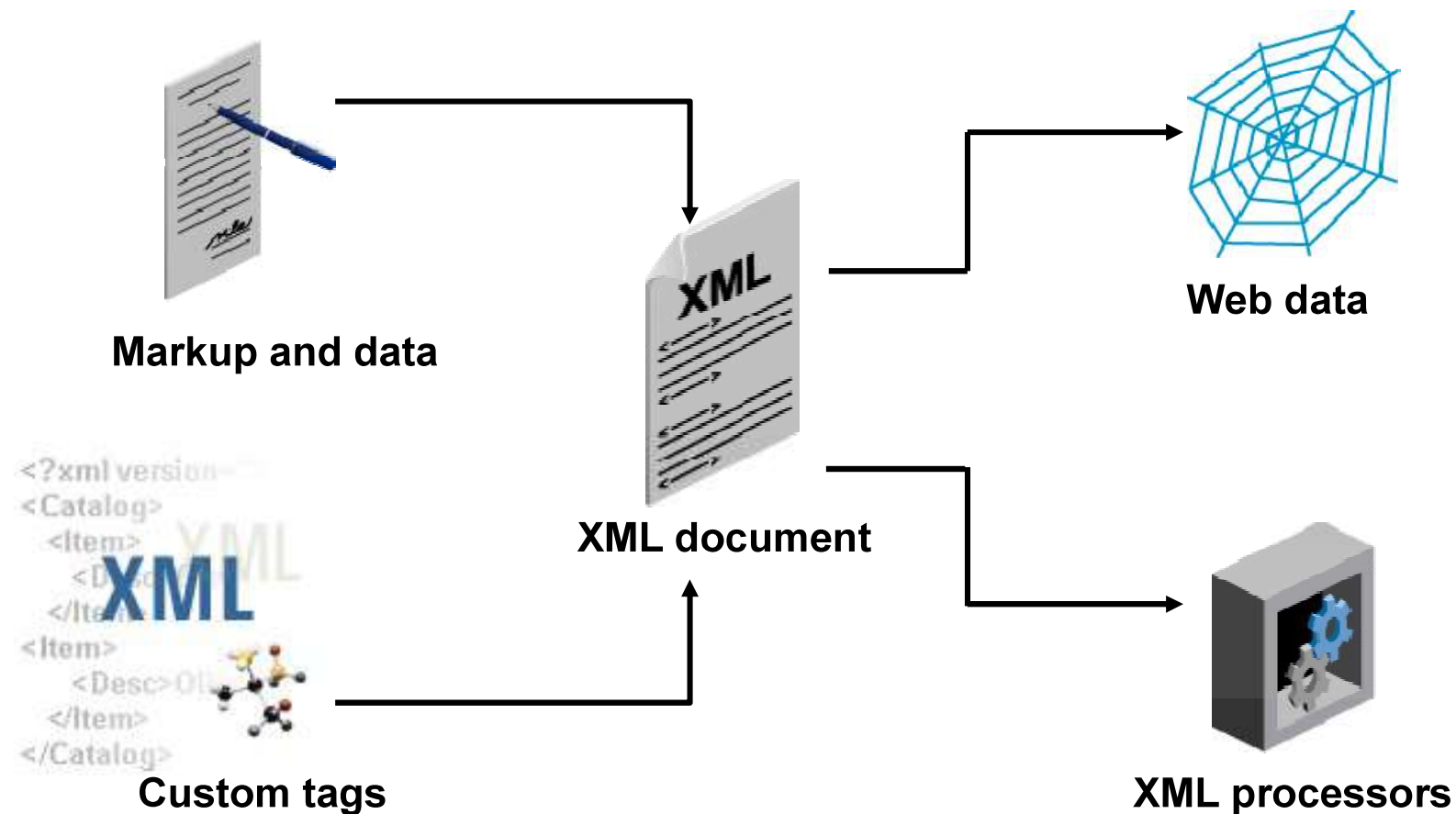
Lesson 11: Web Service Error Handling



Lesson 12: Java EE Security and Securing JAX WS

Extensible Markup Language

Extensible Markup Language (XML) describes data objects called XML documents that are composed of markup and data.



Advantages of Using XML

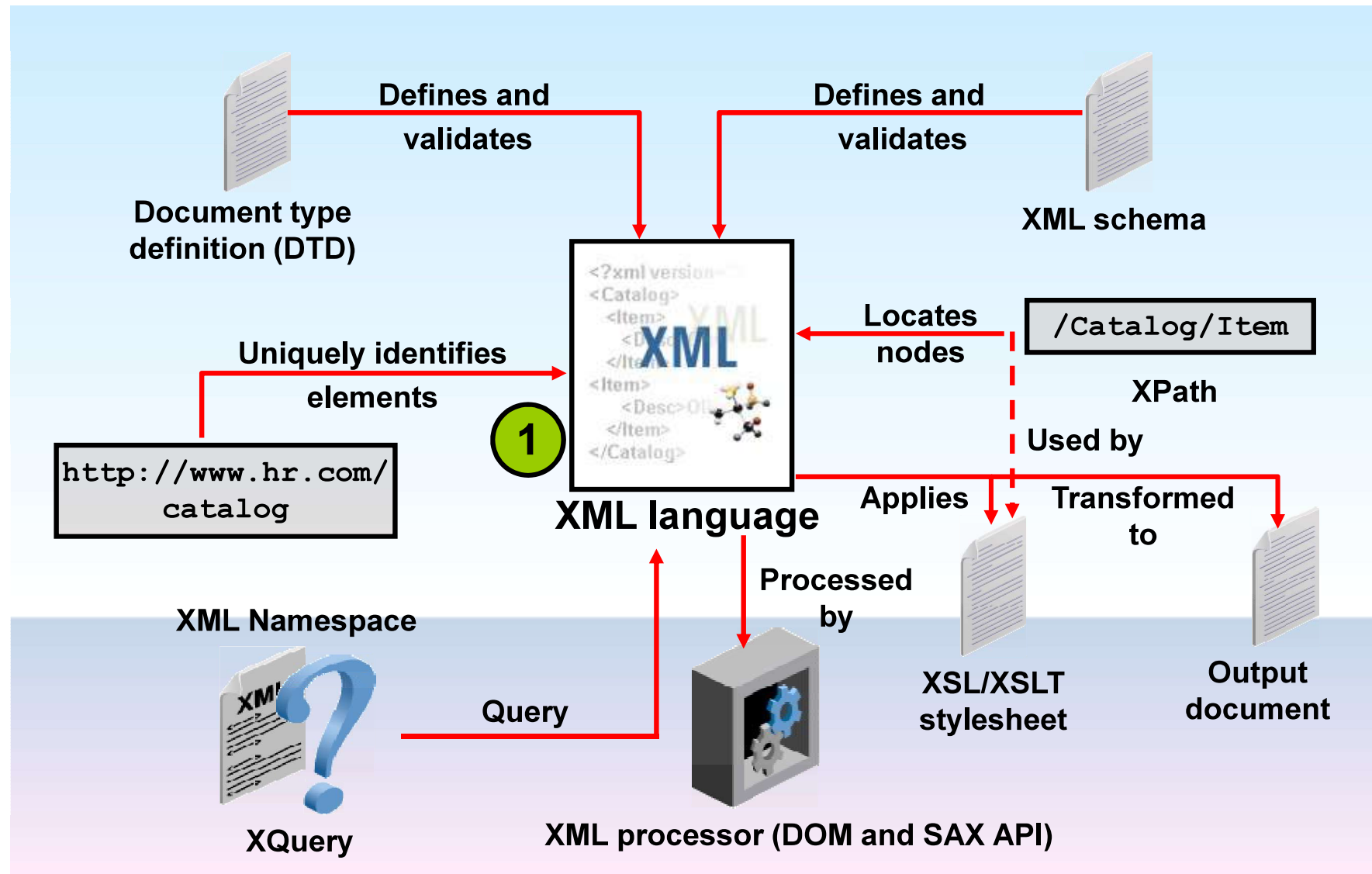
XML enables:

- A simple and extensible way to describe data
- The ability to interchange data
- Simplified business-to-business communication
- Writing of smart agents
- The ability to perform smart searches

Sample XML document:

```
<?xml version="1.0"?>
<books>
  <title>Building Oracle XML Applications</title>
  <title>Oracle XML Handbook</title>
  <title>Beginning XML Second Edition</title>
</books>
```

XML Standards



Importance of XML in Web Services

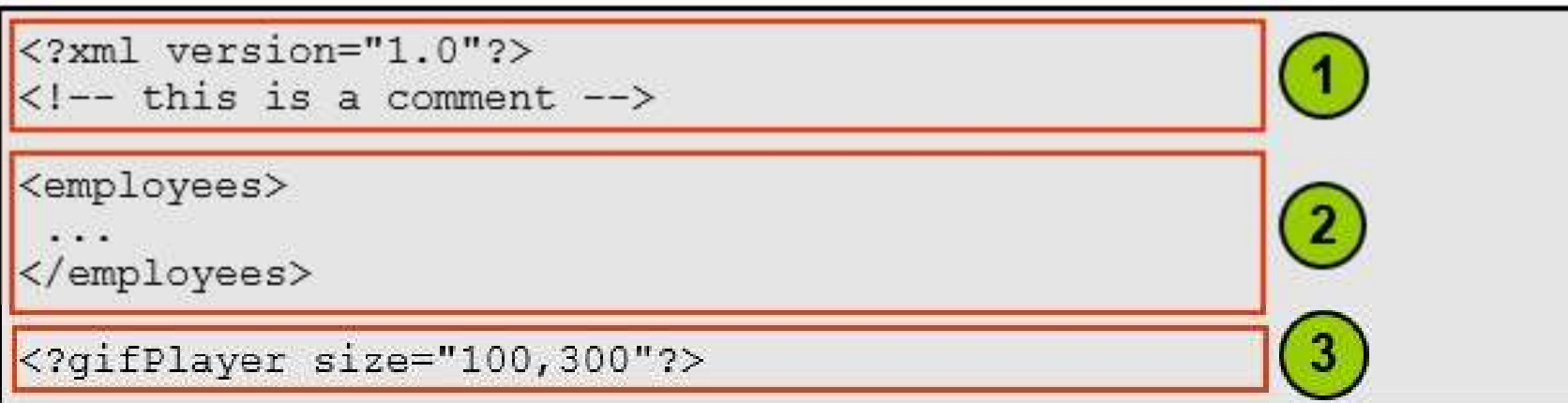
- Web services standards such as SOAP, WSDL, and UDDI are XML based.
- RESTful web services may support multiple data formats including XML.
 - Java RESTful web service clients might prefer XML over JSON due to the amount of XML supporting libraries.



XML Document Structure

An XML document contains the following parts:

1. Prologue (or preamble)
2. Root element: Only one root element per document
3. Epilogue (not common)



XML Declaration

XML documents must start with an XML declaration.

An XML declaration:

- Looks like a processing instruction with the `xml` name
- Example:

```
<?xml version="1.0" encoding="UTF-8"?>
```

- Is optional in XML 1.0 but mandatory in XML 1.1
- Must contain the version attribute
- May (optionally) include the following:
 - Encoding attribute
 - Stand-alone attribute

Components of an XML Document

XML documents comprise storage units that contain:

- Parsed character data (PCDATA), which includes:
 - Markup (elements, attributes, and entities) used to describe the data it contains
 - Character data described by the markup

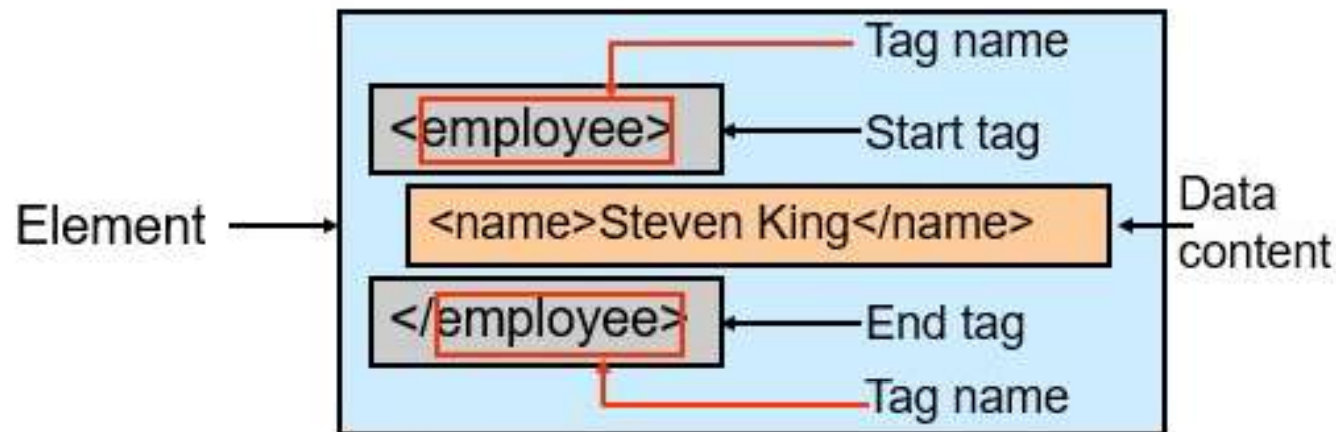
```
<?xml version="1.0" encoding="UTF-8"?> <employees>  
  <employee id="100">  
    <name>Rachael O'Leary</name>  
  </employee>  
</employees>
```

- Unparsed character data (CDATA): Textual or binary information (graphic and sound data) taken as entered

```
<![CDATA[ ...unparsed data... ]]>
```

XML Elements

- An XML element has a start tag, an end tag, and optional data content.
- Tag names are case-sensitive (must be identical).



- Empty elements:
 - Contain no data
 - May appear as a single tag

`<initials></initials>`

`<initials/>`

Markup Rules for Elements

- There is one root element, which is sometimes called the top-level or document element.
- All elements:
 - Must have matching start and end tags, or be a self-closing tag (an empty element)
 - Can contain nested elements so that their tags do not overlap
 - Have case-sensitive tag names that are subject to naming conventions (starting with a letter, no spaces, and not starting with the letters xml)
 - May contain white space (spaces, tabs, new lines, and combinations of them) that is considered part of the element data content

XML Attributes

An XML attribute is a name-value pair that:

- Is specified in the start tag, after the tag name

```
<?xml version="1.0" encoding=" UTF-8 "> <employees>  
  <employee id="100" name='Rachael O&apos;Leary'>  
    <salary>1000</salary>  
  </employee>  
</employees>
```

- Has a case-sensitive name
- Has a case-sensitive value that must be enclosed within matching single or double quotation marks
- Provides additional information about the XML document or XML elements

Naming Convention Rules

Element and attribute names:

- Can contain letters A–Z and a–z
- Can contain numbers 0–9
- Can contain ideograms and non-English characters
- Can contain an _ (underscore), – (dash), and . (period)
- Must start with letters or an “_”
- Cannot start with a number, punctuation character, or the letters xml (or XML, Xml, and so on)
- Cannot contain spaces
- Can use any name; no words are reserved

Using Elements Versus Attributes

```
<?xml version="1.0"?>
  <employees>
    <employee>
      <id>100</id>
      <last_name>King</last_name>
      <salary>24000</salary>
    </employee>
  </employees>
```

1

Elements

```
<?xml version="1.0"?>
  <employees>
    <employee id="100" last_name="King"
      salary="24000">
  </employee>
</employees>
```

Attributes

2

XML Character Data (CDATA)

The CDATA section:

- Is not read by a parser
- Is passed to the application without change
- Is used to contain text that has several XML-restricted characters such as <, >, &, ", and '

```
<script>  
if(a &gt; b &amp; b &gt; c)  
{print("&quot; a is greater than c &quot;;)}  
</script>
```

1

```
<script><![CDATA[  
if(a > b & b > c)  
{print("a is greater than c")}  
]]>  
</script>
```

2

XML Comments

XML comments:

- Start with `<!--`
- End with `-->`
- May appear anywhere in the character data of a document, and before the root element
- Are not elements and can occupy multiple lines
- May not appear inside a tag or another comment

```
<?xml version="1.0" encoding="UTF-8"?>  
<!-- Comment: This document has information about  
employees in the company -->  
<employees>  
  <name>Steven King</name> <!-- Full name -->  
</employees>
```

Well-Formed XML Documents

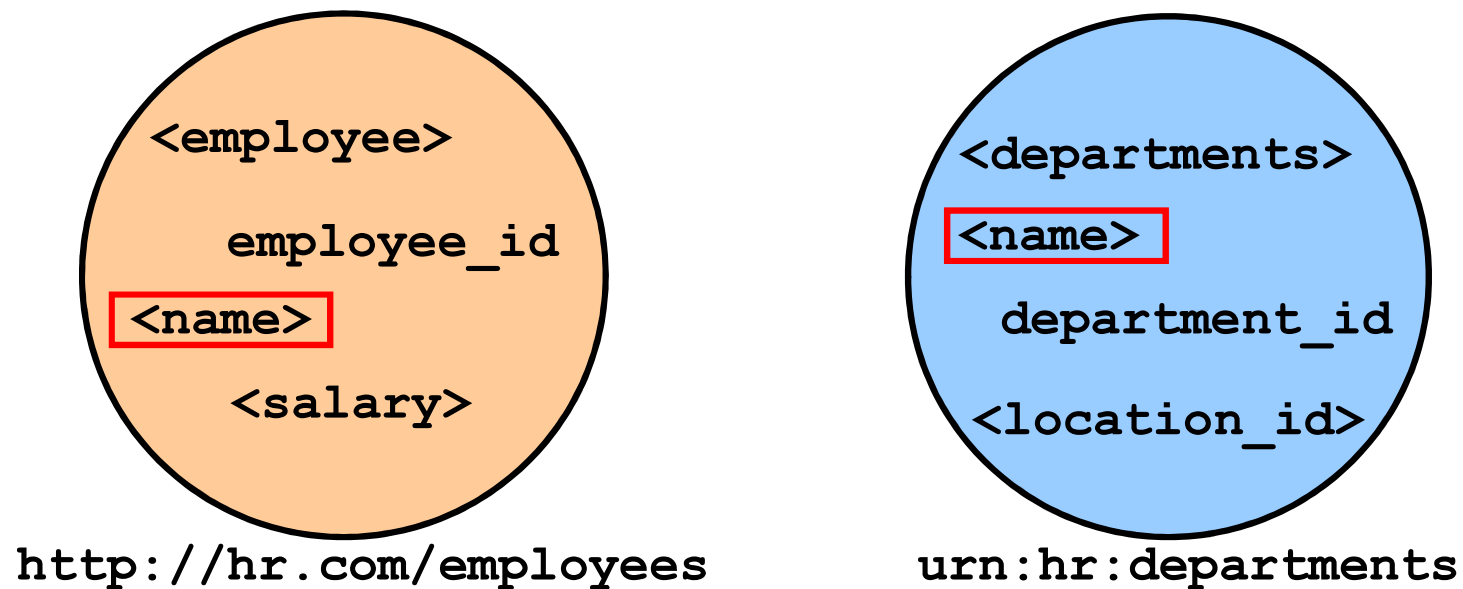
Every XML document must be well-formed:

- An XML document must have one root element.
- An element must have matching start and end tag names unless it is an empty element.
- Elements can be nested but cannot overlap.
- All attribute values must be quoted.
- Attribute names must be unique in the start tag of an element.
- Comments and processing instructions do not appear inside tags.
- The < or & special characters cannot appear in the character data of an element or attribute value.

What Is an XML Namespace?

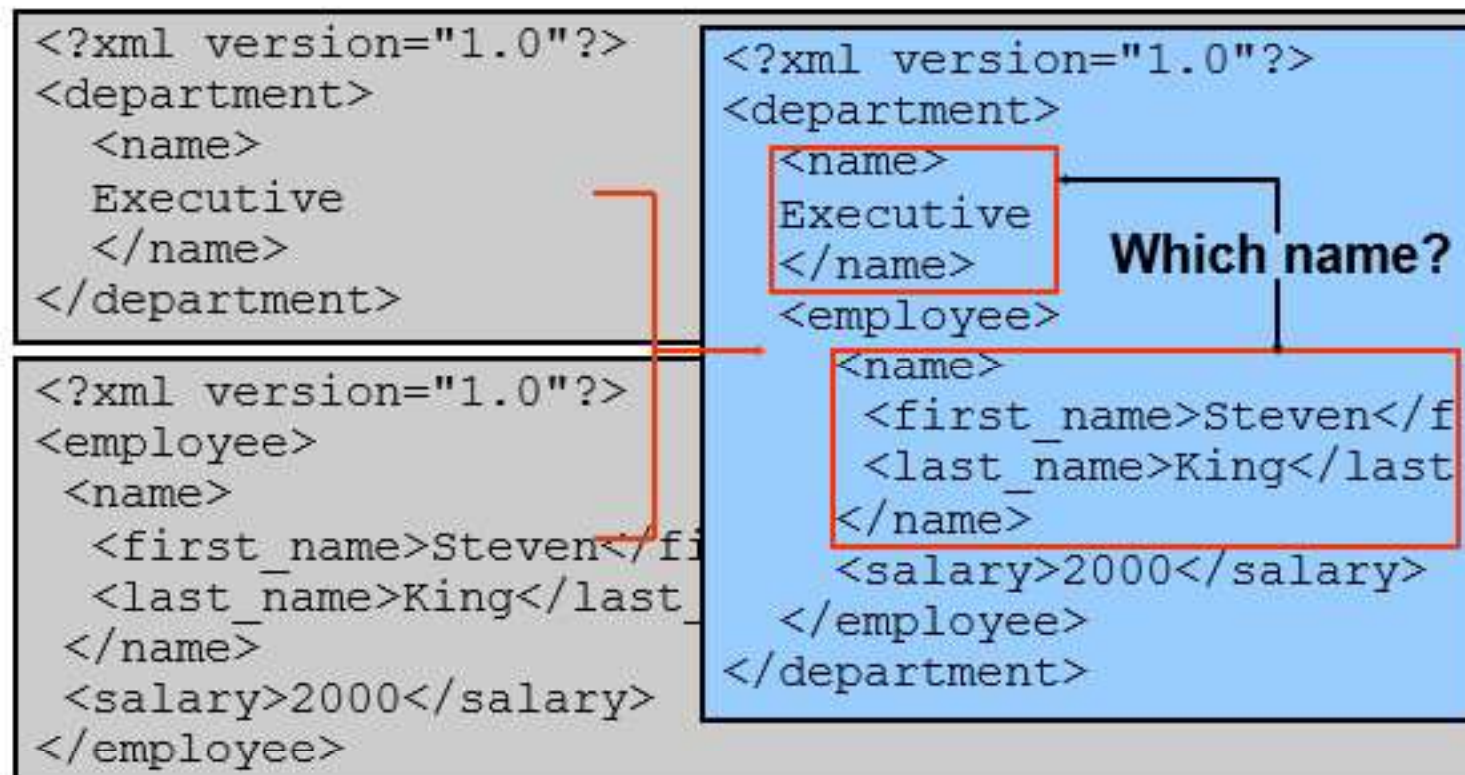
An XML Namespace:

- Is identified by a case-sensitive Internationalized Resource Identifier (IRI) reference (URL or URN)
- Provides universally unique names for a collection of names (elements and attributes)



Why Use XML Namespaces?

Using an XML Namespace resolves name collisions or ambiguities in an XML document.



XML Namespaces

XML Namespaces

To successfully combine XML vocabularies with no confusion there must be a way to uniquely identify each element.

- Java classes like `Person` are placed in packages.
- Java packages are used to create namespaces.
- XML did not originally support namespaces.
- XML Namespaces became a W3C recommendation in 1999.

Declaring XML Namespaces

Declare an XML Namespace:

- With the `xmlns` attribute in an element start tag:
 - Assigned an IRI (URL, URI or URN) string value
 - Provided with an optional namespace prefix
- With a namespace prefix after `xmlns:` to form qualified element names:

```
<dept:department  
  xmlns:dept="urn:hr:department-ns">  
  ...  
</dept:department>
```

- Without a prefix to form a “default namespace”:

```
<department xmlns="http://www.hr.com/departments">  
  ...  
</department>
```


XML Namespace Prefixes

A namespace prefix:

- May contain any XML character except a colon
- Can be declared multiple times as attributes of a single element, each with different names whose values can be the same or a different string
- Can be overridden in a child element by setting the value to a different string. For example:

```
<?xml version="1.0"?>
<emp:employee xmlns:emp="urn:hr:employee-ns">
  <emp:last_name>King</emp:last_name>
  <emp:address xmlns:emp="urn:hr:address-ns">
    500 Oracle Parkway
  </emp:address>
</emp:employee>
```

Example of XML Namespace Declarations

```
<?xml version="1.0"?>
<department xmlns="urn:hr:department-ns"
             xmlns:emp="urn:hr:employee-ns">
  <name>Executive</name>
  <emp:employee>
    <emp:name>
      <emp:first_name>Steven</emp:first_name>
      <emp:last_name>King</emp:last_name>
    </emp:name>
  </emp:employee>
  <emp:employee>
    <emp:name>
      <emp:first_name>Neena</emp:first_name>
      <emp:last_name>Kochhar</emp:last_name>
    </emp:name>
  </emp:employee>
</department>
```

XML Schema Document: Example

- A simple XML Schema uses:
 - A required XML Namespace string, with an `xs` prefix, `http://www.w3.org/2001/XMLSchema`
 - The `<schema>` element as its document root
 - The `<element>` element to declare an element

```
<?xml version="1.0"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="departments" type="xs:string"/>
</xs:schema>
```

XSD

- A valid XML instance document:

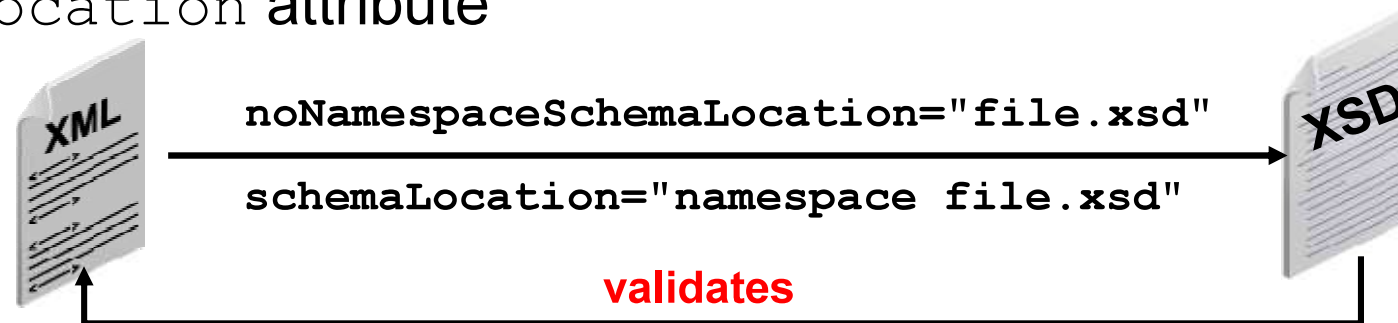
```
<?xml version="1.0"?>
<!-- The element cannot contain child elements -->
<departments>
Finance
</departments>
```

XML

Validating an XML Document

In the XML instance document, use the XML Namespace `http://www.w3.org/2001/XMLSchema-instance`, and reference the XML Schema by using either of the following:

- `noNamespaceSchemaLocation` attribute
- `schemaLocation` attribute



```
<?xml version="1.0"?>
<departments xmlns:xsi=
  "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="departments.xsd">
  Finance
</departments>
```

XML

Referencing an XML Schema

- The XML Schema defines the `targetNamespace` value.

```
<?xml version="1.0"?> <!-- departments.xsd -->
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.hr.com/departments">
  <xs:element name="departments" type="xs:string"/>
</xs:schema>
```

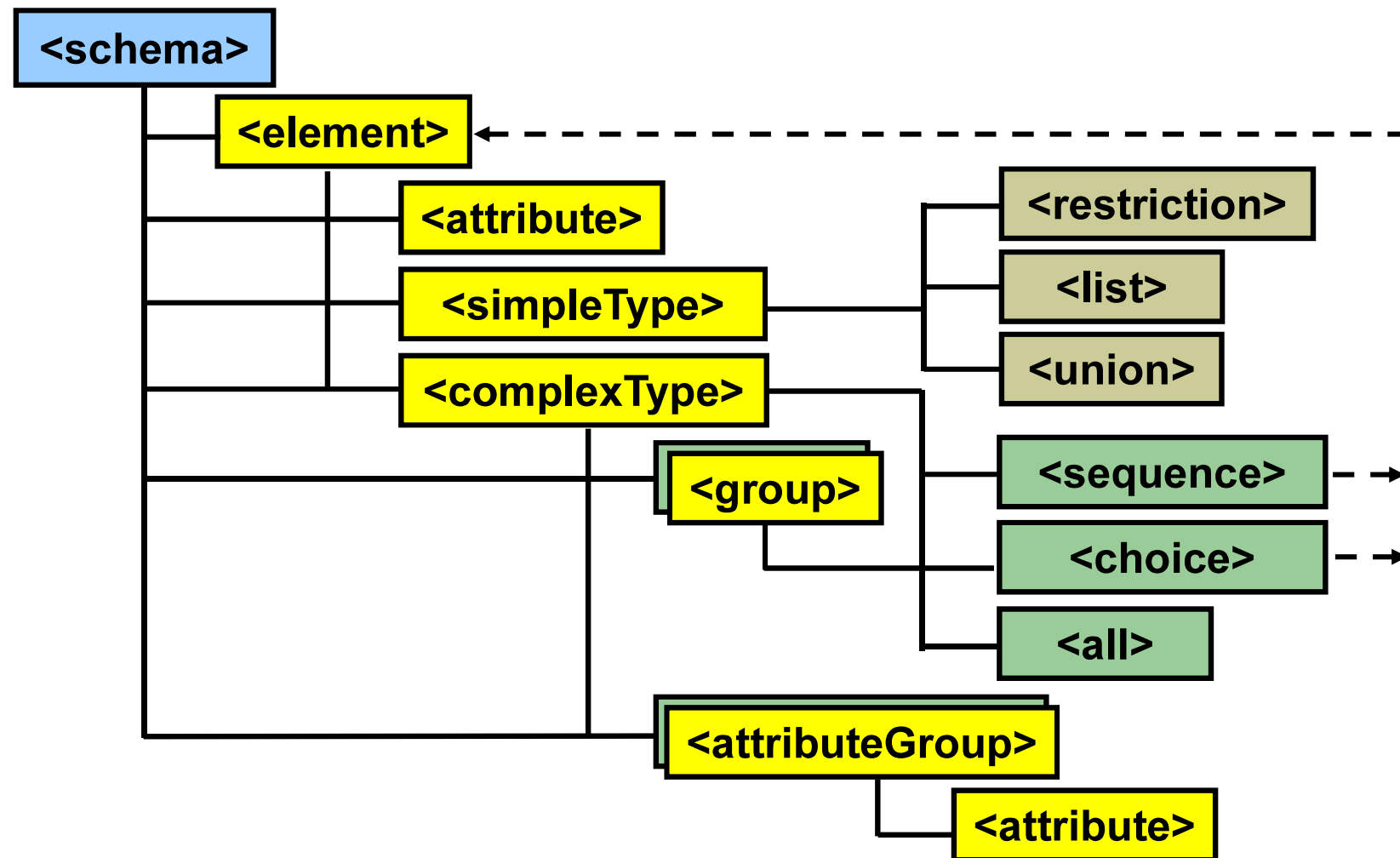
XSD

- The XML document references `targetNamespace` in `schemaLocation` and the default namespace.

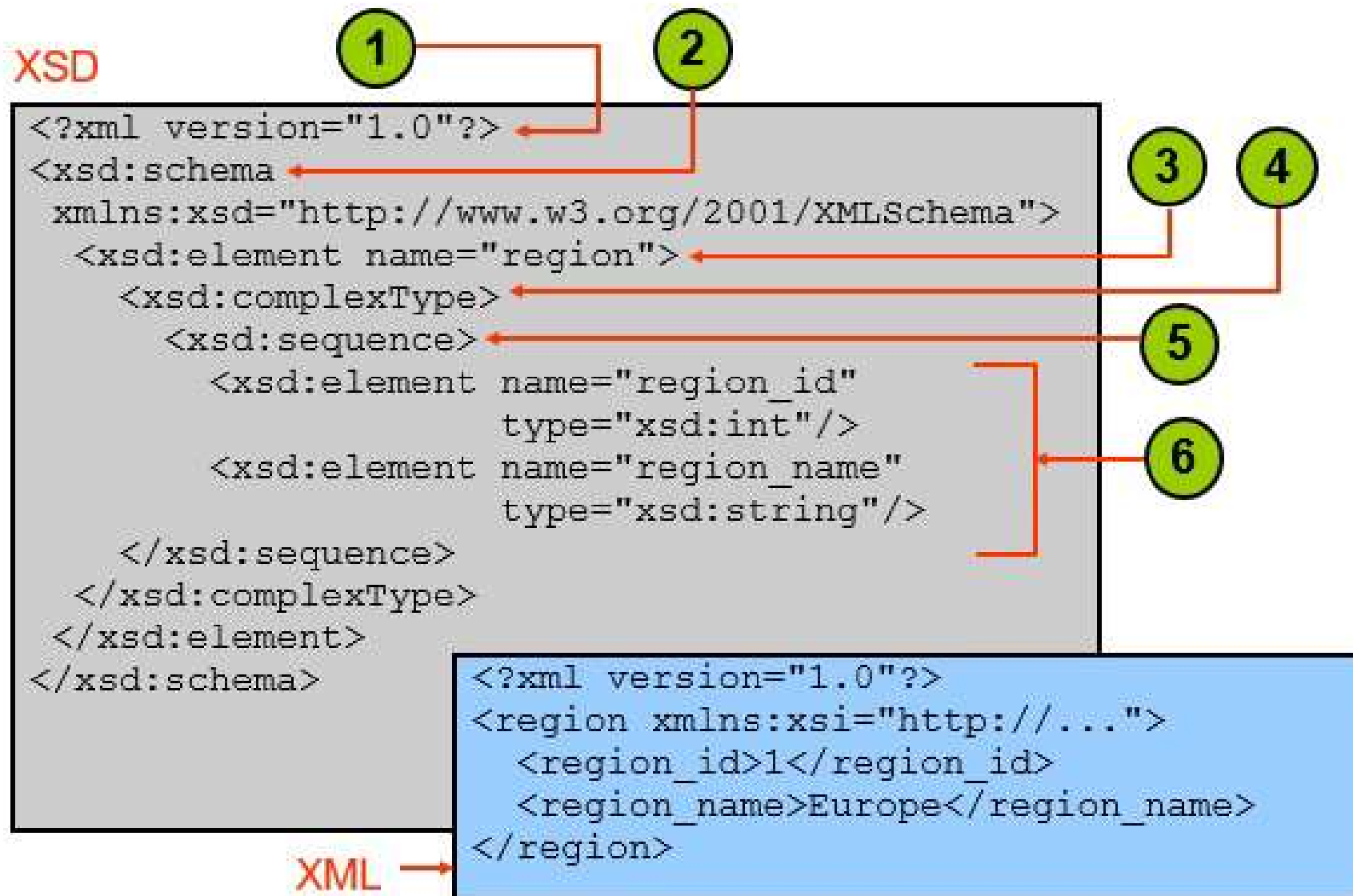
```
<?xml version="1.0"?> <!-- XML document -->
<departments xmlns = "http://www.hr.com/departments"
  xmlns:xsi=
    "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.hr.com/departments departments.xsd">
  Finance
</departments>
```

XML

Components of an XML Schema



XML Schema Components: Example



Global and Local Declarations

➤ Global declarations:

- Appear as direct children of the `<schema>` element
- Can be reused within the XML Schema document

➤ Local declarations:

- Are valid in the context in which they are defined
- Are not children of the `<schema>` element
- Can reference global type declarations by using a namespace prefix

```
<schema ...  
  xmlns:t="http://..."  
  <!-- Global -->  
  <element name="A" />  
  <complexType name="B">  
    <!-- Local -->  
    <sequence>  
      <element ref="t:A" />  
      <element name="C" />  
    </sequence>  
  </complexType>  
</schema>
```


Built-In XML Schema Data Types

The XML Schema language provides built-in data types such as:

- `string` type
- `int` type
- `decimal` type
- Many others (boolean, float, and so on)
- Examples:

```
<xsd:element name="employee_id" type="xsd:int"/>
```

XSD

```
<employee_id>100</employee_id>
```

XML

```
<xsd:element name="salary" type="xsd:decimal"/>
```

XSD

```
<salary>1000.50</salary>
```

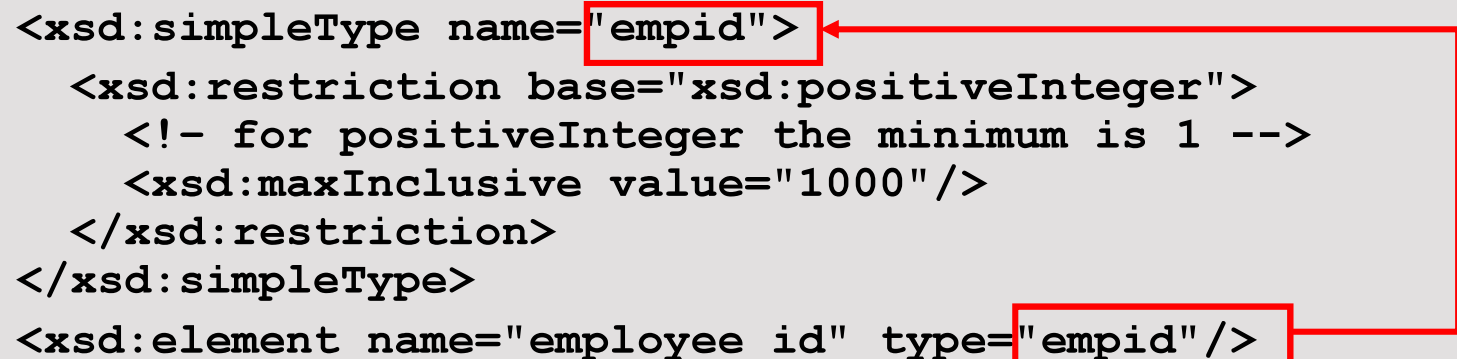
XML

Declaring a `<simpleType>` Component

`<simpleType>`:

- Is a derived type that extends built-in or other types
- Provides three primary derived types:
 - `<restriction>`
 - `<list>`
 - `<union>`
- Has facets (properties) such as `maxInclusive`

```
<xsd:simpleType name="empid">  
  <xsd:restriction base="xsd:positiveInteger">  
    <!-- for positiveInteger the minimum is 1 -->  
    <xsd:maxInclusive value="1000"/>  
  </xsd:restriction>  
</xsd:simpleType>  
<xsd:element name="employee_id" type="empid"/>
```



Using an Enumeration Restriction

➤ Declaring an enumeration:

```
<xsd:simpleType name="mgrTypes">  
  <xsd:restriction base="xsd:string">  
    <xsd:enumeration value="AC_MGR" />  
    <xsd:enumeration value="FI_MGR" />  
  </xsd:restriction>  
</xsd:simpleType>
```

Declaring `<complexType>` Components

The `<complexType>` declaration:

```
<xsd:complexType name="..." mixed="true | false">  
  ...  
</xsd:complexType>
```

- Must be identified by a name attribute if it is global; otherwise, it is an anonymous complex type
- Provides a content model that can contain:
 - Simple content
 - A `<sequence>` declaration
 - A `<choice>` declaration
 - A reference to a global `<group>`
 - An `<all>` declaration
- Can allow mixed or empty content

Declaring a <sequence>

- Defines an ordered sequence of elements
- Must be contained within a <complexType>

```
<xsd:element name="department">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="department_id"
                    type="xsd:int"/>
      <xsd:element name="department_name"
                    type="xsd:string"/>
      <xsd:element name="manager_id"
                    type="xsd:int" minOccurs="0"/>
      <xsd:element name="location_id"
                    type="xsd:int" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Declaring a <choice>

- Defines a choice of alternative elements
- Must also be contained within a <complexType>

```
<xsd:complexType name="employeeType">
  <xsd:choice>
    <xsd:element name="full_time"
                  type="xsd:string"/>
    <xsd:element name="part_time"
                  type="xsd:string"/>
  </xsd:choice>
</xsd:complexType>
<xsd:element name="employee">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="full_name"
                    type="xsd:string"/>
      <xsd:element name="contract"
                    type="employeeType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

The diagram consists of two red rectangular boxes. The first box is positioned around the first `<xsd:choice>` block in the XML code, which contains two `<xsd:element>` declarations for `full_time` and `part_time`. The second box is positioned around the `<xsd:element name="contract" type="employeeType"/>` line. A red line originates from the right side of the first box, extends horizontally to the right, then turns vertically downwards, and finally turns horizontally to the left, ending with an arrowhead pointing at the `type="employeeType"` attribute in the second box. This illustrates that the `employeeType` is a recursive reference to the first `<xsd:choice>` block.

Declaring an Empty Element

- Can be declared using a `<complexType>` declaration without any elements or a content model
- Typically contains attributes

```
<xsd:element name="departments">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="department"
        maxOccurs="unbounded" />
      <xsd:complexType>
        <xsd:attribute name="department_id"
          type="xsd:int" />
        <xsd:attribute name="department_name"
          type="xsd:string" />
        <xsd:complexType>
        </xsd:complexType>
      </xsd:complexType>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Declaring Attributes

Declare an `<attribute>`:

- Identified by the name attribute
- With the type attribute restricted to built-in or user-defined simple types

```
<xsd:attribute name="department_id"  
               type="xsd:string"/>
```

Full Syntax

```
<attribute  
  name="name-of-attribute"  
  type="global-type | built-in-type"  
  ref="global-attribute-declaration"  
  form="qualified | unqualified"  
  used="optional | prohibited | required"  
  default="default-value"  
  fixed="fixed-value">
```


The tags and attributes allowed in an XML document can be constrained by:

- a. DTDs
- b. JAXB
- c. XML Namespaces
- d. XML Schemas

Quiz

An XML schema simpleType may contain:

- a. sequence
- b. choice
- c. restriction
- d. enumeration

Resources

Topic	Website
Extensible Markup Language (XML) 1.0	http://www.w3.org/TR/REC-xml/
Namespaces in XML 1.0	http://www.w3.org/TR/REC-xml-names/
XML Namespaces	http://www.jclark.com/xml/xmlns.htm
XML Schema Part 0: Primer Second Edition	http://www.w3.org/TR/xmlschema-0/
W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures	http://www.w3.org/TR/xmlschema11-1/
W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes	http://www.w3.org/TR/xmlschema11-2/

Summary

In this lesson, you should have learned how to:

- Describe the benefits of XML
- Create an XML declaration
- Assemble the components of an XML document
- Declare and apply XML Namespaces
- Validate XML documents by using XML Schemas
- Create XML Schemas



Practice 2: Overview

This practice covers the following topics:

- Exploring PlayingCards Project
- Creating a XML Document
- Creating a XML Schema
- Using XML Namespaces

