OBJECTIVES

Introduction to XML

LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

- Explain about Markup Languages
- Identify the Core Concepts of XML
- Write XML Syntax
- Distinguish between Well Formed XML and Valid XML

CONCEPT

Markup Languages

UNDERSTANDING MARKUP LANGUAGES

- Markup is information added to a text/document to enhance its meaning
- Helps to identify the pieces of text in a document and the relationship between them
- Markup Language is a set of symbols placed in the text of a document to:
  - Provide clear boundary and label related to pieces of text
  - Describe the document’s logical structure
  - Provide information/metadata about the pieces of text

UNDERSTANDING MARKUP LANGUAGES

- Twentieth century saw huge inflow of information/data, but early electronic formats were more concerned with presentation
- Use was limited to viewing on screen or printing
- It was difficult to write programs to:
    - Search and extract information
    - Make changes for different applications
- Markup helped electronic documents overcome these drawbacks:
    - Computer programs processing the document can distinguish different pieces of data
    - Helps computers treat data differently based on the markup
    - Could be used for presentation, processing or structuring

STANDARD GENERALIZED MARKUP LANGUAGE (SGML)

Markup languages like generic coding, Tex, Scribe, GML were created for specific uses

- Created based on the idea that markup should be focused on the structural aspects of a document
- A meta-language from which many markup languages are derived
- Specifies the standard syntax for including markups in document
- Describes **WHAT** markups are allowed **WHERE**
- Made as an international standard by ISO

STANDARD GENERALIZED MARKUP LANGUAGE (SGML)

- Although SGML was used for defining markup languages, it had some pitfalls
  - Very huge and complicated
  - Looser syntax and highly flexible (software built to process it was complex)
  - Acceptance/usefulness limited to large organizations
- A smaller and simpler markup language was required for the web
- A group of companies/organization (W3C) began to work on markup language that had:
  - Flexibility of SGML
  - Simplicity of HTML

CONCEPT

XML

WHAT IS XML?

- Stands for e**X**tensible **M**arkup **L**anguage
- Text based markup language derived from SGML
- Designed to describe and organize data
- Designed to carry data
- Defines set of rules for encoding documents
- Both human-readable and machine-readable
- Created by World Wide Web Consortium (W3C)

```
<?xml version="1.0" encoding="UTF-8"?>
<product>
    <productcode>101</productcode>
    <productname>Laptop</productname>
</product>
```

USE OF XML

- ➢ Extensible
  - Allows to create your own description/structure of data that suit your application needs
- ➢ Provides clear distinction between data and its presentation
  - Allows to store data in a logical structure irrespective of its presentation
  - Data stored in XML can be presented in any way as per the requirement of application/user
- ➢ Simplifies data exchange
  - Allows easy sharing of data between different applications / platforms
- ➢ Easy to extract data as it is well organized
- ➢ Self describing data which makes it easy to understand
- ➢ An open standard not proprietary

XML VS HTML

XML

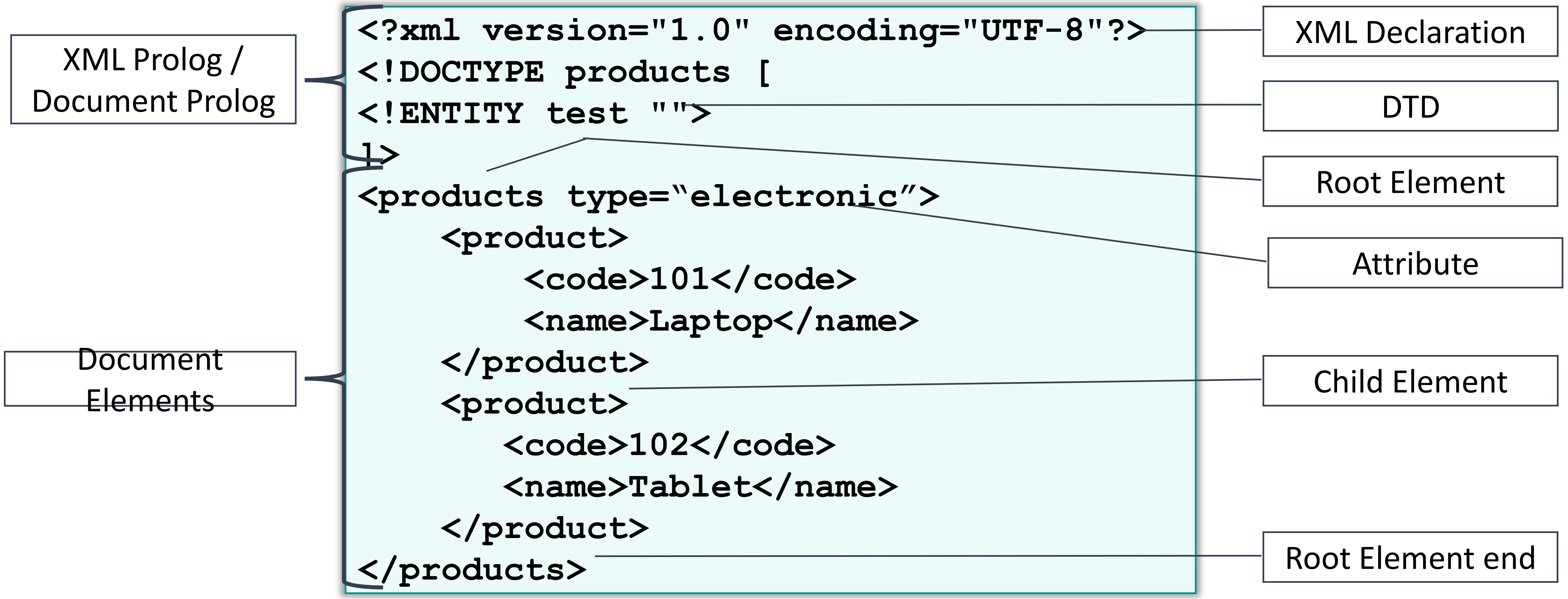
- Markup language
- Facilitates the exchange and storage of data
- Allows to create new tags. That's why it is eXtensible
- Focuses on describing and structuring data
- Stricter syntax
- Any application software can use XML

HTML

- Markup language
- Used to present data in a specified manner
- Predefined tags used for presenting data in web browser
- Focuses on presenting data
- Looser syntax
- HTML is specifically designed for web applications

XML STRUCTURE

XML documents use a self-describing and simple syntax



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE products [
<!ENTITY test "">
]>
<products type="electronic">
    <product>
        <code>101</code>
        <name>Laptop</name>
    </product>
    <product>
       <code>102</code>
       <name>Tablet</name>
    </product>
</products>
```

XML STRUCTURE

XML documents form a tree structure

Must contain a root element which is the “parent" of all other elements

Tree starts at the root and branches to the lowest level

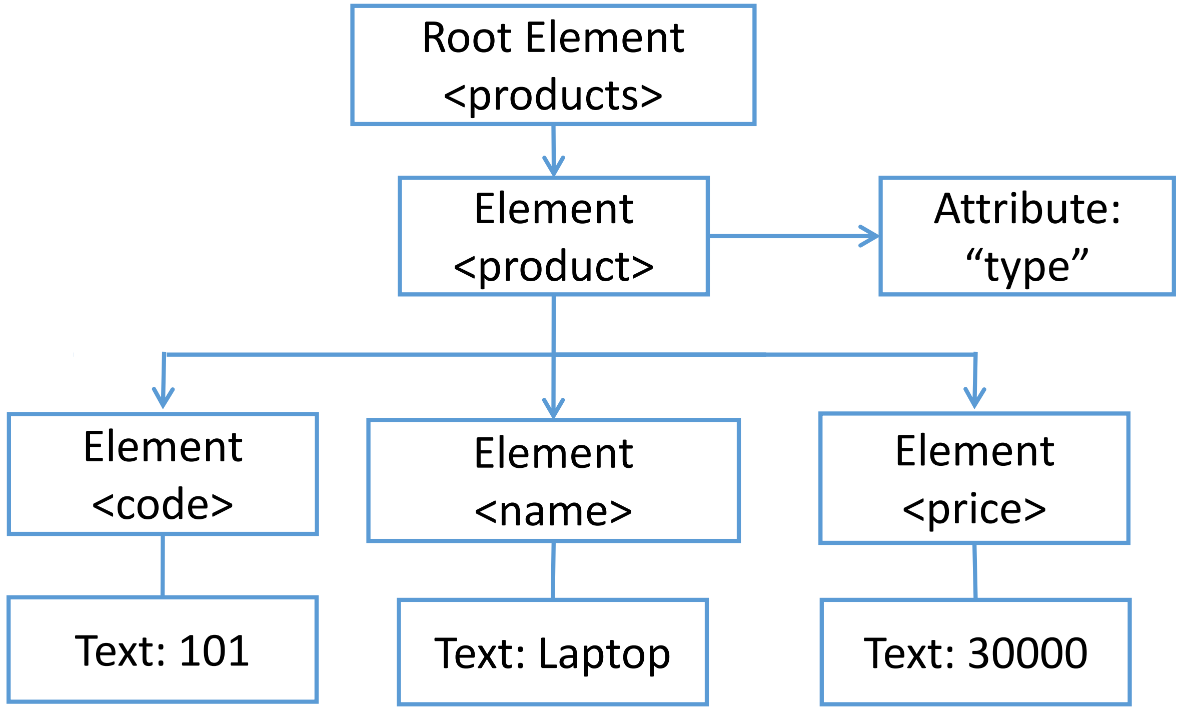
All elements can have sub elements (child elements)

Child elements at the same level are called siblings

All elements can have text and attributes

XML STRUCTURE

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE products [
<!ENTITY test "">
]>
<products type=“electronic”>
    <product>
        <code>101</code>
        <name>Laptop</name>
        <price> 30000</price>
    </product>
    <product>
       <code>102</code>
       <name>Tablet</name>
       <price>20000</price>
    </product>
</products>
```



Root Element is <products>. Also called document element

All <product> are child elements. Child elements are contained within parent <products>

<code>, <name> and <price> are three child elements of <product>

XML DECLARATION

- Describes general properties of the document that help XML processor to interpret the document

Syntax:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

XML DECLARATION

Three parameters:

Version:

Specifies the version of the XML standard used. Currently the version is 1.0

Encoding:

Defines the character encoding used in the document such as US-ASCII, iso-8859-1
Default is UTF-8

Standalone:

Informs the xml processor whether the document relies on the information from an external source. Default is no

XML STRUCTURE

Rules				
If XML declaration is present, it should be the first statement in the document	XML declaration must at least contain version parameter	Parameter names must be in lower case	Order of parameter is important: 1. Version 2. Encoding 3. Standalone	Parameter values should be enclosed either in single or double quotes

XML STRUCTURE

Example:

```
<?xml version="1.0"?>

<?xml version='1.0' encoding='US-ASCII' standalone='yes'?>

<?xml version = '1.0' encoding= 'iso-8859-1' standalone ="no"?>
```

XML ELEMENTS

- Building blocks of the document
- Divide the document into its constituent parts
- Provide logical structure to the data/content
- Begin with a start tag and end with a matching tag

  (tag is a markup construct beginning with '<' and ending with '>')

```
<author category=“poet”>William Shakespeare</author>
```

<author> : Start tag

</author> : End Tag

- Attribute defines a property of the element with a name-value pair

XML ELEMENTS

Contains:

- Text

```
<author>William Shakespeare</author>
```

- Other element

```
<author>
      <firstname>William</firstname>
      <lastname> Shakespeare <lastname>
</author>
```

- Attributes

```
<author category="poet"  lang="English">
       William Shakespeare
</author>
```

XML STRUCTURE

Empty XML Elements

- An element with no content is said to be empty
- Syntax:

```
<elementName></elementName>
                  OR
            <elementName />
```

- Empty elements can have attributes

```
<nextPage layout="portrait" />
```

XML ELEMENTS

XML Attributes

- Provide additional information about the elements
- Provide information that is not a part of the data

```
<fileName type="xls" version="2003">assessment.xls</fileName>
```

Here ‘type’ and ‘version’ are attributes of element fileName

- Typically metadata should be stored as attributes
- Attributes should be avoided for storing the actual data (content) of the element

XML ELEMENTS

Naming rules for element names:

- Case-sensitive
- Must start with a letter or underscore
- Cannot start with the letters xml
- Can contain letters, digits, hyphens, underscores, and periods
- Cannot contain spaces

`<address-residential>`
`<_street>`
`<city.name>` ✓
	`<address*residential>`
	`  <street  name>`
	`  <3city>` ✗

CONCEPT

XML Syntax

XML SYNTAX

- XML document must have one root element
  - All elements in the document should be contained in one parent element called as root element or document element

`<root>`
`  <child>`
`    <subchild>.....</subchild>`
`  </child>`
`</root>` ✓
	`<child>`
	`    <subchild>.....</subchild>`
	` </child>`
	` <child>  .....  </child>` ✗

- All elements must have a closing tag

`<address>`
`     <street>II Main</street>`
`     <city>Bangalore</city>`
`</address>` ✓
	`<address>`
	`     <street>II Main`
	`     <city>Bangalore`
	`</address>` ✗

XML SYNTAX

- XML tags are case sensitive

```
<city>Bangalore</city>
```
✓

```
<city>Bangalore</City>
```
✗

- XML elements must be properly nested

```
<address>
      <city>Bangalore</city>
</address>
```
✓

```
<address>
      <city>Bangalore
</address>
        </city>
```
✗

- XML attribute values must be quoted

```
<address  fromDate=“20-06-2015” >
      <city>Bangalore</city>
</address>
```
✓

```
<address  fromDate=20-06-2015 >
      <city>Bangalore </city>
</address>
```
✗

XML SYNTAX

```
<condition> amount < 500 </amount>
```
✗

Some characters like '<' cannot be used in XML text as it conflicts with markup delimiter

- Predefined character entities
  - Entities are place holder for content
  - Declared in the prolog/DTD and can be used many times in the document
  - Predefined entities are put in XML using entity references

Entity name	Entity value
amp	&
apos	'
gt	>
lt	<
quot	"

```
<condition> amount &lt; 500 </amount>
```

- Numbered character entity

```
<condition> amount &#060; 500 </amount>
```

XML SYNTAX

- XML comment
  - Used to identify purpose of document and sections of document
  - Not interpreted by XML processor

```
<!--     comment goes here   -->
```

- Whitespace
  - In HTML, white spaces are not preserved in text
  - XML does not truncate white spaces in content text

```
<city> Bangalore  South </city>
```

CDATA SECTIONS

- XML document containing many predefined entities like &lt; &amp;
  - Requires a lot of typing
  - Hard to read in the markup
- In such cases, CDATA sections are used
- CDATA means character data, blocks of text that are not parsed by the parser

```
< ! [ CDATA [ text inside is  not parsed    ] ] >
```

```
<condition1> No discount if < ! [ CDATA [ amount > 0 & amount < 500] ] >
</condition1>
```

CONCEPT

Well Formed XML and Valid XML

WELL FORMED XML

XML document that conform to the syntax rules specified by XML 1.0 specification are called "Well Formed" XML documents

To summarize, the rules are:

- XML document must have one root element and begin with XML declaration
- All elements must have starting tag accompanied with closing tag
- Since XML tags are case sensitive, both start and end tags should use same case
- XML elements must be properly nested
- Element attribute values must use quotes
- Predefined entities should be used for forbidden characters
- Element name should follow the rules specified

WELL FORMED XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<employees>
   <employee type="permanent" >
       <empNumber>1000</empNumber>
       <name>
          <first>Mahendra</first>
          <last>Dhoni</last>
       </name>
       <department>corporate</department>
       <email>mdhoni@xml.com</email>
   </employee>
   <employee  type="consultant" >
       <empNumber>9000</empNumber>
       <name>
          <first>yousuf</first>
          <last>pathan</last>
          </name>
          <department>Accounts</department>
          <email>ypathan@xml.com</email>
       </employee>
   </employees>
```

VALID XML DOCUMENT

- ➢ Well formed document just adheres to the syntax of XML specification
- ➢ A valid XML document is one which should:
  - Be well formed
  - Adhere to the rules specified in a template/schema
- ➢ Template/schema defines the rules for the elements and attributes
- ➢ Template/schema contains rules pertaining to
  - Standardized element name and number of elements
  - Hierarchical structure
  - Attributes
  - Entities

VALID XML DOCUMENT

- ➢ Two types of template/schema can be used with XML:
  - DTD - Document type definition
  - XML Schema – XML based alternative to DTD
- ➢ Need of template/schema
  - Business organizations exchange data using XML for purposes like order processing, banking etc.
  - Organizations can agree on a standard for interchanging data
  - Sender can describe the data in a way that the receiver will understand
  - Receiver can verify the validity before processing the data

QUIZ QUESTION

1. What is markup?
   - ❑ It describes logical structure of document
   - ❑ It enhances meaning of text.
   - ❑ It can be used for processing.
   - ❑ All are correct.

QUIZ QUESTION

1. What is markup?

- ☐ It describes logical structure of document
- ☐ It enhances meaning of text.
- ☐ It can be used for processing.
- [✓ All are correct.]

QUIZ QUESTION

2. What is correct syntax to describe XML Prolog?

- ❑ <?xml version=1.0 />
- ❑ <?xml version="1.0" ?>
- ❑ <?xml ver=1.0 />
- ❑ <xml version="1.0" />

QUIZ QUESTION

2. What is correct syntax to describe XML Prolog?

❑ <?xml version=1.0 />

[✓ <?xml version=“1.0” ?>]

❑ <?xml ver=1.0 />

❑ <xml version=“1.0” />

QUIZ QUESTION

3. What are the components of an XML document?

- ❑ Header, Root element
- ❑ Prolog, Header, Body
- ❑ Prolog, Root element
- ❑ Header, Body

QUIZ QUESTION

3. What are the components of an XML document?

- ❑ Header, Root element
- ❑ Prolog, Header, Body
- [✓ Prolog, Root element]
- ❑ Header, Body

QUIZ QUESTION

4. When is XML document said to be valid?

- ❑ If XML document has proper nested structure
- ❑ If it is well-formed
- ❑ If it follows rules defined in DTD or XML Schema
- ❑ If it contains prolog and root element

QUIZ QUESTION

4. When is XML document said to be valid?

- ☐ If XML document has proper nested structure
- ☐ If it is well-formed
- ☑ If it follows rules defined in DTD or XML Schema
- ☐ If it contains prolog and root element

SUMMARY

Introduction to XML

SUMMARY

- Markup Language is a set of symbols placed in the text of a document to provide clear boundary and label related to pieces of text.
- XML is a text based markup language derived from SGML, designed to describe and organize data.
- XML document must have one root element and all elements must have a closing tag. XML tags are case sensitive, elements must be properly nested and attribute values must be quoted.
- XML document that conform to the syntax rules specified by XML 1.0 specification are called "Well Formed" XML documents. A valid XML document is one which should be well formed and should adhere to the rules specified in a template/schema.

OBJECTIVES

DTD

LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

- Discuss the Basics of DTD
- Identify the DTD Elements
- Identify the DTD Attributes

CONCEPT

Basics of DTD

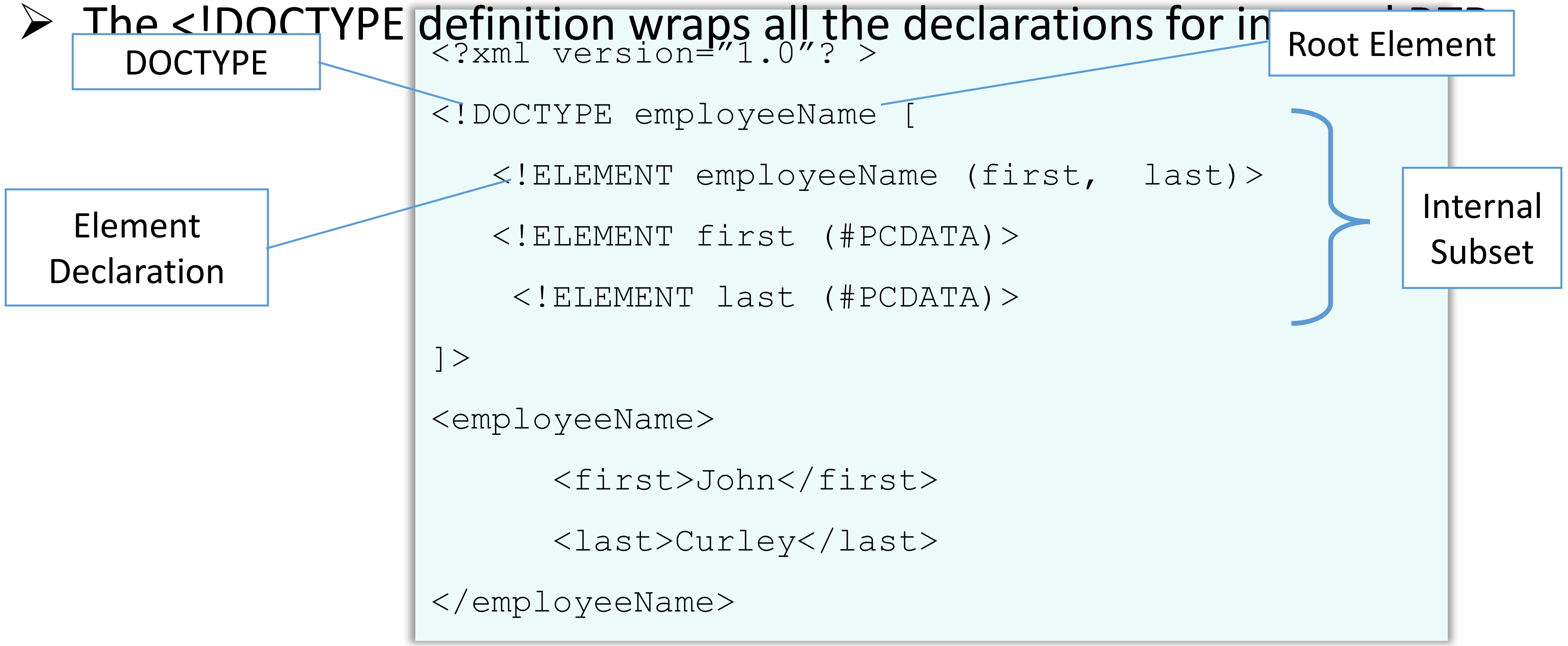
OVERVIEW

- Document Type Definition (DTD)
  - Defines the legal building blocks of an XML document
  - Defines the document structure with a list of legal elements and attributes
  - Can be declared inside an XML document or as a separate document externally
- A DTD contains declarations for:

Element	Attribute	Entity	Notations

INTERNAL DTD DECLARATION

- In Internal DTD, all declarations are done inside the XML document within <!DOCTYPE definition
- The <!DOCTYPE definition wraps all the declarations for in



```
<?xml version="1.0"? >
<!DOCTYPE employeeName [
   <!ELEMENT employeeName (first,  last)>
   <!ELEMENT first (#PCDATA)>
    <!ELEMENT last (#PCDATA)>
]>
<employeeName>
      <first>John</first>
      <last>Curley</last>
</employeeName>
```

INTERNAL DTD DECLARATION

DOCTYPE

- Signals start of DTD and must appear at the start of XML
- Can be preceded only by XML declaration
- Informs the parser that a DTD is associated with this XML

Root element

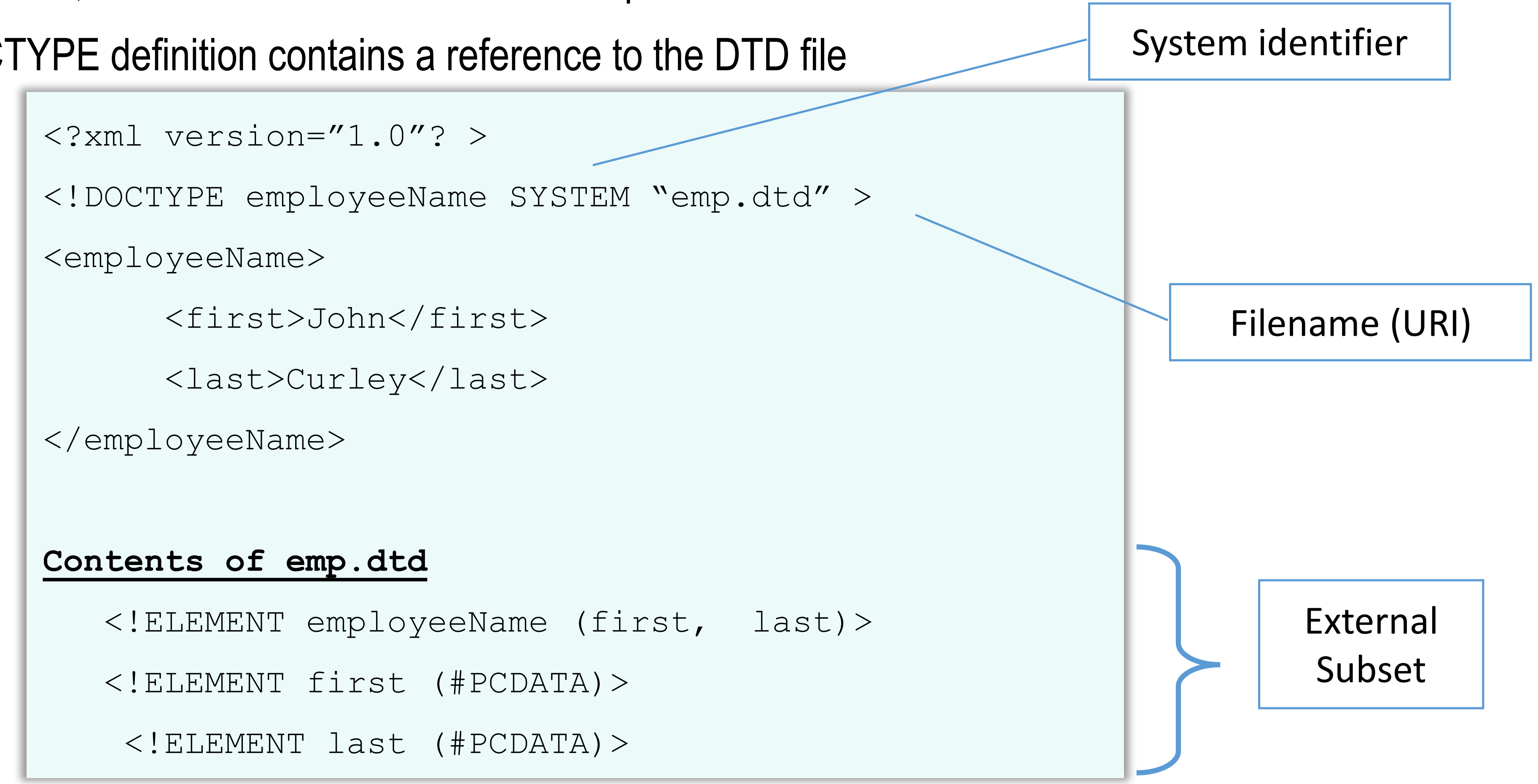
- Followed by <!DOCTYPE declaration (<!DOCTYPE employeeName)
- Defines the root element of the XML document

Declarations (Internal subset)

- All the declarations are done inside the [ ]
- <!ELEMENT employeeName defines that ‘employeeName’ must contain only two elements first and last
- <!ELEMENT first defines the ‘first’ element, should be of type #PCDATA

EXTERNAL DTD DECLARATION

- In external DTD, all declarations are done in a separate file
- The <!DOCTYPE definition contains a reference to the DTD file



```
<?xml version="1.0"? >
<!DOCTYPE employeeName SYSTEM "emp.dtd" >
<employeeName>
      <first>John</first>
      <last>Curley</last>
</employeeName>
```

Contents of emp.dtd

```
   <!ELEMENT employeeName (first,  last)>
   <!ELEMENT first (#PCDATA)>
    <!ELEMENT last (#PCDATA)>
```

EXTERNAL DTD DECLARATION

SYSTEM identifier

- Enables to specify the location of an external file containing DTD Declarations
- Comprises of two parts:
  - Keyword SYSTEM
  - URI reference
- URI can refer to:
  - file on local hard drive
  - file on intranet/network
  - file on the Internet

```
<!DOCTYPE root_element SYSTEM “file:///c:/emp.dtd”>
<!DOCTYPE root_element SYSTEM “http://xml.com/emp/emp.dtd”>
<!DOCTYPE root_element SYSTEM “emp.dtd”>
```

CONCEPT

DTD Elements

ELEMENT DECLARATION

- Character data
- Text will NOT be parsed by a parser
- Tags inside the text will NOT be treated as markup and entities will not be expanded

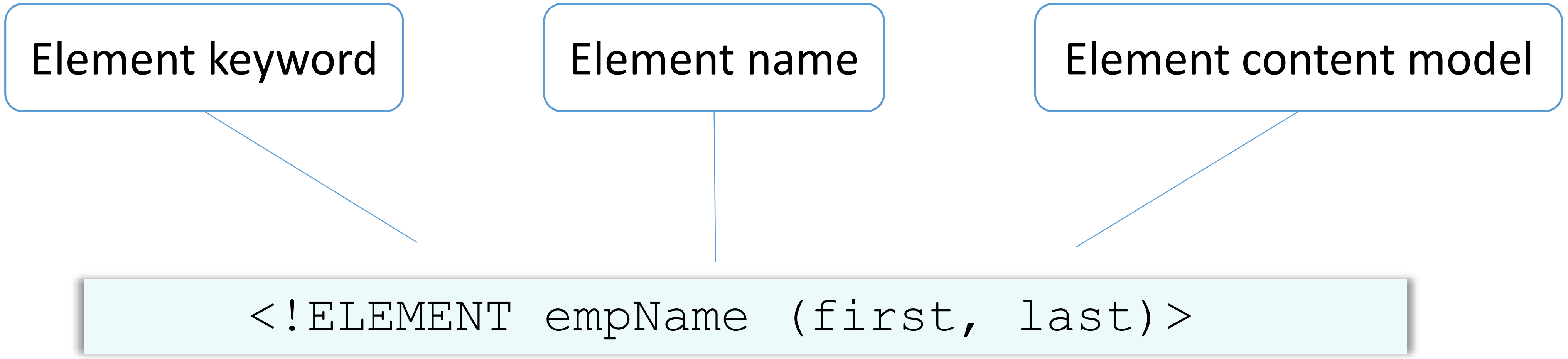
CDATA

- Parsed character data
- Text will be parsed by a parser
- Tags inside the text will be treated as markup and entities will be expanded
- Should not contain any &, <, or > characters and will be represented by predefined

PCDATA

ELEMENT DECLARATION

- Used to define valid elements allowed in XML document
- Consists of 3 parts:



```
<!ELEMENT empName (first, last)>
```

ELEMENT DECLARATION

- Element content model
  - Defines the allowable content within the element
  - Four content models are:

Element

Mixed

Empty

Any

ELEMENT CONTENT MODEL

- Sequences

```
<!ELEMENT address (street, city, state, pincode) >
<address> must have exactly 4 child elements <street>, <city>,
<state> & <pincode> in the same order
```

- Choices

```
<!ELEMENT phone ( landline | mobile ) >
<phone> element can have only 1 child element. Either <landline> or
<mobile>
```

- Combining sequences and choices

```
<!ELEMENT contact ( address, (landline | mobile ),    email))
<contact> must have 3 child elements.<address>, either <landline> or
<mobile>, and <email>
```

ELEMENT CONTENT MODEL

- Mixed content
  - Text can appear by itself or it can be interspersed between elements

```
<!ELEMENT first (#PCDATA)>
      <!ELEMENT empName (#PCDATA, first, #PCDATA,
last)>
```

```
Example:

<empName>Employee first name : <first>John</first>
last name : <last>conner</last></empName>
```

ELEMENT CONTENT MODEL

- Empty content

```
<!ELEMENT  hr  EMPTY>
```

Example:

```
<hr/>
```

- Any content

```
<!ELEMENT description ANY>
```

Text (PCDATA) and/or any elements declared within the DTD can be used within the content

ELEMENT MULTIPLICITY

➢ Element's multiplicity defines how many times it will appear within a content model

Markup	Definition	Example
A,B	Both A and B occur, in that order	`<!ELEMENT contact(name,address,landline)`
A?	A occurs zero or one time	`<!ELEMENT contact(name,address,landline?)`
A*	A occurs zero or more times	`<!ELEMENT contacts (contact*)>`
A+	A occurs one or more times	`<!ELEMENT contact (name,address,email+)>`

```
<!ELEMENT name (first+, middle?, last)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT middle (#PCDATA)>
<!ELEMENT last (#PCDATA)>
```

CONCEPT

DTD Attributes

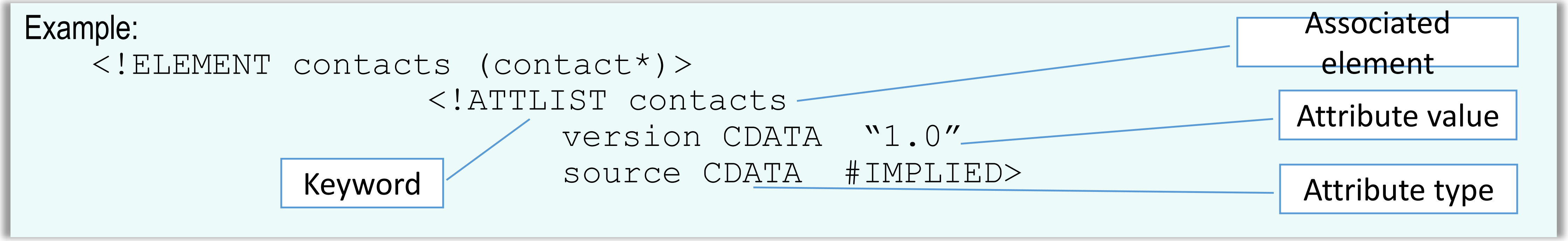
ATTRIBUTE DECLARATION

- Declare a list of allowable attributes for each element
- <!ATTLIST> declaration is used to define attributes for an element
- Single <!ATTLIST> declaration declares all attributes for a given element

Syntax:

```
      <!ATTLIST element_name
                     attribute_name   TYPE   DEFAULT_VALUE
                     attribute_name   TYPE   DEFAULT_VALUE >
```

Example:



```
   <!ELEMENT contacts (contact*)>
                  <!ATTLIST contacts
                      version CDATA  "1.0"
                      source CDATA  #IMPLIED>
```

ATTRIBUTE TYPES AND VALUES

Type	Description
CDATA	Indicates attribute value is Character Data
ENTITY	Attribute value is a reference to an entity
ID	Attribute value uniquely identifies the element
Enumerated	Possible values for the attribute
NMTOKEN	Attribute value must be a valid XML name

Value	Description
Default value	Attribute value enclosed in " "
#IMPLIED	Optional attribute
#REQUIRED	Required attribute
#FIXED value	Attribute value must be the value provided

```
<!ATTLIST contacts version CDATA #FIXED "1.0">

<!ATTLIST phone type (Home | Work | mobile) "Home">

<!ATTLIST contact category CDATA #REQUIRED>
```

ENTITY

- Predefined information which can be reused in the XML Document
- Defined in DTD with a name and associated value

```
<!ENTITY entity-name "entity-value">

      <!ENTITY year "2015">
```

- To Refer entity in XML,

```
   &entity-name

              <copyright> &year
<copyright>
```

DTD EXAMPLE

```
<!ELEMENT contacts (contact+) >
<!ELEMENT contact (name, location, phone,
email?) >
<!ELEMENT location ((city,state)| country) >
<!ELEMENT phone ( home|mobile)* >
<!ELEMENT name (#PCDATA) >
<!ELEMENT city (#PCDATA) >
<!ELEMENT state (#PCDATA) >
<!ELEMENT country (#PCDATA) >
<!ELEMENT home (#PCDATA) >
<!ELEMENT mobile (#PCDATA) >
<!ELEMENT email (#PCDATA) >
<!ATTLIST contact category (family | official
| relative) #REQUIRED >
```

```
<?xml version="1.0"? encoding="UTF-8">
<!DOCTYPE contacts SYSTEM "contacts.dtd" [
        <!ENTITY domain "xml.com" >
]>

<contacts>
        <contact category="official" >
               <name>Jimmy</name>
                <location>
               <city>Hyderabad</city>
               <state>Telangana</state>
                 </location>
                 <phone>
               <mobile>9845022346</mobile>
               <mobile>9845022347</mobile>
                  </phone>

<email>jim@&domain;</email>
        </contact>
</contacts>
```

QUIZ QUESTION

1. Which syntax is appropriate to associate external DTD file with the XML document?

   - ❑ `<!DOCTYPE root-element SYSTEM “filename”>`
   - ❑ `<!DOCTYPE SYSTEM “filename”>`
   - ❑ `<!DOCTYPE “filename”>`
   - ❑ `<!DOCTYPE root-element EXTERNAL “filename”>`

QUIZ QUESTION

1. Which syntax is appropriate to associate external DTD file with the XML document?

- [✓ <!DOCTYPE root-element SYSTEM “filename”>]
- <!DOCTYPE SYSTEM “filename”>
- <!DOCTYPE “filename”>
- <!DOCTYPE root-element EXTERNAL “filename”>

QUIZ QUESTION

2. What is the use of DTD in XML development?

- ❑ To convert XML document from one format to other format
- ❑ To validate XML document
- ❑ To check well-formedness of XML document
- ❑ To create XML template

QUIZ QUESTION

2. What is the use of DTD in XML development?

- ❑ To convert XML document from one format to other format
- [✓] To validate XML document
- ❑ To check well-formedness of XML document
- ❑ To create XML template

SUMMARY

DTD

SUMMARY

- Document Type Definition (DTD) defines the legal building blocks of an XML document and the document structure with a list of legal elements and attributes.
- DTD elements consists of 3 parts, element keyword, element name and element content model.
- DTD Attributes are CDATA, ENTITY, ID, Enumerated and NMTOKEN.

OBJECTIVES

Namespaces and XML Schema

LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

- Identify the need of namespace
- Explain the benefits of XML Schema over DTD
- Use Simple and Complex Elements of XML Schema

CONCEPT

Namespaces

WHAT ARE NAMESPACES?

- Way of grouping identifiers around a behavior
- Avoid naming conflicts of identifiers
- Help to set a context for identifiers with the same name
  - Example: ‘Table’ seen from perspective of Database developer and a HTML developer are totally different things
  - Helps to set the context and avoid name confusions
- Programming languages also have namespace concept for grouping and avoiding program name conflicts
  - Namespace in C#
  - Packages in Java

XML NAMESPACES

- Need
  - XML is used heavily for exchange of data between organizations
  - Since XML element names are created by developers in different organizations, maintaining uniqueness is a challenge
  - Duplicate element names when XML fragments are merged could pose problems in processing of data
- XML Namespaces provide a method to avoid element and attribute name conflicts

```
<employee
xmlns="http://www.organization.com/application/name">

         child-elements

   </employee>
```

DECLARING NAMESPACES

- Syntax: **xmlns:prefix="URI"**
- Declared in the start tag of the element
- Each namespace defined must be associated with a distinct Uniform Resource Identifier (URI)
- Associated with a prefix when they are declared
- Scope for the element in which it is defined and all of its child elements

```
<employee  xmlns: em="http://www.organization.com/application/name" >
      …….
</employee>
```

USING NAMESPACES

Prefix helps to define which child elements belong to a particular

```
<?xml version="1.0"?>
<lib:library    xmlns:lib="http://www.library.com/lb/library
            xmlns:hr="http://www.author.com/at/person" >
     <lib:book available="true">
           <lib:isbn>198745672</lib:isbn>
           <lib:title>XML Simplified</lib:title>
           <hr:author>
                 <hr:name>Martin Reacher</hr:name>
                 <hr:type>Technical</hr:category
           </hr:author>
     </lib:book>
</lib:library>
```

CONCEPT

Features of XML Schema

XML SCHEMAS

DTD Limitations

- Different syntax compared to XML syntax
- Poor support for built in data types and no support for derived datatypes
- No constraint on the range of data allowed in an element
- Large DTDs are hard to read and maintain
- Single DTD per XML document
- DTD doesn’t support inheritance, thus no reusability
- Internal DTD subset can override the external DTD subset
- Poor support for XML namespaces. For a namespace to be used, the entire namespace must be defined within the DTD

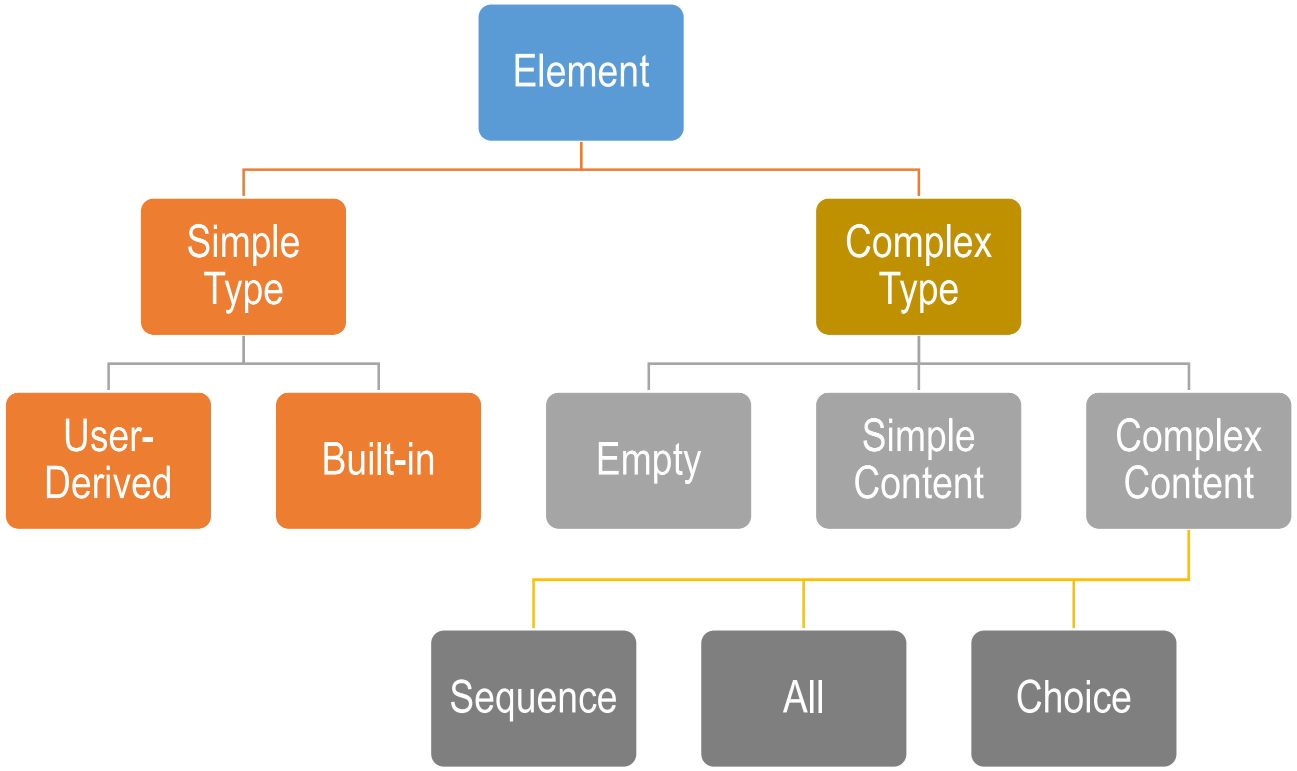
XML SCHEMAS

- Schema is any type of model document that defines the structure of something
- XML schemas are XML based alternative to DTD
- Describe the structure of XML documents in more detail than DTD
- Define the rules for data content and semantics
- Allow more precise validation
- XSD (XML Schema Definition) is the default standard recommended by W3C for XML Schema
- Other XML standards are Schematron, Relax NG

XML SCHEMA BENEFITS

- ➢ Created using basic XML syntax, whereas DTDs utilize a separate syntax
- ➢ Enables to validate text element content based on built-in and user-defined data types
  - describe allowable content
  - validate the correctness of data
  - define data patterns
  - define data restrictions
- ➢ Enables to easily create complex and reusable content models
- ➢ Enables the modeling of programming concepts such as object inheritance
- ➢ Fully supports the Namespace Recommendation

SCHEMA ELEMENTS



OVERVIEW

- Elements can be of simple type or complex type
- Simple type elements contain only text
- Built-in types are simple types (e.g- xs:string)
- User Derived simple types are created by restricting data in simple types (e.g- email pattern)
- Complex type elements can contain child elements and attributes as well as text
- Complex types can be empty having only attributes
- Complex type elements can have simple content like text and have attributes
- By default, complex type elements have complex content (i.e. they have child elements)

CONCEPT

Simple and Complex Elements

SIMPLE SCHEMA EXAMPLE

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
  <xs:element name="empName">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="first" type="xs:string"/>
        <xs:element name="middle" type="xs:string"/>
        <xs:element name="last" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="title" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<?xml version="1.0"?>
<empName xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:noNamespaceSchemaLocation="empName.xsd"
       title="Mr.">
         <first>Rajesh</first>
         <middle>Kumar</middle>
         <last>Dubey</last>
</empName>
```

- ➢ XML schema must follow all the rules of a well formed XML
- ➢ Root element of XML schemas is xs:schema
- ➢ xmlns:xs =http://www.w3.org/2001/XMLSchema indicates that document should follow rules of XML Schema
- ➢ xs:element is used to define an element
- ➢ empName is a complex type element, which contains a sequence of elements: First, middle and Last
- ➢ First is a simple type element with datatype string

SIMPLE TYPE ELEMENT

- A simple element can be defined within an XSD as:

```
<xs:element name="element-name" type="element-
datatype"/>

<xs:element name="first" type="xs:string"/>
```

Each element definition within the XSD must have a 'name' property

- This value of the name property becomes the tag name in XML document
- The type describes the type of data the element text can hold

SIMPLE TYPE ELEMENT

Some of the predefined built-in datatypes are:

- xs:string
- xs:date
- xs:boolean
- xs:integer
- xs:float
- xs:double
- xs:ID
- xs:IDREF
- and many more

For full List, refer http://www.w3.org/TR/xmlschema-2/#built-in-datatypes

SIMPLE TYPE ELEMENT

XSD	XML
`<xs:element name=“empDob" type="xs:date"/>`	`<empDob>2000-01-30</empDob>`
`<xs:element name="empAddress" type="xs:string"/>`	`<empAddress>Bangalore</empAddress>`
`<xs:element name="OrderID" type="xs:int"/>`	`<OrderID>1001</OrderID>`
`<xs:element name=“message" type="xs:string"/>`	`<message> Hello world </message>`
`<xs:element name=“valid" type="xs:boolean"/>`	`<valid>  true  </valid>`

DEFAULT

```
<xs:element name="customerName" type="xs:string" default="unknown"/>
```

FIXED

```
<xs:element name="customerLocation" type="xs:string" fixed="India"/>
```

CARDINALITY

```
<xs:element name="skills" type="xs:string" minOccurs ="0"  maxOccurs="unbounded"/>

<xs:element name="childName" type="xs:string" minOccurs ="1"  maxOccurs="3"/>
```

COMPLEX TYPE ELEMENT

- Has child elements, attributes or combination of two
- Can be empty, can contain simple content such as a string or can contain complex content such as a sequence of elements
- Defined using element xs:complexType

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

QUIZ QUESTION

1. Namespace can be used:
   - ❑ To avoid naming conflict
   - ❑ To avoid ambiguity between elements
   - ❑ With DTD
   - ❑ With XML Schema

QUIZ QUESTION

1. Namespace can be used:
   - [✓] To avoid naming conflict
   - [✓] To avoid ambiguity between elements
   - [ ] With DTD
   - [✓] With XML Schema

QUIZ QUESTION

2. Which type of XML element can be empty?

- ❑ Simple
- ❑ Complex
- ❑ Both
- ❑ none

QUIZ QUESTION

2. Which type of XML element can be empty?

- ☐ Simple
- [✓ Complex]
- ☐ Both
- ☐ none

QUIZ QUESTION

3. Which statement(s) is/are true?

- ❑ Simple elements can have attributes
- ❑ Simple elements can have child elements
- ❑ Each element definition within the XSD must have a 'name' property.
- ❑ It is not mandatory to have <xs:schema> as a root element in XSD
- ❑ XML Schema can put more restrictions than DTD

QUIZ QUESTION

3. Which statement(s) is/are true?

- [ ] Simple elements can have attributes
- [ ] Simple elements can have child elements
- [✓] Each element definition within the XSD must have a 'name' property.
- [ ] It is not mandatory to have <xs:schema> as a root element in XSD
- [✓] XML Schema can put more restrictions than DTD

SUMMARY

Namespaces and XML Schema

SUMMARY

- Since XML element names are created by developers in different organizations, maintaining uniqueness is a challenge. XML Namespaces provide a method to avoid element and attribute name conflicts.
- XML Schema is created using basic XML syntax, whereas DTDs utilize a separate syntax.
- A simple element can be defined within an XSD where each element must have a 'name' property. Complex element type has child elements, attributes or combination of two which is defined using element xs:complexType.