

Modern Markup for Data Interchange

Modern Markup for Data Interchange

Trainer's Guide

© 2016 Aptech Limited

All rights reserved.

No part of this book may be reproduced or copied in any form or by any means – graphic, electronic or mechanical, including photocopying, recording, taping, or storing in information retrieval system or sent or transferred without the prior written permission of copyright owner Aptech Limited.

All trademarks acknowledged.

APTECH LIMITED

Contact E-mail: ov-support@onlinevarsity.com

First Edition - 2016



Unleash your potential



“

A little learning is a dangerous thing,
but a lot of ignorance is just as bad.

”

For Aptech Center Only

Preface

The book 'Modern Markup for Data Interchange' Trainer's Guide aims to teach the fundamentals of XML and JSON. The faculty/trainer should teach the concepts in the theory class using the slides. This Trainer's Guide will provide guidance on the flow of the module and also provide tips and additional examples wherever necessary. The trainer can ask questions to make the session interactive and also to test the understanding of the students.

This book is the result of a concentrated effort of the Design Team, which is continuously striving to bring you the best and the latest in Information Technology. The process of design has been a part of the ISO 9001 Certification for Aptech-IT Division, Education Support Services. As part of Aptech's quality drive, this team does intensive research and curriculum enrichment to keep it in line with industry trends.

We will be glad to receive your suggestions.

Design Team

“ Practice is the best of
all instructors.

For Aptech Center Use Only

Table of Contents

Sessions

1. Introduction to XML
2. Namespaces
3. Document Type Definitions
4. XML Schema
5. Style Sheets
6. XSL and XSLT
7. More on XSLT
8. Introduction to JSON
9. Work with JSON Data
10. JSON in Real World

**“ The future depends on what
we do in the present.**

For Aptech Center User Only

Session 1 - Introduction to XML

1.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth.

1.1.1 Objectives

By the end of this session, the learners will be able to:

- Introduction to XML
- Exploring XML
- Working with XML
- XML Syntax

1.1.2 Teaching Skills

To teach this session, you should be well-versed with XML and drawbacks of earlier markup languages that led to the development of XML. You should also aware yourself with the structure and lifecycle of the XML document, the XML syntax, and the various parts of the XML document.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

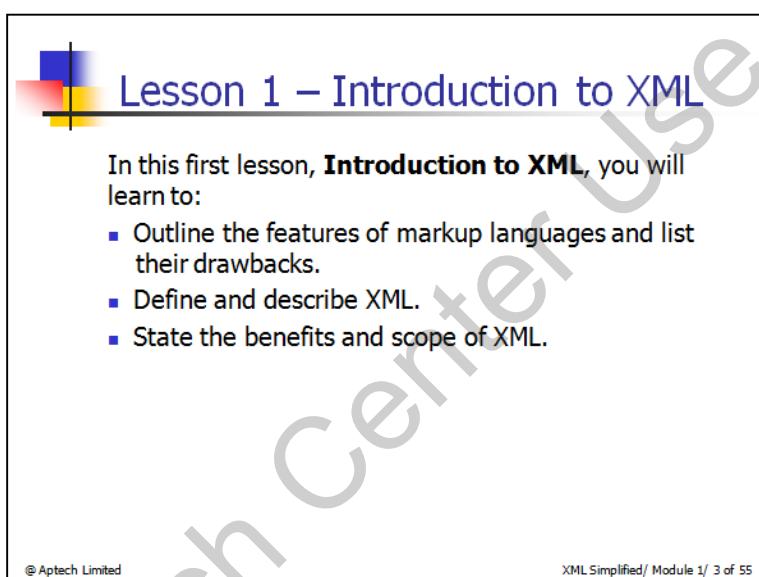
Then give the students the overview of the current session in the form of Session Overview. Show the students slide 2 of the presentation.

Tell the students that this session introduces them to XML. They will learn about the drawbacks of earlier markup language that led to the development of XML and the structure and lifecycle of the XML document. They will also learn about the XML syntax and the various parts of the XML document.

1.2 In-Class Explanations

Slide 3

Let us understand the objectives of lesson 1.



The slide has a decorative graphic in the top-left corner consisting of overlapping colored squares (blue, red, yellow) and a horizontal bar below it. The title "Lesson 1 – Introduction to XML" is centered at the top in a blue font. Below the title, the text reads: "In this first lesson, **Introduction to XML**, you will learn to:" followed by a bulleted list of three items. At the bottom left is the copyright notice "© Aptech Limited" and at the bottom right is the page information "XML Simplified/ Module 1/ 3 of 55".

Lesson 1 – Introduction to XML

In this first lesson, **Introduction to XML**, you will learn to:

- Outline the features of markup languages and list their drawbacks.
- Define and describe XML.
- State the benefits and scope of XML.

© Aptech Limited XML Simplified/ Module 1/ 3 of 55

Using slide 3, describe the topics to be covered in the lesson.

Slides 4 to 6

Let us understand the features of markup languages.

Features of Markup Languages 1-3

- Generalized Markup Language (GML) helps the documents to be edited, formatted, and searched by different programs using its content-based tags.
- Standard Generalized Markup Language (SGML) is a coding scheme for developing specialized markup languages.
- Hyper Text Markup Language (HTML) is used for sharing information by using hyperlinked text documents.

© Aptech Limited XML Simplified/ Module 1/ 4 of 55

Features of Markup Languages 2-3

Technology	Year
SGML	1980
HTML	1989

© Aptech Limited XML Simplified/ Module 1/ 5 of 55

Features of Markup Languages 3-3

Features

- GML describes the document in terms of its format, structure and other properties.
- SGML ensures that system can represent the data in its own way.
- HTML used ASCII text, which allows the user to use any text editor.

Drawbacks

- GML and SGML were not suited for data interchange over the web.
- HTML instructions is used to display content rather than content they encompass.

© Aptech Limited XML Simplified/ Module 1/ 6 of 55

Using slides 4 to 6, explain the features of markup languages.

Explain them that markup languages are used to define the meaning and organize the structure of a text document. Markup languages help the documents by giving them a new

look and formatting the text. Markup languages are expressed with the help of tags. They instruct the software that interprets the text to carry out some action on it.

Earlier markup languages were used in publishing practices which involved handwritten annotations in the form of symbols. Some early examples of computer markup languages available outside the publishing industry can be found in typesetting tools on UNIX systems such as troff and TEX.

In these systems, formatting commands were inserted into the document text so that typesetting software could format the text according to the editor's specifications.

Tips:

Markup was also commonly applied by editors, proofreaders, publishers, graphic designers, and indeed by document authors.

Explain them that based on the action taken on the text represented by markup languages; they are categorized into following categories:

- **Presentational Markup** – Focuses on the structure of the document. Examples include traditional word-processing systems that produce the What You See is What You Get (WYSIWYG) effect within the document text.
- **Procedural Markup** – Similar to presentation markup. In addition, they help the users to arrange the text. Examples of procedural markup systems include the creation of macros that can be defined and invoked by name.
- **Descriptive Markup** – Determines the content of the document. This means they are used to label the markup text, rather than providing instruction on how text should be processed. Example of such markup languages includes use of certain tags such as <cite> used in HTML language to label the text.

Creation of SGML Markup Language

In 1980, American National Standards Institute (ANSI) committee created Standard Generalized Markup Language (SGML), an all-encompassing coding scheme and flexible toolkit for developing specialized markup languages.

SGML is the successor to Generalized Markup Language (GML). In 1986, International Organization for Standardization (ISO) acquired it as a standard. SGML is a meta language as other languages are created from it. SGML has a syntax to include markup in documents as well as the tags describing what markup is allowed to do in the document.

SGML application consists of SGML declaration and SGML Document Type Definition (DTD). SGML found wide acceptance and use in fields with very large-scale documentation requirements.

Creation of HTML Markup Language

In 1989, HyperText Markup Language (HTML), a technology for sharing information by using hyperlinked text documents was developed. HTML was created from SGML. In the early years, it was extensively accessed by scientists and technicians. HTML was originally created to mark up technical papers, so that they could be transferred across different platforms. However, with time, it began to be used for marking up non-technical documents too. As the use of the Internet became popular, browser manufacturers started developing different tags to display documents with more creativity.

HTML became the main markup language for creating Web pages and other information that can be displayed in a Web browser, and is quite likely the most widely used markup language in the world today.

Explain the evolution of markup languages displayed on slide 5 and then explain the features and drawbacks of earlier Markup languages as described on slide 6.

Slides 7 and 8

Let us understand the evolution of XML.



Evolution of XML 1-2

- XML is a W3C recommendation.
- It is a set of rules for defining semantic tags that break a document into parts.
- XML was developed over HTML.

© Aptech Limited

XML Simplified/ Module 1/ 7 of 55

Evolution of XML 2-2

HTML	XML
HTML was designed to display data.	XML was designed to carry data.
HTML displays data and focuses on how data looks.	XML describes data and focuses on what data is.
HTML displays information.	XML describes information.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <Watch>
  <name>Titan</name>
  <price>$50</price>
  <description>A brown diamond studded watch</description>
</Watch>
```

© Aptech Limited XML Simplified/ Module 1/ 8 of 55

Using slides 7 and 8, explain the evolution of XML.

eXtensible Markup Language (XML) is a meta markup language developed by the World Wide Web Consortium. The main purpose of creating XML language was its use in exchanging the information between the applications running in different business environments. XML is a set of rules for defining semantic tags that break a document into parts and identify the different parts of the document.

Explain them that the basic differences between HTML and XML markup languages as listed in the table on slide 8. Also, show the basic XML document.

Slide 9

Let us understand the features of XML.

Features of XML

- XML stands for Extensible Markup Language
- XML is a markup language much like HTML
- XML was designed to describe data
- XML tags are not predefined
- XML uses a Document Type Definition (DTD) or an XML Schema to describe data

© Aptech Limited XML Simplified/ Module 1/ 9 of 55

Using slide 9, explain the features of XML.

Explain them though the writing style of XML is similar to HTML language, but it was created to structure, store, and transport information.

For example, book publishing software wants to describe the book details using XML. The generated XML document will be as follows:

```
<book>
<title> Learning XML </title>
<author> Jon Bosak </author>
<price> 200 </price>
</book>
```

This XML file is very much self-descriptive. The information wrapped in tags of XML document is received by software that can read, process, and use the information further in the application.

Tell them that XML is designed to describe data. For example, `<title>` tag describes the name of the book in the discussed XML. Similarly, `<author>` and `<price>` tag describe the name of the author and number of pages in the book.

Tell them that XML tags are not predefined like HTML, rather the tags are described by the author of the XML document. This is because XML tags describes the document, whereas HTML tags are used for presentation.

Tell them that XML uses a DTD or an XML Schema to describe the data.

In-Class Question:

After you finish the feature of XML, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



How can you use an XML document in application development?

Answer:

XML can be used for data storage and sharing the data by different applications.

Slides 10 and 11

Let us understand the representation of XML markup.

XML Markup 1-2

- XML markup defines the physical and logical layout of the document.
- XML markup divides a document into separate information containers called elements.
- A document consists of one outermost element, called `root` element that contains all the other elements, plus some optional administrative information at the top, known as XML declaration.

© Aptech Limited XML Simplified/ Module 1/ 10 of 55

XML Markup 2-2

Code Snippet

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <FlowerPlanet>
  <Name>Rose</Name>
  <Price>$1</Price>
  <Description>Red in color</Description>
  <Number>700</Number>
</FlowerPlanet>
```

where,
 <Name>, <Price>, <Description> and <Number> tags are
 elements
 <FlowerPlanet> and </FlowerPlanet> are the root elements

© Aptech Limited XML Simplified/ Module 1/ 11 of 55

Using slides 10 and 11, explain the representation of XML markup.

XML markup defines the physical and logical layout of the document. XML can be considered as an information container. It employs a tree-based structure to represent a document. The markup divides the information into a hierarchy of character data and container-like elements and its attributes. All elements can have text content and attributes similar to HTML.

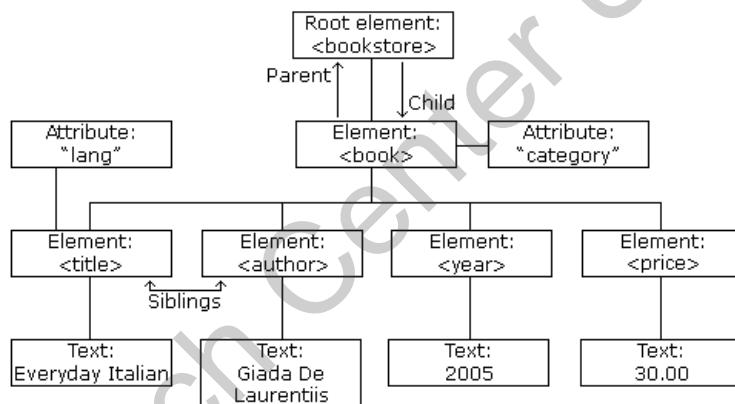
XML documents must contain a **root element**. This element is "the parent" of all other elements.

The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree. All elements can have sub elements (child elements) as shown in the XML template file:

```
<root>
  <child>
    <sub-child>.....</sub-child>
    <sub-child>.....</sub-child>
  </child>
</root>
```

The terms **parent**, **child**, and **sibling** are used to describe the relationships between elements. Parent elements have children. Children on the same level are called **siblings** (brothers or sisters).

Following figure shows the tree structure representation of XML containing book information:



The usage of XML can be observed in many real-life scenarios. It can be used in the fields of information sharing, single application usage, content delivery, re-use of data, separation of data and presentation, semantics, and so forth. News agencies are a common place where XML is used. News producers and news consumers often use a standard specification named XMLNews to produce, retrieve, and relay information across different systems in the world.

Then, explain the XML code provided on slide 11.

In-Class Question:

After you finish the representation of XML markup, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Consider that a bookstore application wants to store the details of books available in the stock. Now, what can be the root element for storing books information?

Answer:

To create a bookstore information file in XML, the root element can be named as:

```
<bookstore>
```

```
    <book>
```

```
        . . .
```

```
    </book>
```

```
    <book>
```

```
        . . .
```

```
    </book>
```

```
</bookstore>
```

XML can be used for data storage and sharing the data by different applications.

Slide 12

Let us understand the benefits of XML.

Benefits of XML

- Data independence - separates the content from its presentation
- Easier to parse - absence of formatting instructions makes it easy to parse
- Reducing Server Load - semantic and structural information enables it to be manipulated by any application, can now be performed by clients
- Easier to create – can easily create with the most primitive text processing tools
- Web Site Content – can be transformed into a number of other formats
- Remote Procedure Calls – protocol that allows objects on one computer to call objects on another computer
- E-Commerce - can be used as an exchange format

Using slide 12, explain the benefits of XML.

Slide 13

Let us understand the objectives of lesson 2.



Lesson 2 – Exploring XML

In this second lesson, **Exploring XML**, you will learn to:

- Describe the structure of an XML document.
- Explain the lifecycle of an XML document.
- State the functions of editors for XML and list the popularly used editors.
- State the functions of parsers for XML and list names of commonly used parsers.
- State the functions of browsers for XML and list the commonly used browsers.

© Aptech Limited XML Simplified/ Module 1/ 13 of 55

Using slide 13, describe the topics covered in lesson 2.

Slides 14 to 17

Let us understand the XML document structure.



XML Document Structure 1-4

- The two sections of an XML document are:
 - Document Prolog
 - Root Element
- An XML document consists of a set of unambiguously named "entities".
- Every document starts with a "root" or document entity. All other entities are optional.
- A single entity name can represent a large amount of text. The alias name is used each time some text is referenced and the processor expands the contents of the alias.

© Aptech Limited XML Simplified/ Module 1/ 14 of 55

XML Document Structure 2-4

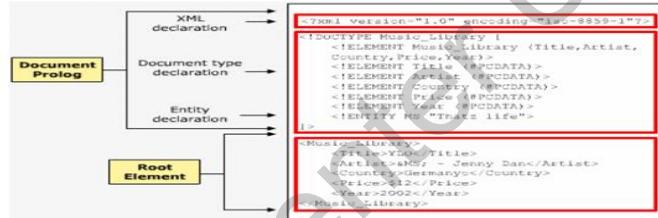
Document Prolog

- Document prolog contains metadata and consists of two parts - XML Declaration and Document Type Declaration.
- XML Declaration specifies the version of XML being used.
- Document Type Declaration defines entities' or attributes' values and checks grammar and vocabulary of markup.

Root Element

- Also called a document element.
- It must contain all the other elements and content in the document.
- An XML element has a start tag and end tag.

XML Document Structure 3-4



XML Document Structure 4-4

Code Snippet

```

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE Music_Library [
  <!ELEMENT Music_Library (Title,Artist,Country,Price,Year)>
  <!ELEMENT Title (#PCDATA)>
  <!ELEMENT Artist (#PCDATA)>
  <!ELEMENT Country (#PCDATA)>
  <!ELEMENT Price (#PCDATA)>
  <!ELEMENT Year (#PCDATA)>
  <!ENTITY MS "Thatz life">
]>
<Music_Library>
  <Title>YLO</Title>
  <Artist>&MS; - Jenny Dan</Artist>
  <Country>Germany</Country>
  <Price>$12</Price>
  <Year>2002</Year>
</Music_Library>
  
```

where,
The first block indicates xml declaration and document type declaration. Music_Library is the root element.

Using slides 14 to 17, explain the XML document structure.

XML documents are commonly stored in text files with extension `.xml`. The two sections of an XML document are as follows:

- **Document Prolog:** XML parser gets information about the content in the document with the help of document prolog. Document prolog contains metadata and consists of two parts - XML Declaration and DTD. XML Declaration specifies the version of XML being used. DTD defines entities' or attributes' values and checks grammar and vocabulary of markup.
- **Root Element:** The second is an element called the root element. The root element is also called a document element. It must contain all the other elements and content in the document. An XML element has a start tag and end tag.

Tell them that an XML document contains XML elements. An XML element is everything from (including) the element's start tag to (including) the element's end tag.

An element can contain:

- other elements
- text
- attributes
- or a mix of all

Figure on slide 16 shows the XML document structure. Explain the code which demonstrates the XML document structure using slide 17.

Tell them that the first block indicates xml declaration. The second block shows the DTD of XML document and finally, the third block shows XML documents with `Music_Library` as the root element.

In-Class Question:

After you finish the XML documents structure, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Why you should use DTD for an XML document?

Answer:

1. To follow or agree on the standards for interchanging data.
2. To verify that the XML data received from the other application is valid.

Slides 18 and 19

Let us understand the logical structure of XML document.

Logical Structure 1-2

- The logical structure gives information about the elements and the order in which they are to be included in the document.
- It shows how a document is constructed rather than what it contains.
- Document Prolog forms the basis of the logical structure of the XML document.
- XML Declaration and Document Type Definition are its components.

© Aptech Limited XML Simplified / Module 1 / 18 of 55

Logical Structure 2-2

Code Snippet

```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

Code Snippet

```
<!DOCTYPE Music_Library SYSTEM "Mlibrary.dtd">
```

© Aptech Limited XML Simplified / Module 1 / 19 of 55

Using slides 18 and 19, explain the logical structure of XML document.

The logical structure of an XML document gives information about the elements and the order in which they are to be included in the document. It shows how a document is constructed rather than what it contains.

Document Prolog forms the basis of the logical structure of the XML document. It contains the following components:

- **XML Declaration** - Gives the version of the XML specification and also identifies the character encoding scheme.
Example: `<?xml version="1.0" encoding="iso-8859-1" ?>`
- **DTD** - Defines a set of rules to which the XML document conforms. A DTD can be either external or internal. In this case, the DTD is an external file.
Example: `<!DOCTYPE Music_Library SYSTEM "Mlibrary.dtd">`

Tips:

An XML document validated against a DTD is "Well Formed" and "Valid".

In-Class Question:

After you finish the logical structure of XML, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



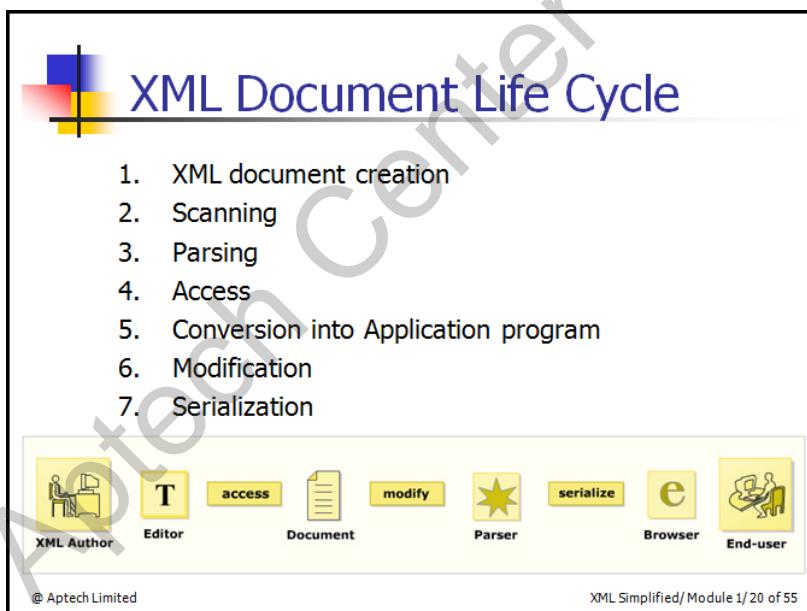
If the XML document contains international characters, then which part of the declaration must be specified to avoid errors?

Answer:

You can specify the encoding used in the XML document.

Slide 20

Let us understand the XML document life cycle.



Using slide 20, explain the XML document life cycle.

An XML editor refers to the DTD and creates an XML document. After the document is created, the document is scanned for elements and attributes in it. This stage is known as **scanning**. The XML parser builds a complete data structure after parsing. The data is then extracted from elements and attributes of the document. This stage is known as **access**. It is then converted into the application program. The document structure can also be modified during the process by inserting or deleting elements or by changing textual content of element or attribute. This stage is known as **modification**. The data is then serialized to a

textual form and is passed to a browser or any other application that can display it. This stage is known as **serialization**.

Figure shown on the slide displays the XML document life cycle.

Slide 21

Let us understand the editors.

Main Functions:

- Add opening and closing tags to the code
- Check for validity of XML
- Verify XML against a DTD/Schema
- Perform series of transforms over a document
- Color the XML syntax
- Display the line numbers
- Present the content and hide the code
- Complete the word

Some popularly used editors are:

- XMLwriter
- XML Spy
- XML Pro
- XMLmind
- XMetal

© Aptech Limited XML Simplified/ Module 1/ 21 of 55

Using slide 21, explain the editors.

An XML Editor is used to create and edit XML documents. Any application can be used as an editor in XML. Since all XML documents are text-based markup languages, a standard Windows Notepad or WordPad can also be used. However, for various reasons, Notepad should not be used for writing professional XML. Notepad does not know that the text written in it is XML code, and thus, can create problem.

For an XML document to be error-free and possess XML-specific features such as the ability to edit elements and attributes, a professional XML editor should be used.

Explain the main functions that XML editors must provide and also list the names of some of the popularly used XML editors.

Slides 22 and 23

Let us understand the parsers.



Parsers 1-2

- **An XML parser:**
 - Reads the document
 - Verifies it for its well-formedness
 - After verification, converts it into a tree of elements
- **Commonly used parsers:**
 - Crimson, Xerces, Oracle XML Parser, JAXP, MSXML
- **Type of parsers:**
 - Non Validating parser
 - Validating parser

© Aptech Limited

XML Simplified/ Module 1/ 22 of 55



Parsers 2-2

- Non Validating parser**
 - It checks the well-formedness of the document.
 - Reads the document and checks for its conformity with XML standards.
- Validating parser**
 - It checks the validity of the document using DTD.

© Aptech Limited

XML Simplified/ Module 1/ 23 of 55

Using slides 22 and 23, explain the parsers.

An XML parser or XML processor reads the document and verifies it for its well-formedness.

An XML parser keeps the entire representation of XML data in memory with the help of Document Object Model (DOM). The in-built parser used in IE 5.0 is also known as Microsoft XML Parser (MSXML). It is a component, which is available once IE 5.0 is installed. Microsoft's XML parser goes through the entire data structure and accesses the values of the attributes in the elements. The parser also creates or deletes the elements and converts the tree structure into XML.

MSXML supports some of the COM objects for backward interoperability. Some of them are XSLProcessor objects, XSLTemplate objects, XMLDOMSelection objects, XMLSchemaCache objects, and XMLDOMDocument2 objects.

XML parsing in Mozilla performs functions such as going through the elements, accessing their values, and so on. Using JavaScript, an instance of XML parser can be created in Mozilla browsers. The XML parsing in Firefox automatically parses the data into DOM. Opera uses only those XML parsers that do not validate DTDs by default.

Speed and performance are the criteria against which XML parsers are selected. Then, mention the commonly used XML parsers.

Also, explain the types of parsers in details using slide 23.

Slide 24

Let us understand the browsers.



Browsers

- Web browsers can format XML data and display it to the user.
- Other programs like database, Musical Instrument Digital Interface (MIDI) program or a spreadsheet program may present XML data accordingly.
- Commonly used web browsers:
 - Netscape, Mozilla, Internet Explorer, Firefox, Opera

@ Aptech Limited XML Simplified/ Module 1/ 24 of 55

Using slide 24, explain the browsers.

After the XML document is read, the parser passes the data structure to the client application. The application can be a Web browser. The browser then formats the data and displays it to the user. Other programs such as database, Musical Instrument Digital Interface (MIDI) program, or a spreadsheet program may also receive the data and present it accordingly.

The final output of XML data is viewed in a browser. XML is not supported by all browsers, for example, Netscape Navigator 4.0 does not support XML but later versions of the browser like Netscape 6.0 do support it. Only browsers like IE 5.0 or greater give full support for XML

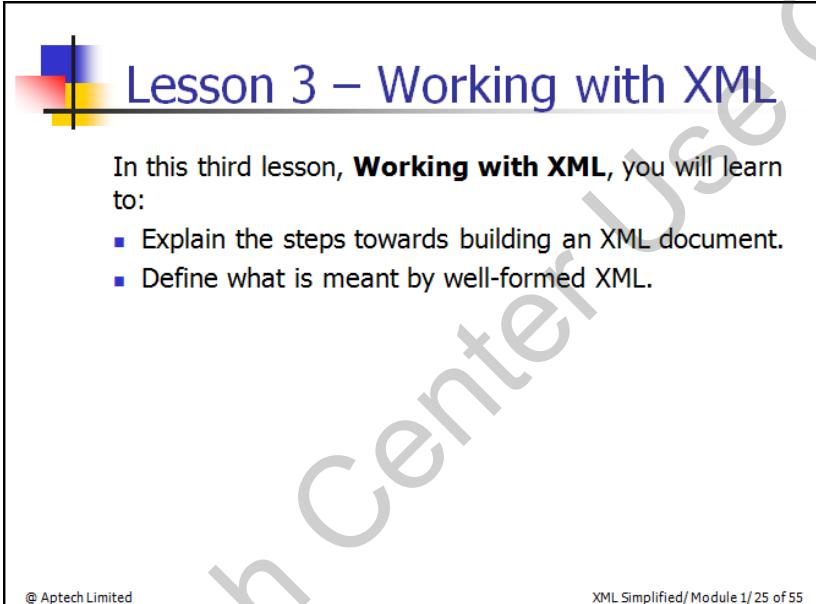
specifications. In IE, XML can be directly viewed using style sheets. It gives support to namespaces and handles a mechanism known as **data islands** where XML is embedded into HTML.

Mozilla 5.0 uses an interface to the XML DOM via JavaScript and plug-ins. It also supports elements from the HTML namespace.

Mention the commonly used browsers.

Slide 25

Let us understand the objectives of lesson 3.



The slide content is contained within a rectangular frame. At the top left is a graphic of three overlapping squares in red, blue, and yellow. To its right is the title "Lesson 3 – Working with XML". Below the title is a paragraph of text. To the right of the text is a bulleted list. At the bottom left is a copyright notice, and at the bottom right is a page number.

In this third lesson, **Working with XML**, you will learn to:

- Explain the steps towards building an XML document.
- Define what is meant by well-formed XML.

© Aptech Limited XML Simplified/ Module 1/ 25 of 55

Using slide 25, describe the topics covered in lesson 3.

Slide 26

Let us understand how to create a XML document.



Creating an XML document

- An XML document has three main components:
 - Tags(markup) and text(content)
 - DTD or Schema
 - Formatting or display specifications

- The steps to build an XML document are as follows:
 1. Create an XML document in an editor
 2. Save the XML document
 3. Load the XML document in a browser

@ Aptech Limited

XML Simplified/ Module 1/ 26 of 55

Using slide 26, explain how to create a XML document.

The steps to build an XML document are as follows:

➤ **Create an XML document in an editor**

To create a simple XML document, you type XML code in any text or XML editor.

Explain the following code snippet in demonstration:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Soft_drink>
<Name>Refresh</Name>
<Price>$1</Price>
</Soft_drink>
```

➤ **Saving the XML file**

After the typing is done, you save the contents of the file. To save the file, you click the **File** menu, click the **Save** option and provide the file name such as, **SoftDrink.xml**. The extension **.xml** is compulsory if you are typing the code in Notepad or any kind of word processor such as Microsoft Word.

➤ **Load XML document in a browser**

After the document is saved, you can open the file directly in a browser that supports XML. A common browser used is Internet Explorer 5.0 or later.

Slides 27 to 29

Let us understand the building blocks of XML document.

Exploring the XML document 1-3

- Various building blocks of an XML document:
 - XML Version Declaration
 - Document Type Definition
 - Document instance in which the content is defined by the mark up

@ Aptech Limited XML Simplified/ Module 1/ 27 of 55

Exploring the XML document 2-3

```

1  <?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
2
3  <!DOCTYPE student [
4    <!ELEMENT student (name,dob,bloodgroup,rollnumber)>
5    <!ELEMENT name (#PCDATA)>
6    <!ELEMENT dob (#PCDATA)>
7    <!ELEMENT bloodgroup (#PCDATA)>
8    <!ELEMENT rollnumber (#PCDATA)>
9  ]>
10
11 <student>
12   <name>James</name>
13   <dob>26th September, 1983</dob>
14   <bloodgroup>A positive</bloodgroup>
15   <rollnumber>17</rollnumber>
16 </student>

```

Characters are encoded using various encoding formats. The character encoding is declared in encoding declaration.

XML declaration tells the version of XML, the type of character encoding being used and the markup declaration used in the XML document.

XML declaration should start with the five character string, <?xml. It indicates that the document is an XML document.

@ Aptech Limited XML Simplified/ Module 1/ 28 of 55

Exploring the XML document 3-3

```

1  <?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
2
3  <!DOCTYPE student [
4      <!ELEMENT student (name,dob,bloodgroup,rollnumber)>
5      <!ELEMENT name (#PCDATA)>
6      <!ELEMENT dob (#PCDATA)>
7      <!ELEMENT bloodgroup (#PCDATA)>
8      <!ELEMENT rollnumber (#PCDATA)>
9  ]>
10
11 <student>
12     <name>James</name>
13     <dob>26th September, 1983</dob>
14     <bloodgroup>A positive</bloodgroup>
15     <rollnumber>17</rollnumber>
16 </student>

```

Indicates the presence of external markup declarations. "Yes" indicates that there are no external markup declarations and "no" indicate that external markup declarations might exist.

Declares and defines the elements used in the document class.

Defines the content of the XML document.

© Aptech Limited XML Simplified/Module 1/29 of 55

Using slides 27 to 29, explain the the building blocks of XML document.

Figure on slide 28 depicts a XML document. Explain the code in the figure with each block in details.

➤ <?xml

The XML declaration should start with the five character string, <?xml. It indicates that the document is an XML document.

➤ version="1.0"

The XML declaration tells the version of XML, the type of character encoding being used and the markup declaration used in the XML document.

➤ encoding="iso-8859-1"

Characters are encoded using various encoding formats. The character encoding is declared in encoding declaration.

➤ standalone="yes"

The standalone declaration indicates the presence of external markup declarations. "Yes" indicates that there are no external markup declarations and "no" indicates that external markup declarations might exist.

DTD for the XML document can be declared internally or externally.

Slide 30

Let us understand the meaning of components in markup.



Meaning in Markup

- Structure
- Semantic
- Style

@ Aptech Limited

XML Simplified/ Module 1/ 30 of 55

Using slide 30, explain the meaning of components in markup.

Markup can be divided into:

- **Structure**
It describes the form of the document by specifying the relationship between different elements in the document. It emphasizes to specify a single non-empty, root element that contains other elements, and the content.
- **Semantic**
Semantics describes how each element is specified to the outside world of the document. For example, an HTML enabled Web browser assigns "paragraph" to the tags <P> and </P>, but not to the tags <Message> and </Message>.
- **Style**
It specifies how the content of the tag or element is displayed. It indicates whether the tag is bold, normal, and pink in color or with the font size 10.

In-Class Question:

After you finish the components of XML, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



An XML document with correct syntax is referred to as ____.

Answer:

Well-formed

Slides 31 to 33

Let us understand the well-formed XML document.



Well-formed XML document 1-3



Well-formed XML document 2-3



Well-formed XML document 3-3

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Library>
<Name>
4ULibrary
#1, A Mansion, Izaho
$10000
</Library>
```

Non Well- Formed Document

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Library>
<Name>4ULibrary</Name>
<Address> #1, A Mansion, Izaho </Address>
<Auction>$10000</Auction>
</Library>
```

Well-Formed Document

@ Aptech Limited

XML Simplified/Module 1/33 of 55

Using slides 31 to 33, explain the well-formed XML document.

Well-formedness refers to the standards that are to be followed by the XML documents. Well-formedness makes XML processors and browsers read XML documents.

Minimum of One Element is Required

Every well-formed XML document should consist of a minimum of one element.

All XML Elements Must Have a Closing Tag

In HTML, some elements do not have a closing tag. For example, `
`. In XML, all elements must have a closing tag.

For example, consider the following XML syntax:

```
<p> This is a paragraph. </p>
<br />
```

XML Tags are Case Sensitive

XML tags are case sensitive. The tag `<author>` is different from the tag `<Author>`. Opening and closing tags must be written with the same case.

For example,

```
<author>This is incorrect</Author>
<author>This is correct</author>
```

XML Elements Must be Properly Nested

In HTML, you might see improperly nested elements. For example, `<i>This text is bold and italic</i>`. In XML, all elements must be properly nested within each other. For example, `<i>This text is bold and italic</i>`.

XML Documents Must Have a Root Element

XML documents must contain one element that is the **parent** of all other elements. This element is called the **root** element.

XML Attribute Values Must be Quoted

XML elements can have attributes in name/value pairs just like in HTML. In XML, the attribute values must always be quoted.

```
For example, <Person age="27">
    <first_name>John</first_name>
    <last_name>Jani</last_name>
</Person>
```

Comments in XML

The syntax for writing comments in XML is similar to that of HTML. For example, `<!-- This is a comment -->`

White-space is Preserved in XML

In HTML, multiple white-space characters are truncated to one single white-space. In XML, the white-spaces in a document is not truncated.

XML Tags should be Valid

- Tags should begin with letter, an underscore (_), or a colon (:).
- Tags should contain combination of letters, numbers, periods (.), colons, underscores, or hyphens (-).
- Tags should not contain white space.
- Tags should not start with reserved words like 'xml'.

If an XML documents that conform to the syntax rules are said to be 'Well formed' XML documents.

Slide 34

Let us understand the objectives of lesson 4.



Lesson 4 – XML Syntax

In this last lesson, **XML Syntax**, you will learn to:

- State and describe the use of comments and processing instructions in XML.
- Classify character data that is written between tags.
- Describe entities, DOCTYPE declarations and attributes.

@ Aptech Limited XML Simplified/ Module 1/34 of 55

Using slide 34, explain the topics to be covered in lesson 4.

Slides 35 to 37

Let us understand the comments.



Comments 1-3

- Used for the people to give information about the code
- Usually made for the content

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <Name NickName="John">
  <First>John</First>
  <!-- John is yet to pay the term fees -->
  <Last>Brown</Last>
  <Semester>Final</Semester>
</Name>
```

@ Aptech Limited XML Simplified/ Module 1/35 of 55



Comments 2-3

■ Rules:

- Comments should not include “-” or “_”
- Comments should not be placed within a tag or entity declaration
- Comments should not be placed before the XML declaration
- Comments can be used to comment the tag sets
- Comment should not be nested



Comments 3-3

Code Snippet

```
<Name NickName='John'>
    <First>John</First>
    <!--John is yet to pay the term fees-->
    <Last>Brown</Last> <Semester>Final</Semester>
</Name>
```

Using slides 35 to 37, explain the comments.

XML comments are used for the people to give information about the code in the absence of the developer. It makes a document more readable. Comments are not restricted to DTD, but may be placed anywhere in the document.

Comments in XML are similar to those in HTML. Comments should be used only when needed, as they are not processed. Comments are used only for human consumption rather than machine consumption. Since the comments are not parsed, their presence or absence does not make any difference to the processors.

Comments start with the string `<!--` and end with the string `-->`. The parser believes that the comment has come to an end when it finds a `>` as shown in figure on slide 35.

Explain the rules for writing comment and demonstrate the code using slides 36 and 37.

Slides 38 and 39

Let us understand the processing instructions.



Processing Instructions 1-2

- The main objective is to present some special instructions to the application.
- All the processing instructions should begin with an identifier called target.

Syntax

```
<?PITarget <instruction>??>
```

where,

PITarget is the name of the application that should receive the processing instructions.

<instruction> is the instruction for the application

© Aptech Limited

XML Simplified/Module 1/38 of 55



Processing Instructions 2-2

Code Snippet

```
<Name NickName='John'>
  <First>John</First>
  <!--John is yet to pay the term fees-->
  <Last>Brown</Last>
  <?feesprocessor SELECT fees FROM
STUDENTFEES??>
  <Semester>Final</Semester>
</Name>
```

where,

feesprocessor is the name of the application that receives the processing instruction

SELECT fees FROM STUDENTFEES is the instruction.

© Aptech Limited

XML Simplified/Module 1/39 of 55

Using slides 38 and 39, explain the processing instructions.

Processing instructions are information which is application specific. These instructions do not follow XML rules or internal syntax. With the help of a parser, these instructions are passed to the application. The application can either use the processing instructions or pass them on to another application.

The main objective of a processing instruction is to present some special instructions to the application. All processing instructions must begin with `<?` and end with `?>`.

Though an XML declaration also begins with <? and end with ?>, it is not considered as a processing instruction. It is because an XML declaration provides information only for the parsers and not for the application. In some cases, the application might need the information in the processing instruction only if it displays the output to the user.

Then, explain the syntax using slide 38 to write process instructions. Also, explain the code on slide 39 which demonstrates an example of processing instruction.

Slide 40

Let us understand the classification of character data.



Classification of character data

- Character data describes the document's actual content with the white space.
- The character data can be classified into:
 - Character Data (CDATA)
 - Parsed Character Data (PCDATA)

@ Aptech Limited

XML Simplified/Module 1/ 40 of 55

Using slide 40, explain the classification of character data.

An XML document is divided into markup and character data.

Character data describes the document's actual content with the white space. The text in character data is not processed by the parser and thus, not treated as a regular text. The character data can be classified into:

- CDATA
- PCDATA

Slides 41 and 42

Let us understand the PCDATA.

PCDATA 1-2

- The data that is parsed by the parser is called as PCDATA.
- The main objective is to present some special instructions to the application.

Output

The XML page cannot be displayed

Cannot view XML input using XSL style sheet. Please correct the error and then click the [Refresh](#) button, or try again later.

Whitespace is not allowed at this location. Error processing resource 'file:///D:/XML By Example/Data.xml'. Line 5, Position...

```
<Semester>Final> 10 & <20</Semester>
-----^
```

@ Aptech Limited XML Simplified/Module 1/ 41 of 55

PCDATA 2-2

Code Snippet

```
<Name nickname='John'>
    <First>John</First>
    <!--John is yet to pay the term fees-->
    <Last>Brown</Last>
    <Semester>Final> 10 & <20</Semester>
</Name>
```

@ Aptech Limited XML Simplified/Module 1/ 42 of 55

Using slides 41 and 42, explain the PCDATA.

The data that is parsed by the parser is called as parsed character data (PCDATA). The PCDATA specifies that the element has parsed character data. It is used in the element declaration.

Escape character like '<' when used in the XML document, will make the parser interpret it as a new element. As a result, it will generate an error as shown in figure on slide 41.

Explain the code on slide 42 which demonstrates an example of PCDATA.

Tips:

To avoid error for special characters, replace the '<' character with an entity reference.

Slide 43

Let us understand the CDATA.



CDATA

- CDATA part begins with a "<! [CDATA[" and ends with "]]>".
- CDATA sections cannot be nested.

Code Snippet

```
<?xml version="1.0" standalone="yes"?>
<Svg>
  <Desc>Three shapes</Desc>
  <Para>But the formula was <! [CDATA[if (&x1 +
&x2)]]> which
resulted in 7.</Para>
</Svg>>
```

© Aptech Limited XML Simplified/ Module 1/ 43 of 55

Using slide 43, explain the CDATA.

CDATA stands for character data that has reserved and white space characters in it. Though the text inside the CDATA is not parsed by the parser, it is commonly used for scripting code. XML parser ignores all tags and entity references inside the CDATA blocks. The CDATA block indicates to the parser that it is just a text but not a markup. A CDATA block always begins with the delimiter <! [CDATA[and ends with the delimiter]]>.

Since the ending delimiter marks the end of the CDATA block, the character string]] / is not allowed in the middle of the CDATA block. This will signal the end of the CDATA section.

Explain the syntax of CDATA and then explain the code on slide 43 which demonstrates CDATA.

Slides 44 to 46

Let us understand the entities.

Entities 1-3

- XML document is made up of large amount of information called as entities.
- Every entity consists a name and a value.
- Entity reference consists of an ampersand (&), the entity name, and a semicolon (;).

Entities 2-3

Predefined Entity	Description	Output
<	produces the left angle bracket	<
>	produces the right angle bracket	>
&	produces the ampersand	&
'	produces a single quote character	'
"	produces a double quote character	"

Entities 3-3

Code Snippet

```

<?xml version="1.0"?>
<!DOCTYPE Letter [
<!ENTITY address "15 Downing St Floor 1">
<!ENTITY city "New York">
]>
<Letter>
<To>&quot;Tom Smith&quot;</To>
    <Address>&address;</Address>
    <City>&city;</City>
    <Body>
        Hi! How are you?
        The sum is &gt; $1000
    </Body>
    <From>ARNOLD</From>
</Letter>

```

Using slides 44 to 46, explain the entities.

The XML document is made up of large amount of information called as entities. Entities are used to avoid typing long pieces of text repeatedly, within the document.

Explain the categories of entities are as follows:

- **Character entities:** They form the mechanism, which is used in place of a character's literal form. They provide the meaning of '>' when the symbol '>' is typed. Character entities can also be used with decimal or hexadecimal values with a condition that the numbers support Unicode coding. An XML processing program replaces character entities with their equivalent characters.
- **Content entities:** These entities are used to replace certain values. They are similar to text substitution macros in programming languages such as C. Content entity has the following syntax: <!Entity name value>
- **Unparsed entities:** These entities when used, turn off the parsing process. They can be used to include multimedia content in the XML document.

Every entity consists a name and a value. The value ranges from a single character to a XML markup file. As the XML document is parsed, it checks for entity references. For every entity reference, the parser checks the memory to replace the entity reference with a text or markup.

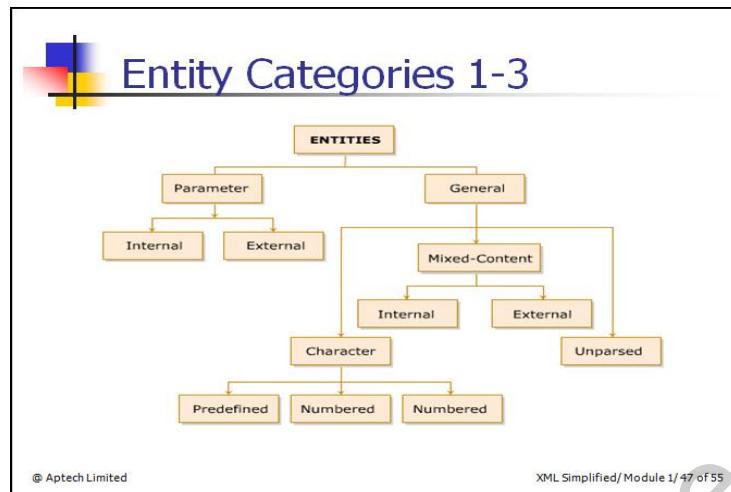
All the entities must be declared before they are used in the document. An entity can be declared either in a document prolog or in a DTD.

Some of the entities are defined in the system and are known as pre-defined entities. These entities are described in table shown on slide 45.

Then, explain the code on slide 46 which demonstrates an example of entities.

Slides 47 to 49

Let us understand the entity categories.



Entity Categories 2-3

- **General Entity**
 - These entities are used within the document content.

Code Snippet

```
<<!ENTITY % ADDRESS "text that is to be
represented by an entity">
```

A well-formed parameter entity will look like a general entity, except that it will include the "%" specifier.

Entity Categories 3-3

- **Parameter Entity**
 - These entities are used only in the DTD.

Code Snippet

```
<!DOCTYPE MusicCollection [
  <!ENTITY R "Rock">
  <!ENTITY S "Soft">
  <!ENTITY RA "Rap">
  <!ENTITY HH "Hiphop">
  <!ENTITY F "Folk">
]>
```

Using slides 47 to 49, explain the entity categories.

Entities are used as shortcuts to refer to the data pages. There are two types of entities that are as follows:

- General Entity
- Parameter Entity

General entities are used within the document content. General entities can either be declared internally or externally. The references for general entities start out with an ampersand (&) and end with a semicolon (;). The entity's name is present within these two characters.

Explain the code on slide 48 which demonstrates an example of general entity.

Mention parameter entities are used only in the DTD. These types of entities are declared in DTD. Both internal and external parameter entities should not be used in the content of the XML document as the processor does not recognise them. A well-formed parameter entity is similar to general entity, except that, it will include the % specifier. The reference is also similar to the general entity reference. References to these entities are made by using percent-sign (%) and semicolon (;) as delimiters.

Explain the code on slide 49 which demonstrates an example of parameter entity.

Slides 50 to 52

Let us understand the DOCTYPE declarations.

The slide has a decorative graphic in the top-left corner consisting of overlapping colored squares (blue, red, yellow).

DOCTYPE declarations 1-3

- **DTD File:**

Syntax

```
<!DOCTYPE name_of_root_element  
         SYSTEM "URL of the external DTD subset" [  
               Internal DTD subset  
           ]>
```

where,
name_of_root_element is the name of the root element
SYSTEM is the url where the DTD is located
[Internal DTD subset] are declarations in the document

© Aptech Limited XML Simplified/Module 1/ 50 of 55



DOCTYPE declarations 2-3

Code Snippet

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE program SYSTEM "HelloJimmy.dtd">
<Program>
<Comments>
This is a simple Java Program. It will display
the message "Hello Jimmy, How are you?" on
execution.
</Comments>
<Code> public static void main(String[] args) {
System.out.println("Hello Jimmy, How are you?");
// Display the string.
}
</Code>
</Program>
```

@ Aptech Limited XML Simplified/Module 1/51 of 55



DOCTYPE declarations 3-3

- Document Type Definition defines the elements in the document

```
<!ELEMENT Program (comments, code)>
<!ELEMENT Comments (#PCDATA)>
<!ELEMENT Code (#PCDATA)>
```

- Output:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE program (View Source for full doctype...)>
- <Program>
  <Comments>This is a simple Java Program. It will display the message "Hello
  Jimmy, How are you?" on execution.</Comments>
  <Code>public static void main(String[] args) { System.out.println("Hello
  Jimmy, How are you?"); // Display the string. } </Code>
</Program>
```

@ Aptech Limited XML Simplified/Module 1/52 of 55

Using slides 50 to 52, explain the DOCTYPE declarations.

The DTD defines the elements to be used in the document. A DTD is declared to indicate what DTD the document adheres to. It can be declared either in the XML document or can be referenced to the external document.

Explain the syntax of document type declaration using slide 50.

Explain the code on slide 51 demonstrates an example of document type declaration.

Slide 53

Let us understand the attributes.



Attributes

- Attributes are case sensitive and must start with a letter or underscore.

Syntax

```
<elementName attName1="attValue2"
attName2="attValue2" ...>
```

Code Snippet

```
<?xml version="1.0" ?>
<Player Sex="male">
  <FirstName>Tom</FirstName>
  <LastName>Federer</LastName>
</Player>
```

© Aptech Limited

XML Simplified / Module 1 / 53 of 55

Using slide 53, explain the attributes in XML document.

Attributes are part of the elements. They provide information about the element and are embedded in the element start-tag. An attribute consists of an attribute name and an attribute value. The name always precedes its value, which are separated by an equal sign. The attribute value is enclosed in the quotes to delimit multiple attributes in the same element.

An attribute can be of CDATA, ENTITY, ENUMERATION, ID, IDREF, NMTOKEN, or NOTATION.

Explain the syntax and code snippet which demonstrates an example of attributes using slide 53.

Tips:

Some of the problems with using attributes are:

- Attributes cannot contain multiple values.
- Attributes cannot contain tree structures.
- Attributes are not easily expandable.

Slides 54 and 55

Let us summarize the session.



Summary 1-2

- **Introduction to XML**
 - XML was developed to overcome the drawbacks of earlier markup languages.
 - XML consists of set of rules that describe the content to be displayed in the document.
 - XML markup contains the content in the information containers called as elements.

- **Exploring XML**
 - XML is divided into two parts namely, document prolog and root element.
 - An XML editor creates the XML document and the parser validates the document.

© Aptech Limited

XML Simplified/ Module 1/ 54 of 55



Summary 2-2

- **Working with XML**
 - XML document is divided into XML Version Declaration, DTD and the document instance in which the markup defines the content.
 - The XML markup is again categorized into structural, semantic and stylistic.
 - The output of the XML document is displayed in the browser if it is well formed.

- **XML Syntax**
 - Comments are used in the document to give information about the line or block of code.
 - The content in XML document is divided into markup and character data.
 - The entities in XML are divided into general entities and parameter entities.
 - A DTD can be declared either internally or externally.

© Aptech Limited

XML Simplified/ Module 1/ 55 of 55

Using slides 54 and 55, summarize the session. End the session with a brief summary of what has been taught in the session.

1.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Namespaces that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 2 - Namespaces

2.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

2.1.1 Objectives

By the end of this session, the learners will be able to:

- XML Namespaces
- Working with Namespaces Syntax

2.1.2 Teaching Skills

To teach this session, you should be well-versed with XML namespaces and the reasons for using namespaces in XML documents. You should also aware yourself with the syntax of creating namespaces.

To teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Then give the students the overview of the current session in the form of Session Overview. Show the students slide 2 of the presentation.

Tell the students that this session introduces them to the need for a namespace. They will learn the syntax for namespace attribute in XML. They will also learn how to use default namespaces.

2.2 In-Class Explanations

Slide 3

Let us understand the objectives of lesson 1.

A decorative graphic in the top-left corner featuring overlapping colored squares (blue, red, yellow) and a crosshair pattern.

Lesson 1 – XML Namespaces

In this first lesson, **XML Namespaces**, you will learn to:

- Identify the need for a namespace.
- Define and describe namespaces in XML.

© Aptech Limited

XML Simplified / Module 2 / 3 of 20

Using slide 3, explain the topics to be covered in lesson 1.

Slide 4

Let us understand the duplicate elements name.



Duplicate Element Names

- It allows developers to create their own elements and attributes for their own projects.
- Developer has to ensure the uniqueness of the element names and attributes in a document.

© Aptech Limited

XML Simplified / Module 2 / 4 of 20

Using slide 4, explain the duplicate elements name.

Mention XML allows developers to create the XML documents with their own elements and attributes within the projects. Now, it may happen that element names defined by one developer may get repeated by some other developer while creating XML documents.

For example, an author of an XML document includes element called <title> in a <CD> element and another author creates an element <title> in a <Book> element. A problem will arise if the <title> element created by two different authors, is merged into a single XML document. Thus, XML developer has to ensure the uniqueness of the element names and attributes in a document.

Slide 5

Let us understand consequences of duplicate element names.

The slide has a decorative graphic in the top-left corner consisting of overlapping colored squares (blue, red, yellow). The title 'Consequences of Duplicate Element Names' is centered above a bulleted list. Below the list is a code snippet in a light gray box, followed by a callout bubble containing a question. At the bottom left is the copyright notice, and at the bottom right is the module information.

Consequences of Duplicate Element Names

- Name conflicts are inevitable from different developers.
- It is difficult for the browser to distinguish a conflicting element.

```
<Cd>
  <Name> ... </Name>
</Cd>
<Singer>
  <Name> ... </Name>
</Singer>
```

How do I search for all CD titles/names?
How do I differentiate between cd name and singer name?

© Aptech Limited XML Simplified/Module 2/ 5 of 20

Using slide 5, explain the consequences of using duplicate element names.

When authors begin integrating the XML documents from different developers, name conflicts are inevitable. In such a scenario, it becomes difficult for the browser to distinguish a conflicting element in a XML document.

Explain the figure provided on the slide to explain the assigning of duplicate names to elements in an XML document.

Slide 6

Let us understand the namespaces.

Namespaces

- Elements are distinguished by using namespaces.
- A namespace is a collection of names.
- Namespaces allow the browser to:
 - Combine documents from different sources
 - Identify the source of elements or attributes

© Aptech Limited XML Simplified/ Module 2/ 6 of 20

Using slide 6, explain the namespaces.

In XML, elements are distinguished by using namespaces. XML namespaces provide a globally unique name for an element or attribute so that they do not conflict one another.

A namespace is a collection of names that can be used as element names or attribute names in XML document.

XML namespaces provide the following advantages:

➤ **Reusability**

XML Namespaces enable reuse of markup by making use of the elements and attributes that were defined earlier.

➤ **Modularity**

Reusable code sessions can be written, and these can be invoked for specific elements or attributes. Elements and attributes from different sessions can be integrated into a single XML document. Universally unique element names and attributes guarantee that such sessions can be invoked for certain elements and attributes.

➤ **Extensibility**

XML namespaces provide the XML documents with the ability to embed elements and attributes from other vocabularies such as MathML, eXtensible HyperText Markup Language (XHTML), and so forth.

Slide 7

Let us understand the objectives of lesson 2.



Lesson 2 – Working with Namespaces Syntax

In this last lesson, **Working with Namespaces syntax**, you will learn to:

- Explain the syntax for XML namespaces.
- Discuss attributes and namespaces.
- Discuss how to use default namespaces.

© Aptech Limited XML Simplified/Module 2/ 7 of 20

Using slide 7, explain the topics to be covered in the lesson 2.

Slide 8

Let us understand how to prefix an element name.



Prefixing element names

- Prefixes in element names provide a means to prevent name collisions.

Code Snippet

```
<CD:Title> Feel </CD:Title>
and
<Book:Title> Returning to Earth </Book:Title>.
```

In the above example, both CD and Book are namespace prefixes.

© Aptech Limited XML Simplified/Module 2/ 8 of 20

Using slide 8, explain how to prefix an element name.

Namespaces are a mechanism by which element and attribute names can be assigned to groups. They also ensure that there is no conflict within element names. The best way to solve this problem is for every element in a document to have a completely distinct name. Therefore, the XML namespace prefixes are directly attached to names as well as names of its attributes and descendants.

Mention, using prefixes in the element names provide a means for the document authors to prevent name collisions, as demonstrated in the code shown in slide 8.

Explain the same scenario of `<title>` duplication in `<CD>` and `<Book>`, demonstrate prefixing of the elements with namespace prefix.

<CD:Title> Feel </CD:Title>

and

<Book:Title> Returning to Earth </Book:Title>

Here, both CD and Book are namespace prefixes.

Slide 9

Let us understand the problems posed by prefixes.



Problems Posed by Prefixes

- Duplication would still exist if prefixes are not unique
- To solve this problem, each namespace prefix is added to a Uniform Resource Identifier (URI)

Code Snippet

```
<s:Student xmlns:s=" http://www.spectrafocus.com
/student/">
<s:First>John</s:First>
<s>Last>Dewey</s>Last>
<s>Title>Student</s>Title>
</s:Student>
```

© Aptech Limited
XML Simplified/ Module 2/ 9 of 20

Using slide 9, explain the problems posed by prefixes.

There is a drawback to the prefix approach of the namespaces in an XML document. The reason for prefixing an element in a XML document is to prevent it from being duplicated. However, if the prefixes are not unique, the original problem of duplication would still exist.

To solve this problem, each namespace prefix is added to a Uniform Resource Identifier or URI that uniquely identifies the namespace.

URI is a series of characters used to differentiate names.

To guarantee name uniqueness in any XML documents that the company creates, the documents are associated with their namespace.

Explain the code snippet demonstrating a company's URI which is located at <http://www.spectrafocus.com>.

Slides 10 to 12

Let us understand the namespace syntax.

Namespace Syntax 1-3



```
<namespacePrefix:  
elementName xmlns:  
namespacePrefix = 'URI'>
```

© Aptech Limited XML Simplified/ Module 2/ 10 of 20

Namespace Syntax 2-3



- NamespacePrefix**
 - Used as a reference to the namespace
 - Prefixes must not begin with xmlns or xml
- ElementName**
 - Specifies the name of the element
- xmlns**
 - xmlns stands for XML namespace
- URI**
 - URI is a string of characters which identifies an Internet Resource

© Aptech Limited XML Simplified/ Module 2/ 11 of 20

Namespace Syntax 3-3



Code Snippet

```
<Auc:Books  
xmlns:Auc="http://www.auction.com/books"  
xmlns:B="http://www.books.com/HTML/1998/xml1">  
...  
<Auc:BookReview>  
<B:Table>  
...
```

© Aptech Limited XML Simplified/ Module 2/ 12 of 20

Using slides 10 to 12, explain the namespace syntax.

The XML namespace attribute is placed in the start tag of an element and the syntax for the namespace is given in figure.

- **namespacePrefix** - Each namespace has a prefix that is used as a reference to the namespace. Prefixes must not begin with `xmlns` or `xml`. A namespace prefix can be any legal XML name that does not contain a colon. A legal XML name must begin with a letter or an underscore.
- **elementName** - It specifies the name of the element.
- **XmLns** - The `xmlns` attribute is what notifies an XML processor that a namespace is being declared. `xmlns` stands for XML namespace.
- **URI** – A URI is a string of characters which identifies an Internet Resource. The URI includes Uniform Resource Name (URN) and a Uniform Resource Locator (URL). URLs contain the reference for a document or an HTML page on the Web. URLs are also case-sensitive.

Tips:

The namespace URI is not parsed by the parser. Thus, its purpose is to uniquely identify the namespace.

In-Class Question:

After you finish explaining the namespace syntax, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Whether it is important to have valid URIs names for the resources?

Answer:

Not necessary, URIs do not need to be valid, as they are treated just like strings by XML namespace. Thus, these URIs are valid: `http://www.xmlnamespaces.com` or `http://xmlnamespaces.com`.

Slides 13 to 15

Let us understand how to place attributes in a namespaces.

Placing attributes in a Namespace 1-3

- Attributes belong to particular elements
- They are not a part of namespace, even if the element is within some namespace
- If an attribute has no prefix, it has no namespace
- An attribute without a prefix is in default namespace
- If an attribute name has a prefix, its name is in the namespace indicated by the prefix

Placing attributes in a Namespace 2-3

Syntax

```
<Catalog xmlns:Book =
"http://www.aptechworldwide.com">
<Book:Booklist>
<Book:Title Book:Type = "Fiction">Evening in
Paris</Book:Title>
<Book:Price>$123</Book:Price>
</Book:Booklist>
</Catalog>
```

Placing attributes in a Namespace 3-3

Code Snippet

```
prefix:localname='value'
or
prefix:localname="value"
```

where,
 prefix is used as a reference to the namespace. Prefixes must
 not begin with xmlns or xml
 localname is the name of an attribute
 value mentions a user defined value for an attribute

Using slides 13 to 15, explain how to place attributes in a namespaces.

Attributes belong to particular elements and they are not a part of namespace, even if the element is within some namespace. However, if an attribute name has no prefix, it has no namespace. An attribute without a prefix is in default namespace.

If an attribute name has a prefix, its name is in the namespace indicated by the prefix.

Syntax: Emp

`xmlns:prefix="URI"`

Explain code using slides 14 and 15 in which the `type` attribute is associated with the Book namespace, since it is preceded by the prefix Book.

Slides 16 and 17

Let us understand the default namespaces.

Default Namespaces 1-2

MathML Document

- XML-based markup language to represent complex mathematical expressions
- Comes in two types:
 - As a markup language for presenting the layout of mathematical expressions
 - As a markup language for presenting the mathematical content of the formula

Code Snippet

```
<MRow>
<Mi>x</Mi>
<Mo>+</Mo>
<Mn>1</Mn>
</MRow>
```

© Aptech Limited XML Simplified/ Module 2/ 16 of 20

Default Namespaces 2-2

Syntax

```
<elementName xmlns='URL'>
```

where,
`elementName` specifies the name of the element belonging to the same namespace
`URL` specifies the namespace which is reference for a document or an HTML page on the Web

Code Snippet

```
<Catalog xmlns:Book = "http://www.aptechworldwide.com">
<Book:Booklist>
  <Book:Title Book:Type = "Fiction">Evening in
  Paris</Book:Title>
  <Book:Price>$123</Book:Price>
</Book:Booklist>
</Catalog>
```

A default namespace using the `xmlns` attribute with a URI as its value

© Aptech Limited XML Simplified/ Module 2/ 17 of 20

Using slide 16, explain the default namespaces.

Consider an XML document which contains the details of all books in a library. Since this document includes a lot of markup all in the same namespace, it may be inconvenient to add a prefix to each element name.

This problem can be solved by adding a default namespace to an element and to its child elements using an `xmlns` attribute with no prefix.

A default namespace is used by an element and its child elements if the element does not have a namespace prefix.

Syntax:

```
xmlns="namespaceURI"
```

Explain the code which demonstrates the default namespace using slide 17.

A default namespace using the `xmlns` attribute with a URI as its value. Once this default namespace is declared, child elements that are part of this namespace do not need a namespace prefix.

In-Class Question:

After you finish explaining the default namespaces, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What are the drawbacks of omitting the namespace?

Answer:

1. It is difficult to understand which element belongs to which namespace.
2. Also, which namespace is applicable is difficult to understand.

Tips:

When a default namespace is declared, the namespace is applied only to the element, and not to any attributes.

Slides 18 and 19

Let us understand the override default namespaces.



Override Default Namespaces 1-2

- Default namespace applies to the element on which it was defined and all descendants of that element
- New namespace definition overrides the previous one and becomes the default for that element

© Aptech Limited

XML Simplified/Module 2/18 of 20



Override Default Namespaces 2-2

Code Snippet

```
<Catalog xmlns = "http://www.aptechworldwide.com">
  <Book>
    <Title type = "Fiction">Evening in Paris</Title>
    <Price>$123</Price>
  </Book>
  <Book>
    <Title type = "Non-Fiction">Return to Earth</Title>
    <Price xmlns = "http://www.aptech.ac.in">$23</Price>
    <Title type = "Non-Fiction">Journey to the center of
      the Moon</Title>
    <Price>$123</Price>
  </Book>
</Catalog>
```

This namespace of price element applies only to it and overrides the namespace in the catalog element

© Aptech Limited

XML Simplified/Module 2/19 of 20

Using slides 18 and 19, explain the override default namespaces.

The default namespace applies to the element on which it was defined and all descendants of that element. If one of the descendants has another default namespace defined on it, this new namespace definition overrides the previous one and becomes the default for that element and all its descendants as demonstrated in the code.

Notice that in the `price` element in the second book element a different namespace is provided. This namespace applies only to the `price` element and overrides the namespace in the `catalog` element.

Slide 20

Let us summarize the session.



Summary

- **XML Namespaces**
 - Namespaces distinguish between elements and attributes with the same name from different XML applications.
 - It is a collection of names that can be used as element names or attribute names in XML document.
 - XML namespaces provide a globally unique name for a element or attribute to avoid name collisions.
- **Working with Namespaces syntax**
 - Namespaces are declared by an `xmlns` attribute whose value is the URI of the namespace.
 - If an attribute name has no prefix, it has no namespace.
 - A default namespace is used by an element and its child elements if the element does not have a namespace prefix.

© Aptech Limited XML Simplified/ Module 2/20 of 20

Using slide 20, summarize the session. End the session, with a brief summary of what has been taught in the session.

2.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the DTDs that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 3 - DTDs

3.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

3.1.1 Objectives

By the end of this session, the learners will be able to:

- Document Type Definition
- Working with DTDs
- Valid XML Documents
- Declarations

3.1.2 Teaching Skills

To teach this session, you should be well-versed with how to create DTDs for XML files. The topics such as DOCTYPE declarations, types of DTDs, well-formedness, and validity of XML files. You should also aware yourself with the declaration of elements, attributes, and entities in a DTD.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

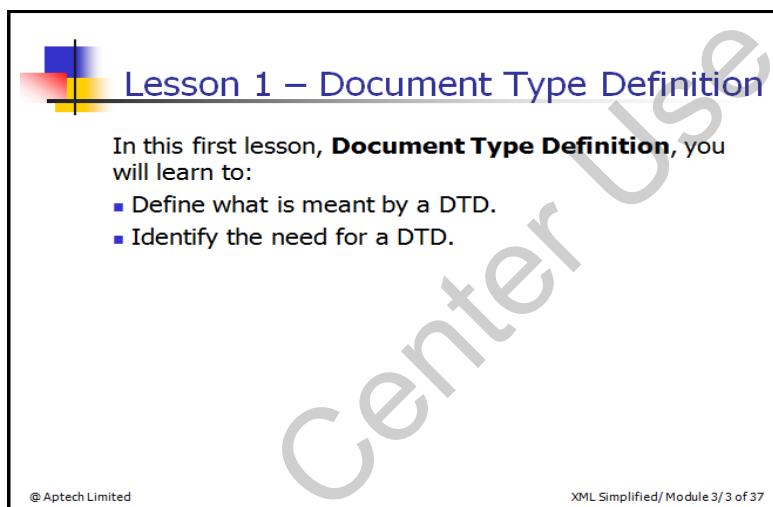
Then give the students the overview of the current session in the form of Session Overview. Show the students slide 2 of the presentation.

Tell the students that this session introduces DTDs and how to create DTDs for XML files. They will learn the need for a DTD and its structure, DOCTYPE declarations, types of DTDs, well-formedness, and validity of XML files. They will also learn how to declare elements, attributes, and entities in a DTD.

3.2 In-Class Explanations

Slide 3

Let us understand the objectives of lesson 1.

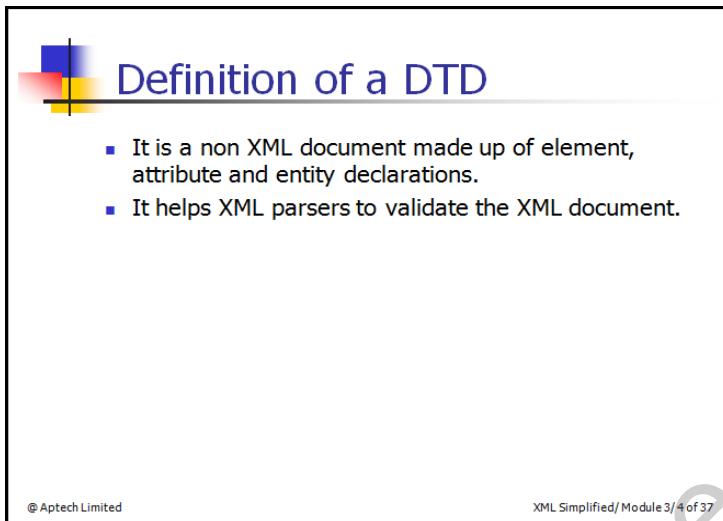


The slide has a decorative graphic in the top-left corner consisting of overlapping colored squares (red, blue, yellow) and a black crosshair. The title "Lesson 1 – Document Type Definition" is centered at the top. Below the title, the text reads: "In this first lesson, **Document Type Definition**, you will learn to:" followed by a bulleted list: "■ Define what is meant by a DTD." and "■ Identify the need for a DTD." At the bottom left is the copyright notice "@ Aptech Limited" and at the bottom right is the page number "XML Simplified/ Module 3/ 3 of 37".

Using slide 3, explain the topics covered in the lesson 1.

Slide 4

Let us understand the definition of a DTD.



The slide has a decorative graphic in the top-left corner consisting of overlapping colored squares (blue, red, yellow). The title "Definition of a DTD" is centered at the top in a blue font. Below the title is a bulleted list of two items. At the bottom left is the copyright notice "© Aptech Limited" and at the bottom right is the page number "XML Simplified/ Module 3/ 4 of 37".

Definition of a DTD

- It is a non XML document made up of element, attribute and entity declarations.
- It helps XML parsers to validate the XML document.

© Aptech Limited XML Simplified/ Module 3/ 4 of 37

Using slide 4, explain the definition of a DTD.

An XML parser is a software program that parses XML documents for its well-formedness. XML parsers are capable of validating XML files based on the document structure. Thus, to define the document structure, you need Document Type Definition (DTD).

DTD is a non-XML document made up of element, attribute, and entity declarations. The elements, attributes, and entities defined belong to the XML document(s), the DTD is defined for.

The DTD helps XML parsers to validate the XML document. It also helps authors to set default values for element attributes. The document's tree structure can be determined based on the DTD.

In other words, DTD defines the building blocks of an XML document. It defines the elements and attributes that forms the document structure for an XML document.

In-Class Question:

After you finish explaining the definition of DTD, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What are the main building blocks of XML documents?

Answer:

- Elements
- Attributes
- Entities
- CDATA and PCDATA

Slide 5

Let us understand the need for DTD.



Need for a DTD

- XML allows a user to define his/her own tag.
- Standardization of elements and attributes was needed.
- A DTD can define all the possible combinations and sequences for elements.

© Aptech Limited XML Simplified/Module 3/ 5 of 37

Using slide 5, explain the need for DTD.

The fact that XML allowed its user to define his/her own tag was one of its biggest advantages. However, it also led to a genuine problem. People working on parts of the same XML document sometimes used similar tags for different purposes and at other times, used different tags for the same purpose. At times, same elements possessed or exhibited different attributes. Some sort of standardization of elements and attributes was needed.

That is how DTDs came into picture.

A DTD can define all the possible combinations and sequences for elements to be used in an XML document along with their attributes and their acceptable values.

Slide 6

Let us understand the objectives of lesson 2.



Lesson 2 – Working with DTDs

In this second lesson, **Working with DTDs**, you will learn to:

- Describe the structure of a DTD.
- Explain how to create a simple DTD.
- Describe what is meant by document type declarations.

© Aptech Limited XML Simplified/Module 3/ 6 of 37

Using slide 6, explain the topics covered in the lesson 2.

Slide 7

Let us understand the structure of DTD.

Structure of DTD

- Element Declarations
- Attribute Declarations
- Entity Declarations

@ Aptech Limited XML Simplified/ Module 3/ 7 of 37

Using slide 7, explain the structure of DTD.

Explain that the DTD document contains element, attribute, and entity.

Each element declaration specifies the name of the element, and the content which that element can contain. Each attribute declaration specifies the element that owns the attribute, the attribute name, its type, and its default value (if any). Each entity declaration specifies the name of the entity and either its value or location of its value.

These groups of element, attribute, and entity declarations that form the three blocks of a DTD respectively can be declared in any sequence.

Tips:

A DTD can be declared either inline, that is, inside an XML document, or as an external reference.

Slides 8 to 10

Let us understand how to create internal DTDs.

Creating Internal DTDs 1-3

- Declare all the possible elements
- Specify the permissible element children, if any
- Set the order in which elements must appear
- Declare all the possible element attributes
- Set the attribute data types and values
- Declare all the possible entities

@ Aptech Limited

XML Simplified/Module 3/ 8 of 37

Creating Internal DTDs 2-3

Syntax

```
<!ELEMENT element-name (element-content)>
...
<!ATTLIST element-name attribute-name attribute-
type default-value>
...
<!ENTITY entity-name "entity-value">
...
```

@ Aptech Limited

XML Simplified/Module 3/ 9 of 37



Creating Internal DTDs 3-3

Code Snippet

```
<!ELEMENT Mobile (Company, Model, Price,
Accessories)>
<!ELEMENT Company (#PCDATA)>
<!ELEMENT Model (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
<!ELEMENT Accessories (#PCDATA)>
<!ATTLIST Model Type CDATA "Camera">
<!ENTITY HP "Head Phones">
<!ENTITY CH "Charger">
<!ENTITY SK "Starters Kit">
```

@ Aptech Limited

XML Simplified/Module 3/ 10 of 37

Using slides 8 to 10, explain how to create internal DTDs.

Mention that creating an internal DTD involves six step process.

Step 1 - Declare all the possible elements

In the DTD, XML elements are declared with an element declaration. An element declaration has the following syntax:

```
<!ELEMENT element-name (element-content)>
```

Example:

```
<!ELEMENT Mobile>
```

Step 2 - Specify the permissible element children, if any

Elements with one or more children are defined with the name of the children elements inside the parentheses:

```
<!ELEMENT element-name (child-element-name)>
```

Or

```
<!ELEMENT element-name (child-element-name, child-element-name, . . . . .)>
```

Example:

```
<!ELEMENT Mobile (Company, Model, Price, Accessories)>
```

Step 3 - Set the order in which elements must appear

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document.

Example:

```
<!ELEMENT Mobile (Company, Model, Price, Accessories)>
<!ELEMENT Company (#PCDATA)>
<!ELEMENT Model (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
<!ELEMENT Accessories (#PCDATA)>
```

Tips:

In DTD declaration, the children must be declared. The children can also have children which must also be specified in the DTD declaration.

Step 4 - Declare all the possible element attributes

In the DTD, XML element attributes are declared with an ATTLIST declaration. An attribute declaration has the following syntax:

```
<!ATTLIST element-name attribute-name>
```

Example:

```
<!ATTLIST Model Type
```

Step 5 - Set the attribute data types and values

The ATTLIST declaration defines the element which can have the attribute, the name of the attribute, the type of the attribute, and the default attribute value.

Example:

```
<!ATTLIST Model Type CDATA "Camera">
```

Step 6 - Declare all the possible entities

Entities as variables used to define shortcuts to common text appearing in the XML document. It has the following syntax:

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

Example:

```
<!ENTITY HP "Head Phones">
<!ENTITY CH "Charger">
<!ENTITY SK "Starters Kit">
```

Slides 11 and 12

Let us understand the DOCTYPE declaration.

- It specifies the name of the DTD and either its content or location.
- It begins with `<!DOCTYPE` and ends with a `>`.

```

10  <!DOCTYPE ... >
11  <!ELEMENT ...>
12  <!ATTLIST ... >
13  <!ENTITY ... >
14  </XmlFile>

```

© Aptech Limited XML Simplified/Module 3/ 11 of 37

Syntax

```

<!DOCTYPE name_of_root_element [ internal DTD
subset ]>
or
<!DOCTYPE name_of_root_element SYSTEM "URL of the
external DTD subset" >

```

© Aptech Limited XML Simplified/Module 3/ 12 of 37

Using slides 11 and 12, explain the DOCTYPE declarations.

Placed in the XML document's prolog, a DOCTYPE declaration begins with <!DOCTYPE and ends with a >. In between is the name of the root element, followed either by a pair of square brackets containing the DTD itself or by the SYSTEM keyword and a URL specifying where the DTD can be found.

Without the document type declaration, a DTD is nothing but plain text.

Syntax:

```
<!DOCTYPE name_of_root_element [ internal DTD subset ]>
```

Or

```
<!DOCTYPE name_of_root_element SYSTEM "URL of the external DTD subset" >
```

Slides 13 and 14

Let us understand the types of DTDs.

The slide has a title 'Types of DTDs 1-2' with a small graphic of overlapping squares (red, blue, yellow) to its left. Below the title, there are two main bullet points:

- DTDs can be classified as Internal or External.
- **Internal DTDs**
 - It consists of the DTD name followed by the DTD enclosed in square brackets.

A code snippet titled 'Internal DTD' is shown in a box:

```
14 ...
15 <!DOCTYPE Mobile SYSTEM "mobile.dtd">
16 ...
```

At the bottom left is the copyright notice '© Aptech Limited' and at the bottom right is 'XML Simplified/ Module 3/ 13 of 37'.



Types of DTDs 2-2

- **External DTDs**
- It consists of the DTDs name followed by the SYSTEM keyword followed by the address

```

External DTD Reference
<!DOCTYPE Mobile [
<ELEMENT Mobile (Company, Model, Price, Accessories)>
<ELEMENT Company (#PCDATA)>
<ELEMENT Model (#PCDATA)>
<ELEMENT Price (#PCDATA)>
<ELEMENT Accessories (#PCDATA)>
<ATTLIST Model Type #CDATA "Camera">
<ENTITY NP "#Head Phones">
<ENTITY OF "#Charger">
<ENTITY SK "#Starters Kit">
]>
  
```

@ Aptech Limited XML Simplified/ Module 3/ 14 of 37

Using slides 13 and 14, explain the types of DTDs.

DOCTYPE declarations have two basic forms of syntax, depending on the DTD used.

DTDs can be classified as Internal or External DTDs.

If the DTD is declared inside the XML file, it should be wrapped in a `DOCTYPE` definition.

The `DOCTYPE` definition syntax is as follows:

```
<!DOCTYPE root-element [element-declarations]>
```

If the DTD is declared in an external file, it should be wrapped in a `DOCTYPE` definition with the following syntax:

```
<!DOCTYPE root-element SYSTEM "filename">
```

Slide 15

Let us understand the objectives of lesson 3.



Lesson 3 – Valid XML Documents

In this third lesson, **Valid XML Documents**, you will learn to:

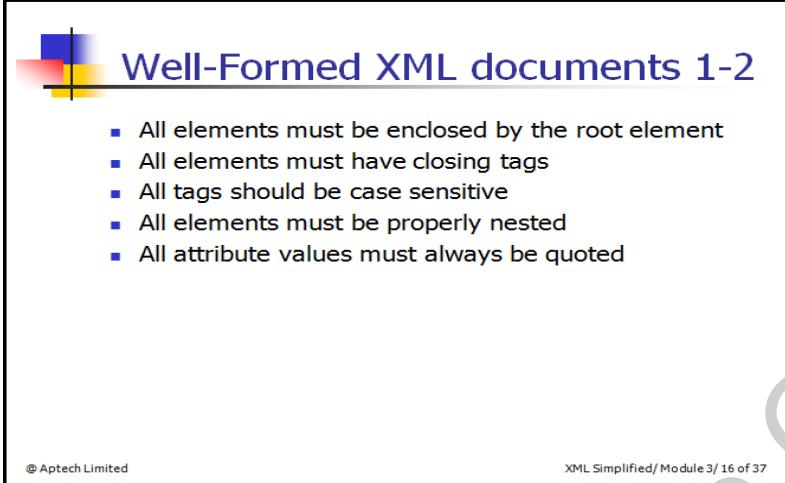
- Define document validity.
- Describe in brief how to test for document validity.

@ Aptech Limited XML Simplified/ Module 3/ 15 of 37

Using slide 15, explain the topics covered in the lesson 3.

Slides 16 and 17

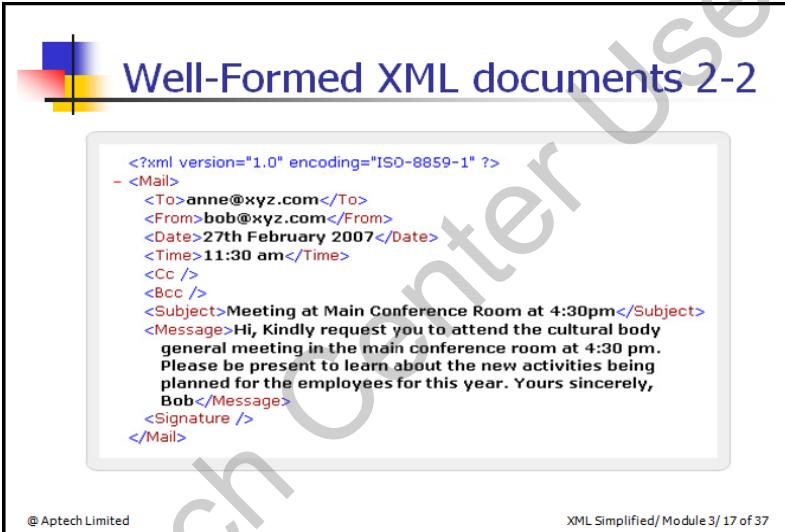
Let us understand the well-formed XML document.



Well-Formed XML documents 1-2

- All elements must be enclosed by the root element
- All elements must have closing tags
- All tags should be case sensitive
- All elements must be properly nested
- All attribute values must always be quoted

© Aptech Limited XML Simplified/ Module 3/ 16 of 37



Well-Formed XML documents 2-2

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <Mail>
  <To>anne@xyz.com</To>
  <From>bob@xyz.com</From>
  <Date>27th February 2007</Date>
  <Time>11:30 am</Time>
  <Cc />
  <Bcc />
  <Subject>Meeting at Main Conference Room at 4:30pm</Subject>
  <Message>Hi, Kindly request you to attend the cultural body
  general meeting in the main conference room at 4:30 pm.
  Please be present to learn about the new activities being
  planned for the employees for this year. Yours sincerely,
  Bob</Message>
  <Signature />
</Mail>
```

© Aptech Limited XML Simplified/ Module 3/ 17 of 37

Using slides 16 and 17, explain the well-formed XML document.

For an XML document to execute properly, it should be well-formed. A well-formed XML document adheres to the basic XML syntax rules. The World Wide Web Consortium in its specifications, states that XML documents with errors should not be processed by any program.

Explain the basic XML syntax rules and example of well-formed XML document provided on the slides.

Slides 18 and 19

Let us understand the valid XML document.



Valid XML documents 1-2

- A valid XML document is a well-formed XML document that adheres to its DTD.
- The validity of an XML document is determined by checking it against its DTD.

@ Aptech Limited XML Simplified/ Module 3/ 18 of 37



Valid XML documents 2-2

Code Snippet

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Mail [
<!ELEMENT Mail (To, From, Date, Time, Cc, Bcc, Subject, Message, Signature)
>>
<!ELEMENT To (#PCDATA)>
<!ELEMENT From (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Time (#PCDATA)>
<!ELEMENT Cc (#PCDATA)>
<!ELEMENT Bcc (#PCDATA)>
<!ELEMENT Subject (#PCDATA)>
<!ELEMENT Message (#PCDATA)>
<!ELEMENT Signature (#PCDATA)>
]>
</Mail>
<To> aman@xyz.com </To>
<From> bob@xyz.com </From>
<Date> 27th February 2007 </Date>
<Time> 11:00 am </Time>
<Cc> </Cc>
<Bcc> </Bcc>
<Subject> Meeting at Main Conference Room at 4:30pm </Subject>
<Message> Hi, Kindly request you to attend the cultural body general meeting in the main conference room at 4:30 pm. Please be present to learn about the new activities being planned for the employees for this year. Yours sincerely, Bob
</Message>
<Signature> </Signature>
</Mail>
```

@ Aptech Limited XML Simplified/ Module 3/ 19 of 37

Using slides 18 and 19, explain the valid XML document.

A valid XML document is a well-formed XML document that adheres to its DTD. The validity of an XML document is determined by checking it against its DTD. Once it has been confirmed that the components used in the XML document adhere to the declarations in the DTD, a well-formed XML document can be termed as a valid XML document.

Validity of XML documents plays an important role in all XML applications. Be it creating modular parts of the code, or data interchange, or processing of transferred data, or database access, and so forth, validity of the XML document is a recommended feature.

Explain the code on slide 19 which demonstrates a valid XML document.

Slides 20 and 21

Let us understand how to test XML for validity.

Testing XML for Validity 1-2

- It can be determined by using a validating parser such as Microsoft XML Core Services (MSXML) 6.0 Parser.
- MSXML enables the Internet Explorer (IE) browser to validate the code.
- The first image displays the validation result of a valid mobile.xml file.
- The second image displays the validation result after the removal of the signature element.

© Aptech Limited XML Simplified/Module 3/ 20 of 37

Testing XML for Validity 2-2

The slide shows two screenshots of Microsoft Internet Explorer validating XML files.

Left Screenshot: Shows the validation of a valid XML file (mobile.xml). A message box says "Validation Successful. File:///D:/ACT->XML by Example/mobile.xml".

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Mail (View Source for full doctype...)>
- <Mail>
  <To>anne@xyz.com</To>
  <From>bob@xyz.com</From>
  <Date>27th February 2007</Date>
  <Time>11:30 am</Time>
  <Cc />
  <Bcc />
  <Subject>Meetin
  4:30pm</Subject>
  <Message>Hi, Kui
  cultural body
  conference ro
  present to learn about the new activities
  being planned for the employees for this
  year. Yours sincerely, Bob</Message>
  <Signature />
</Mail>
```

Right Screenshot: Shows the validation of an XML file where the "Signature" element has been removed. An error message box says "Parse Error at line 24 in file:///D:/ACT->XML by Example/mobile.xml Source: <Signature>". Below it, the XML code is shown again.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Mail (View Source for full doctype...)>
- <Mail>
  <To>anne@xyz.com</To>
  <From>bob@xyz.com</From>
  <Date>27th February 2007</Date>
  <Time>11:30 am</Time>
  <Cc />
  <Bcc />
  <Subject>Meetin
  4:30pm</Subject>
  <Message>Hi, Kindly request you to attend the
  cultural body general meeting in the main
  conference room at 4:30 pm. Please be
  present to learn about the new activities
  being planned for the employees for this
  year. Yours sincerely, Bob</Message>
  <Signature />
</Mail>
```

© Aptech Limited XML Simplified/Module 3/ 21 of 37

Using slides 20 and 21, explain how to test XML for validity.

Validity being a desirable trait, it is necessary to be able to validate XML documents after their creation. Validity of XML documents can be determined by using a validating parser such as MSXML 6.0 Parser.

MSXML enables the Internet Explorer (IE) browser to validate the code. Once the code is displayed in IE, right-click the code to display the context menu. The menu provides the option of validating the code. On selection, IE internally validates the code against the code's DTD.

The first figure on slide 21 displays the validation result of a valid mobile.xml file.

The second figure on slide 21 displays the validation result after the removal of the signature element from the document's DTD.

Slide 22

Let us understand the objectives of lesson 4.



Lesson 4 - Declarations

In this last lesson, **Declarations**, you will learn to:

- Explain how to declare elements.
- Explain how to declare attributes.
- Describe entity declaration in a DTD.

© Aptech Limited XML Simplified/ Module 3/ 22 of 37

Using slide 22, explain the topics covered in the lesson 4.

Slides 23 and 24

Let us understand how to declare elements.



Declaring Elements 1-2

- XML elements are declared with an element declaration.

Syntax

```
<!ELEMENT element-name element-rule>
```

where,
ELEMENT is the keyword,
element-name is the name of the element,
element-rule can be one of the following: No Content, Only
Parsed Character Data, Any Contents, Children, Only One
Occurrence, Minimum One Occurrence, Zero or More
Occurrences, Zero or One Occurrence, Either/Or Content or
Mixed Content

© Aptech Limited XML Simplified/ Module 3/ 23 of 37



Declaring Elements 2-2

```

15  <!ELEMENT Mobile ( Company, Model, Price, Accessories)>
16  <!ELEMENT Company (#PCDATA)>
17  <!ELEMENT Model (#PCDATA)>
18  <!ELEMENT Price (#PCDATA)>
19  <!ELEMENT Accessories (#PCDATA)>
20  <!ATTLIST Model Type #CDATA "Camera">
21  <!ENTITY HP "Head Phones">
22  <!ENTITY CH "Charger">
23  <!ENTITY SK "Starters Kit">

```

@ Aptech Limited  

XML Simplified/ Module 3/ 24 of 37

Using slides 23 and 24, explain how to declare elements.

In the DTD, XML elements are declared with an element declaration. Explain the syntax for declaring elements.

Some of the element-rule options for the element declaration are as follows:

➤ **Empty Elements**

Empty elements are declared with the category keyword **EMPTY**.

```
<!ELEMENT element-name EMPTY>
```

Example:

```
<!ELEMENT br EMPTY>
```

XML example:

➤ **Elements with Parsed Character Data**

Elements with only parsed character data are declared with #PCDATA inside parentheses.

```
<!ELEMENT element-name (#PCDATA) >
```

Example:

```
<!ELEMENT from (#PCDATA) >
```

➤ Elements with any Contents

Elements declared with the category keyword **ANY**, can contain any combination of parsable data.

```
<!ELEMENT element-name ANY>
```

Example:

```
<!ELEMENT note ANY>
```

➤ Elements with Children (sequences)

Elements with one or more children are declared with the name of the children elements inside parentheses.

```
<!ELEMENT element-name (child1)>
```

Or

```
<!ELEMENT element-name (child1, child2, ...)>
```

Example:

```
<!ELEMENT message (to, from, subject, body)>
```

➤ Declaring Only One Occurrence of an Element

```
<!ELEMENT element-name (child-name)>
```

Example:

```
<!ELEMENT message (subject)>
```

The example declares that the child element "subject" must occur once, and only once inside the "message" element.

➤ Declaring Minimum One Occurrence of an Element

```
<!ELEMENT element-name (child-name+)>
```

Example:

```
<!ELEMENT message (subject+)>
```

The + sign in the example declares that the child element "subject" must occur one or more times inside the "message" element.

➤ Declaring Zero or More Occurrences of an Element

```
<!ELEMENT element-name (child-name*)>
```

Example:

```
<!ELEMENT message (subject*)>
```

The * sign in the example declares that the child element "subject" can occur zero or more times inside the "message" element.

➤ Declaring Zero or One Occurrence of an Element

```
<!ELEMENT element-name (child-name?)>
```

Example:

```
<!ELEMENT note (message?)>
```

The ? sign in the example declares that the child element "message" can occur zero or one time inside the "note" element.

➤ Declaring either/or Content

Example:

```
<!ELEMENT message (to, from, subject, (header|body))>
```

The example declares that the "message" element must contain a "to" element, a "from" element, a "subject" element, and either a "header" or a "body" element.

➤ Declaring Mixed Content

Example:

```
<!ELEMENT message (#PCDATA|to|from|header|body)*>
```

The example declares that the "message" element can contain zero or more occurrences of parsed character data, "to", "from", "header", or "body" elements.

Tips:

When the element content is a text string or numeric value, the element is designated as #PCDATA. Under DTD, there is no difference between numeric type data and text type data.

Slides 25 and 26

Let us understand how to declare attributes.

Declaring Attributes 1-2

Syntax

```
<!ATTLIST element-name attribute-name attribute-type default-value>
```

where,

- element-name is the element the attribute belongs
- attribute-name is the name of the attribute
- attribute-type is type of data the attribute can accept
- default-value is the default value for the attribute

© Aptech Limited XML Simplified/Module 3/25 of 37

Declaring Attributes 2-2

Value	Description
PCDATA	Parsed character data
CDATA	Character data
(en1 en2 ...)	Enumerated list
ID	A unique id
IDREF	Id of another element
IDREFS	List of other ids
NMTOKEN	Valid XML name
NMTOKENS	List of valid XML names
ENTITY	An entity
ENTITIES	List of entities
NOTATION	Name of a notation
xml:	Predefined xml value

© Aptech Limited XML Simplified/Module 3/26 of 37

Using slides 25 and 26, explain how to declare attributes.

Explain the different attribute types that can be declared for the attribute declaration.

Slides 27 to 29

Let us understand how to specify the attribute values.

Specifying Attribute Values 1-3

Value	Description
value	Default value
#REQUIRED	Value must be included
#IMPLIED	Value does not have to be included
#FIXED	Value is fixed
en1 en2 ...	Listed enumerated values

© Aptech Limited XML Simplified/Module 3/ 27 of 37

Specifying Attribute Values 2-3

Syntax

```
>#IMPLIED
<!ATTLIST element-name attribute-name attribute-
type #IMPLIED>

>#REQUIRED
<!ATTLIST element-name attribute-name attribute-
type #REQUIRED>

>#FIXED
<!ATTLIST element-name attribute-name attribute-
type #FIXED "value">

>Enumerated Attribute Values
<!ATTLIST element-name attribute-name
(en1|en2|...) default-value>
<!ATTLIST payment type (check|cash) "cash">
```

© Aptech Limited XML Simplified/Module 3/ 28 of 37

Specifying Attribute Values 3-3

Code Snippet

```
>Default Value
<!ATTLIST Model Type CDATA "Camera">

>#IMPLIED
<!ATTLIST Model Type CDATA "Camera" #IMPLIED>

>#REQUIRED
<!ATTLIST Model Type CDATA #REQUIRED>

>#FIXED
<!ATTLIST Model Type CDATA #FIXED "Camera">

>Enumerated Attribute Values
<!ATTLIST Model Type (Camera|Bluetooth) "Camera">
```

© Aptech Limited XML Simplified/Module 3/ 29 of 37

Using slides 27 to 29, explain how to specify the default values for the attribute.

The attribute-value in a DTD declaration can have default values.

The explanation for the attribute-value is as follows:

Default – Specifying a default value for an attribute value ensures that the attribute will be assigned a default value, even if the author of XML document does not assign the value.

Syntax:

```
<!ATTLIST element-name attribute-name CDATA "default-value">
```

DTD example:

```
<!ATTLIST payment type CDATA "check">
```

XML example:

```
<payment type="check">
```

Implied – An implied attribute is used when you don't want to force the author to include an attribute.

Syntax:

```
<!ATTLIST element-name attribute-name attribute-type #IMPLIED>
```

DTD example:

```
<!ATTLIST contact fax CDATA #IMPLIED>
```

XML example:

```
<contact fax="555-667788">
```

Required - Use a required attribute to force the attribute to be present.

Syntax:

```
<!ATTLIST element-name attribute_name attribute-type #REQUIRED>
```

DTD example:

```
<!ATTLIST person number CDATA #REQUIRED>
```

XML example:

```
<person number="5677">
```

Fixed – Does not allow the author to change it. If an author includes another value, the XML parser will return an error.

Syntax:

```
<!ATTLIST element-name attribute-name attribute-type #FIXED "value">
```

DTD example:

```
<!ATTLIST sender company CDATA #FIXED "Microsoft">
```

XML example:

```
<sender company="Microsoft">
```

Enumerated - When the attribute values needs to be one of a fixed set of values.

Syntax:

```
<!ATTLIST element-name attribute-name (eval|eval|..) default-value>
```

DTD example:

```
<!ATTLIST payment type (check|cash) "cash">
```

XML example:

```
<payment type="check">
or
<payment type="cash">
```

Tips:

Some of the problems with attributes are:

- Attributes cannot contain multiple values
- Attributes are not easily expandable
- Attributes cannot describe structures
- Attributes are more difficult to manipulate by program code
- Attribute values are not easy to test against a DTD

Slides 30 and 31

Let us understand the entities in DTD.

Entities in DTD 1-2

Syntax

```
>Entity declaration:  
<!ENTITY entity-name "entity-value">  
  
>Entity Reference:  
&entity-name;
```

© Aptech Limited XML Simplified/ Module 3/ 30 of 37

Entities in DTD 2-2

Code Snippet

```
<!DOCTYPE Mobile [  
  <ELEMENT Mobile (Company, Model, Price, Accessories)>  
  <ELEMENT Company (#PCDATA)>  
  <ELEMENT Model (#PCDATA)>  
  <ELEMENT Price (#PCDATA)>  
  <ELEMENT Accessories (#PCDATA)>  
  <!ELEMENT Model Type CDATA "Camera">  
  <!ENTITY HP "Head Phones">  
  <!ENTITY CH "Charger">  
  <!ENTITY SK "Starters Kit">  
>]  
<Mobile>  
  <Company> Nokia </Company>  
  <Model Type="Camera"> 6600 </Model>  
  <Price> 9999 </Price>  
  <Accessories> &HP; , &CH; and a &SK; </Accessories>  
</Mobile>
```

© Aptech Limited XML Simplified/ Module 3/ 31 of 37

Using slides 30 and 31, explain the entities in DTD.

An entity is a placeholder that consists of a name and a value. It is declared once and then repeatedly used throughout the document in place of its value.

Entities can be categorized as parameter entities and general entities.

When creating a parameter entity, you need to insert a percentage sign (%) between ENTITY and the name of the entity.

Syntax:

```
<!ENTITY % name definition>
```

General entities are further classified as:

- Character
- Mixed Content
- Unparsed

Character entities are entities that have a single character as their replacement value. Character entities are further classified as:

- Pre-defined
- Numbered
- Name

XML has a set of pre-defined entities to facilitate the use of characters such as <, >, &, ", ', since these are used by XML in its tags.

The **Pre-defined** entities and their references are shown in table 3.6.

Entity Name	Value
amp	&
apos	'
gt	>
lt	<
quot	"

Numbered entities are entities which use a character reference number, for a character from the Unicode character set, in place of the character itself. These entity references have a special syntax. Instead of the "&", "&#" is used. The reference number is used as the name, followed by a semicolon. Thus, a numbered entity reference would look like this: õ.

Name entities are the same as **Numbered** entities, except that the numbers are replaced by a descriptive name for the character.

Mixed content entities are entities that contain either text or markup language text as their replacement value. For example, the entity test could also have "trial" or

"<trial>Hi!</trial>" as its replacement value. The text "<trial>Hi!</trial>" will be replaced with the entity reference and will act as an element trial.

Unparsed entities are entities that have data that is not to be parsed. The replacement value might be an image or a file or something else. The syntax for an unparsed entity is as follows:

```
<!DOCTYPE name [  
  !ENTITY entity-name SYSTEM "entity-location" NDATA Notation  
  Identifier>  
>
```

Syntax to declare and reference entity is as follows:

Entity Declaration:

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

Entity Reference:

```
&entity-name;
```

Slides 32 to 35

Let us understand the different kinds of entity declarations.

Kinds of Entity Declarations 1-4

Internal Entity Declaration

- The entity value is explicitly mentioned in the entity declaration.

Syntax

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

© Aptech Limited XML Simplified/ Module 3/ 32 of 37

Kinds of Entity Declarations 2-4

Code Snippet

```
<!DOCTYPE Mobile [  
<!ELEMENT Mobile (Company, Model, Price, Accessories)>  
<!ELEMENT Company (#PCDATA)>  
<!ELEMENT Model (#PCDATA)>  
<!ELEMENT Price (#PCDATA)>  
<!ELEMENT Accessories (#PCDATA)>  
<!ATTLIST Model Type CDATA "Camera">  
<!ENTITY HP "Head Phones">  
<!ENTITY CP "Charger">  
<!ENTITY SK "Starters Kit">  
>  
<Mobile>  
<Company> Nokia </Company>  
<Model Type="Camera"> 6600 </Model>  
<Price> 9999 </Price>  
<Accessories> &HP; , &CP; and a &SK; </Accessories>  
</Mobile>
```

© Aptech Limited XML Simplified/ Module 3/ 33 of 37

Kinds of Entity Declarations 3-4

External Entity Declaration

- A link or path to the entity value is mentioned in place of the entity value

© Aptech Limited XML Simplified/ Module 3/ 34 of 37

Kinds of Entity Declarations 4-4

Syntax

```
<!ENTITY entity-name SYSTEM "URI/URL">
```

```
<!DOCTYPE Mobile [
<ELEMENT Mobile (Company, Model, Price, Accessories)>
<ELEMENT Company (#PCDATA)>
<ELEMENT Model (#PCDATA)>
<ELEMENT Price (#PCDATA)>
<ELEMENT Accessories (#PCDATA)>
<ELEMENT Headphones (#PCDATA)>
<ELEMENT Camera (#PCDATA)>
<ENTITY HP SYSTEM "&hp.txt">
<ENTITY CH SYSTEM "&ch.txt">
<ENTITY SK SYSTEM "&sk.txt">
]>
<Mobile>
<Company> Nokia </Company>
<Model Type="Camera"> 6600 </Model>
<Price> 9999 </Price>
<Accessories> &HP; , &CH; and a &SK; </Accessories>
</Mobile>
&hp.txt
Head Phones
&ch.txt
Camera
&sk.txt
Starters Kit
```

© Aptech Limited XML Simplified/ Module 3/ 35 of 37

Using slides 32 to 35, explain the kind of entity declarations.

Entity references are used extensively in XML. That leads to Internal and External entity declarations, the two basic kinds of entity declarations used in XML.

In internal entity declarations, the entity value is explicitly mentioned in the entity declaration.

Explain the syntax of internal entities using slide 32. Also, explain the code on slide 33 which demonstrates the properties of internal entity declaration.

In external entity declaration, a link or path to the entity value is mentioned in place of the entity value with the help of the **SYSTEM** keyword. External entity declarations are in a way similar to declaring external DTDs.

Explain the syntax of internal entities using slide 34. Also, explain the code on slide 35 which demonstrates the properties of external entity declaration.

Slides 36 and 37

Let us summarize the session.



Summary 1-2

- **Document Type Definition**
 - A DTD is a non XML document made up of element, attribute and entity declarations.

- **Working with DTDs**
 - The DTD structure is composed of element declarations, attribute declarations, and entity declarations.
 - A document type declaration declares that the XML file in which it is present adheres to a certain DTD.

© Aptech Limited

XML Simplified/ Module 3/ 36 of 37



Summary 2-2

- **Valid XML Documents**
 - A well-formed XML document adheres to the basic XML syntax rules.
 - A valid XML document is a well-formed XML document that adheres to its DTD.

- **Declarations**
 - XML elements are declared with an element declaration in the DTD.
 - An entity is a placeholder that consists of a name and a value.

© Aptech Limited

XML Simplified/ Module 3/ 37 of 37

Using slides 36 and 37, summarize the session. End the session, with a brief summary of what has been taught in the session.

3.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the XML Schema that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 4 - XML Schema

4.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

4.1.1 Objectives

By the end of this session, the learners will be able to:

- XML Schema
- Exploring XML Schemas
- Working with Complex Types
- Working with Simple Types

4.1.2 Teaching Skills

To teach this session, you should be well-verses with XML schema and its features. The topics such as structure of an XML document with the help of schema, different elements, and data types supported by an XML schema.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

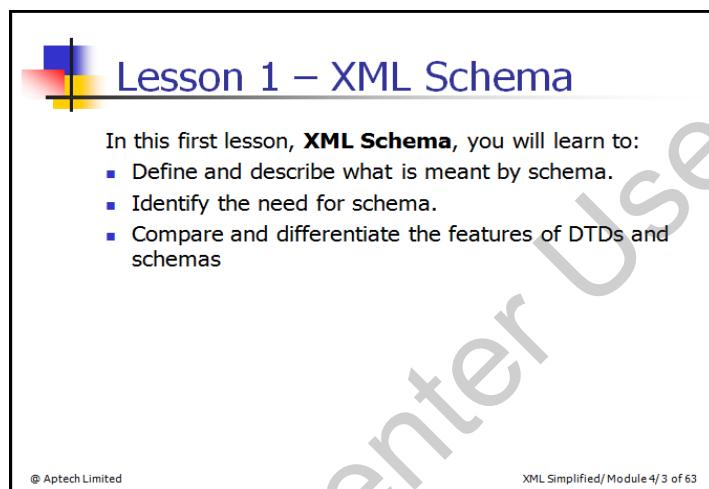
Then give the students the overview of the current session in the form of Session Overview. Show the students slide 2 of the presentation.

Tell the students that this session introduces XML schema and its features. They will learn how to define the structure of an XML document with the help of schema. They will also understand the different elements and data types supported by an XML schema.

4.2 In-Class Explanations

Slide 3

Let us understand the objectives of lesson 1.



The slide has a decorative graphic in the top-left corner consisting of overlapping colored squares (blue, red, yellow). The title "Lesson 1 – XML Schema" is centered at the top in a blue font. Below the title, a text block states: "In this first lesson, **XML Schema**, you will learn to:" followed by a bulleted list. At the bottom left is the copyright notice "© Aptech Limited" and at the bottom right is the page information "XML Simplified/Module 4/ 3 of 63".

Lesson 1 – XML Schema

In this first lesson, **XML Schema**, you will learn to:

- Define and describe what is meant by schema.
- Identify the need for schema.
- Compare and differentiate the features of DTDs and schemas

© Aptech Limited

XML Simplified/Module 4/ 3 of 63

Using slide 3, explain the topics covered in the lesson 1.

Slides 4 and 5

Let us understand the XML schema.

XML Schema 1-2

- It defines the valid building blocks of an XML document.
- XML Schema language is referred as XML Schema Definition (XSD).
- It describes the data that is marked up, and also specifies the arrangement of tags and text.
- The dictionary defines a schema as "An outline or a model".

Validating XML data

XML

```
<Book>
    <Title>Path To Paradise</Title>
    <Author>David White</Author>
    <Theme>Philosophy</Theme>
    <Publisher>ABC Publication</Publisher>
    <ISBN>11</ISBN>
    <Price>$12481</Price>
    <Edition>June 2000</Edition>
</Book>
```

Declare a Book element. Require that its components be Title, Author, Theme, Publisher, ISBN, Price and Edition. The Author, Theme, Publisher and will take alphanumeric values where as Price and ISBN will take the numeric values.

© Aptech Limited XML Simplified/Module 4/ 4 of 63

XML Schema 2-2

Code Snippet

```

<Book>
    <Title>Path To Paradise</Title>
    <Author>David White</Author>
    <Theme>Philosophy</Theme>
    <Publisher>ABC Publication</Publisher>
    <ISBN>11</ISBN>
    <Price>$12481</Price>
    <Edition>June 2000</Edition>
</Book>
```

© Aptech Limited XML Simplified/Module 4/ 5 of 63

Using slides 4 and 5, explain the XML schema.

Consider a scenario where requested to create an XML document to serve as a purchase order to be sent to Mr. Y at Company X, what kind of XML document would you create?

Following are three XML document examples, created by three different individuals.

- Ms. A has created very semantic element names.
- Mr. B has opted for rather abbreviated element names.
- Ms. C has created elements having a hierarchical structure.

Following figure shows the creation of purchase order document created by the three individuals:

Purchase order created by Ms. A:	Purchase Order Created by Ms. C:
<pre><PurchaseOrder> <Name>Jenny</Name> <Address>Tokyo</Address> <ProductName>television</ProductName> <NoItems>10</NoItems> </PurchaseOrder></pre>	<pre><PurchaseOrder> <PurchaserInformation> <Name>Jenny</Name> <Address>Tokyo</Address> <ContactNo>555-5555</ContactNo> </PurchaserInformation> <OrderInformation> <Product> <ProductNo>ID001</ProductNo> <ProductName>television</ProductName> <UnitPrice>128000</UnitPrice> </Product> <NoItems>10</NoItems> </OrderInformation> </PurchaseOrder></pre>
Purchase Order Created by Mr. B:	
<pre><a> Jenny <c>Tokyo</c> <d>television</d> <e>10</e> </pre>	

Any of the three examples can be considered to be proper XML documents. As long as the information required for a purchase order is included, likely any XML document you could create would be a valid XML document for the purpose.

What would change if you approached the task from Mr. Y's perspective?

Assume that Mr. Y uses the three XML documents mentioned, or your XML document, for order processing. Having received XML documents with different element names and hierarchical structures, Mr. Y would have to open each XML document in an editor, confirm whether all of the required information was present, and then process the purchase order. In this case, every purchase order would have to be processed by hand, and the entire system could never be automated.

What would happen if all XML documents sent had the same element names and hierarchical structure? With standard element names and structures, a system could be created to handle all incoming XML documents, and order processing could be automated, without Mr. Y having to verify the content of each individual document.

A "Schema" is what is required to allow the acceptance (or creation) of XML documents with a standardized element name and hierarchy structure.

Let's take another look at the task for creating an XML document to be used for a purchase order. Assume that Mr. Y sends to Ms. A, Mr. B, and Ms. C a schema document for purchase orders (XML document). Ms. A, Mr. B, and Ms. C then each create an XML document based on Mr. Y's schema. The element names and hierarchical structure of the XML documents, they send to Mr. Y, are completely identical.

Mr. Y can now use an XML parser to verify whether the documents have been created according to the schema, so there is no need to open each file and check element names and hierarchy structures. This reduces Mr. Y's workload significantly.

Thus, the use of schema for creating XML documents ensures that the hierarchical structure of the document is based on the defined schema.

The word ‘schema’ originated from a Greek word symbolizing form or shape. The dictionary meaning of schema is: "A diagrammatic representation; an outline or a model". Initially, the word ‘schema’ was only in the reach of the philosophers, till it entered the zone of computer science. In the context of software, a schema is generally understood to be a model used to describe the structure of a database. It defines internal structures such as tables, fields, and the relationship between them.

An XML Schema defines the valid building blocks of an XML document. It can be considered as a common vocabulary that different organizations can share to exchange documents. The XML Schema language is referred as XML Schema Definition (XSD).

Explain the figure on slide 4 which depicts the XML data validation. The previous knowledge of what a book is, what its attributes are, could be a kind of schema, and against which this 'BOOK' is compared would help to validate the attributes of the book.

Slides 6 and 7

Let us understand the XML schema objectives.



XML Schema Objectives 1-2

```

1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <!--
4      Document : books.xml
5      Author   : vincent
6      Description:
7          This document defines a schema for a library of books.
8  -->
9
10 <library>
11  <fiction>
12    <book>The Firm, John Grisham</book>
13    <book>Coma, Robin Cook</book>
14  </fiction>
15  <nonfiction>
16    <book>Freakonomics , Malcolm Gladwell</book>
17  </nonfiction>
18  <?latest editions only?>
19 </library>

```

© Aptech Limited XML Simplified / Module 4 / 7 of 63

Using slides 6 and 7, explain the XML schema objectives.

The objective of an XML Schema is to define the valid structure of an XML document, similar to a DTD. However, XML schemas are more powerful than DTD.

Explain the features of XML schema.

Additionally, you can mention that XML schemas will be used in most Web applications as a replacement for DTD, as they are richer in data types and provide more support for XML document structure. Also, XML schemas are written in XML.

Explain the figure on slide 7 that depicts the XML document based on schema of books.

Tips:

Some of the advantages of using XML schema are as follows:

- XML Schemas are written in XML.
- XML Schemas are extensible to additions.
- XML Schemas support data types.
- XML Schemas support namespaces.

In-Class Question:

After you finish explaining the XML schema, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



How is XML schema different from DTD?

Answer:

XML schema is written in XML, whereas DTD is written in plain text file.

Slides 8 and 9

Let us understand how to write an XML schema.

How to write an XML Schema? 1-2

XML File

- The XML document contains a single element, <Message>.

XSD File

- The file is saved with ".xsd" as the extension for storing schema documents

message.xsd

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

message.xml

```
<?xml version="1.0"?>
<Message>
  Hello World!
</Message>
```

Linking XML file with its schema

© Aptech Limited XML Simplified/Module 4/ 8 of 63

How to write an XML Schema? 2-2

Code Snippet

XML File: message.xml

```
<?xml version="1.0"?>
<Message>
  Hello World!
</Message>
```

XSD File: message.xsd

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

© Aptech Limited XML Simplified/Module 4/ 9 of 63

Using slides 8 and 9, explain how to write XML schema.

To create the schema for XML document, you need two files, they are as follows:

➤ XML File

Given XML document contains a single element, <Message>. A schema for this document has to declare the <Message> element.

➤ XSD File

For storing schema documents, the file is saved with .xsd as the extension. Schema documents are XML documents and can have DTDs, DOCTYPE declarations.

Explain the figure on slide 8 that depicts XSD file.

Explain the code on slide 9 which demonstrates the properties of XML schema.

The code explanation is as follows:

XSD file is a schema for <Message> elements. The file is stored as message.xsd. It is an XML document; therefore it has an XML declaration. This schema file may declare other elements too, including ones that are not present in this XML file, but it must at least declare the <Message> element.

Slides 10 to 12

Let us understand the features of XML schema.

What is an XML Schema? 1-3

- XML schemas allow Web applications to exchange XML data more robustly using the following range of new features:
 - Schemas support data types
 - Schemas are portable and efficient
 - Schemas secure data communication
 - Schemas are extensible
 - Schemas catch higher-level mistakes
 - Schemas support Namespace

Schemas support data types

- Support and ability to create the required datatypes has overcome the drawbacks of DTDs
- Easy to define and validate valid document content and data formats
- Easy to implement the restrictions on data

© Aptech Limited XML Simplified/Module 4/ 10 of 63

What is an XML Schema? 2-3

Schemas are portable and efficient

- Portable and efficient
- No need to learn any new language or any editor
- Similar XML parser can be used to parse the Schema files

Schemas secure data communication

- Sender can specify the data in a way that the receiver will understand

Schemas are extensible

- It is possible to reuse an existing schema to create another schema
- It is possible to create own data types derived from the standard types
- Support reference of multiple schemas in the same document

© Aptech Limited XML Simplified/Module 4/ 11 of 63



What is an XML Schema? 3-3

Schemas catch higher-level mistakes

- Catch higher-level mistakes that arise, such as a required field of information is missing or in a wrong format, or an element name is mis-spelled

Schemas support Namespace

- Support for XML Namespaces allows the programmer to validate documents that use markup from multiple namespaces
- Constructs can be re-used from schemas already defined in a different namespace

© Aptech Limited XML Simplified/Module 4/ 12 of 63

Using slides 10 to 12, explain the features of XML schema.

Slides 13 and 14

Let us understand the difference in DTDs and schemas.



Comparing DTDs with Schemas 1-2

Drawbacks of using DTDs are:

- DTDs are written in a non-XML syntax
- DTDs are not extensible
- DTDs do not support namespaces
- DTDs offer limited data typing

Sample External DTD File: program.dtd

```
<!ELEMENT program (comments, code)>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT code (#PCDATA)>
```

© Aptech Limited XML Simplified/Module 4/ 13 of 63



Comparing DTDs with Schemas 2-2

Sample XML File with a reference to dtd: program.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE program SYSTEM "program.dtd">
<program>
    <comments>
        This is a simple Java Program. It will display the
        message
        "Hello world!" on execution.
    </comments>
    <code>
        public static void main(String[] args)
        System.out.println("Hello World!"); // Display the
        string.
    </code>
</program>
```

© Aptech Limited XML Simplified/Module 4/ 14 of 63

Using slide 13, explain the difference in DTDs and schemas.

XML inherited the concept of DTDs from Standard Generalized Markup Language (SGML), which is an international standard for markup languages. DTDs are used to define content models, nesting of elements in a valid order, and provide limited support to data types and attributes.

Explain the drawbacks of using DTDs.

➤ **DTDs are written in a non-XML syntax**

DTDs do not use XML notation and therefore they are difficult to write and use.

➤ **DTDs are not extensible**

DTDs are not extensible. For example, if there is an Address DTD to catalog friends, and one wants to add a new section to the code for official contacts, then the entire DTD has to be rewritten.

➤ **DTDs do not support namespaces**

Namespaces can be used to introduce an element type into an XML document. However, a Namespace cannot be used to refer to an element or an entity declaration in the DTD. If a Namespace is used, then the DTD has to be modified to include any elements taken from the Namespace.

➤ **DTDs offer limited data typing**

DTDs can only express the data type of attributes in terms of explicit enumerations and a few coarse string formats. DTDs do not have a facility to describe numbers, dates, currency values, and so forth. Furthermore, DTDs do not have the ability to express the data type of character data in elements. For example, a <Zip> element can be defined to contain CDATA. However, the element cannot be constrained to just numerals when it uses a DTD.

Explain the code on slide 13 which demonstrates a sample external DTD file, program.dtd.

Explain the code on slide 14 which demonstrates a sample XML file with a reference to DTD program.xml.

Slides 15 to 17

Let us understand the advantages of XML schemas over DTD.



Advantages of XML Schemas over DTD 1-3

XML schema offers a range of new features:

- Richer data types
- Archetypes
- Attribute grouping
- Refinable archetypes

© Aptech Limited XML Simplified/ Module 4/ 15 of 63



Advantages of XML Schemas over DTD 2-3

Sample schema File: mail.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.abc.com"
xmlns="http://www.abc.com"
elementFormDefault="qualified">
<xs:element name="mail">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="header" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

© Aptech Limited XML Simplified/ Module 4/ 16 of 63



Advantages of XML Schemas over DTD 3-3

Sample XML File with a reference to schema: mail.xml

```
<?xml version="1.0"?>
<mail
  xmlns="http://www.abc.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.abc.com mail.xsd">

  <to>John</to>
  <from>Jordan</from>
  <header>Scheduler</header>
  <body>3rd March Monday, 7:30 PM: board meeting!</body>
</mail>
```

© Aptech Limited XML Simplified/ Module 4/ 17 of 63

Using slides 15 to 17, explain the advantages of XML schemas over DTD.

Explain the code on slide 16 which demonstrates a sample schema file, mail.xsd.

Explain the code on slide 17 which demonstrates a sample XML file with a reference to schema mail.xml.

Slide 18

Let us understand the objectives of lesson 2.



Lesson 2 – Exploring XML Schemas

Using slide 18, explain the topics covered in the lesson 2.

Slides 19 to 24

Let us understand the data types supported by schemas.



Data Types Supported by Schema 1-6

- XML Schema describes a number of built-in data types, which can be used to specify and validate the intended data type of the content.
- It also allows a user to create a user-defined data type by extending the built-in data types using facets.
- XML Schema recommendation defines two sorts of data types:
 - Built-in data types
 - User-derived data types

Built-in data types

- Available to all XML Schema authors, and should be implemented by a conforming processor.

User-derived data types

- Defined in individual schema instances, and are particular to that schema.

Data Types Supported by Schema 2-6

string	<ul style="list-style-type: none"> can contain characters, line feeds, carriage returns, and tab characters
boolean	<ul style="list-style-type: none"> legal values for boolean data type are true and false true can be replaced by the numeric value 1 and false can be replaced by the value 0
numeric	<ul style="list-style-type: none"> represents a numerical value includes numbers such as whole numbers, and real numbers
dateTime	<ul style="list-style-type: none"> represents a particular time on a given date, written as a string
binary	<ul style="list-style-type: none"> include graphic files, executable programs, or any other string of binary data
anyURI	<ul style="list-style-type: none"> represents a file name or location of the file

© Aptech Limited

XML Simplified/Module 4/20 of 63

Data Types Supported by Schema 3-6

Syntax: string

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

Syntax: boolean

```
<xs:attribute name="attribute_name" type="xs:boolean"/>
```

Syntax: numeric

```
<xs:element name="element_name" type="xs:numeric"/>
```

© Aptech Limited

XML Simplified/Module 4/21 of 63

Data Types Supported by Schema 4-6

Syntax: dateTime

```
<xs:element name="element_name" type="xs:dateTime"/>
```

Syntax: binary

```
<xs:element name="image_name" type="xs:hexBinary"/>
```

Syntax: anyURI

```
<xs:attribute name="image_name" type="xs:anyURI"/>
```

© Aptech Limited

XML Simplified/Module 4/22 of 63

Data Types Supported by Schema 5-6

Code Snippet: string

```
<xs:element name="Customer" type="xs:string"/>
<Customer>John Smith</Customer>
```

Code Snippet: boolean

```
<xs:attribute name="Disabled" type="xs:boolean"/>
<Status Disabled="true">OFF</Status>
```

Code Snippet: numeric

```
<xs:element name="Price" type="xs:numeric"/>
<Price>500</Price>
```

© Aptech Limited XML Simplified/Module 4/23 of 63

Data Types Supported by Schema 6-6

Code Snippet: dateTime

```
<xs:element name="BeginAt" type="xs:dateTime"/>
<start>2001-05-10T12:35:40</start>
```

Code Snippet: binary

```
<xs:element name="Logo" type="xs:hexBinary"/>
```

Code Snippet: anyURI

```
<xs:attribute name="flower" type="xs:anyURI"/>
<image
flower="http://www.creativepictures.com/gallery/flower.gif"
/>
```

© Aptech Limited XML Simplified/Module 4/24 of 63

Using slide 19, explain the data types supported by schemas.

XML Schema describes a number of built-in data types, which can be used to specify and validate the intended data type of the content. It also allows a user to create a user-defined data type by extending the built-in data types using facets.

Explain two types of data types in XML schemas.

- **Built-in** data types, are available to all XML Schema authors, and should be implemented by a conforming processor.
- **User-derived** data types, are defined in individual schema instances, and are particular to that schema (although, it is possible to import these definitions into other definitions).

Using slide 20, explain the types of built-in data types mentioned in the table. Also, using slides 21 and 22, explain the syntax of data types.

Using slides 23 and 24, explain the code snippets which demonstrates the use of data types.

Slides 25 to 27

Let us understand the additional data types.



Additional Data types 1-3

- Additional data types are derived from the basic built-in data types, which are called base type data types.
- The generated or derived data types, supported by the XML schema include:
 - integer
 - decimal
 - time

Integer

- Base type is numeric data type. Includes both positive and negative numbers. Used to specify a numeric value without a fractional component.

Decimal

- Represent exact fractional parts such as 3.26. Base type is numeric data type. Used to specify a numeric value.

Additional Data types 2-3

Additional Data types 3-3

Using slides 25 to 27, explain the additional data types.

Additional data types are derived from the basic built-in data types, which are called base type data types. Explain the generated or derived data types supported by the XML schema.

Using slides 26 and 27, explain the code and syntax for derived data types.

Slides 28 to 30

Let us understand the schema vocabulary.

Schema Vocabulary 1-3

- Creating a schema using XML schema vocabulary is like creating any other XML document using a specialized vocabulary.

Every XML Schema starts with the root element <schema>.

The code indicates that the elements and data types used in the schema are derived from the "http://www.w3.org/2001/XMLSchema" namespace and prefixed with xs.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema"
4      targetNamespace="http://www.abc.com"
5      xmlns="http://www.abc.com"
6      elementFormDefault="qualified">
7      <xss:element name="Mail">
8          <xss:complexType>
9              <xss:sequence>
10                 <xss:element name="To" type="xs:string"/>
11                 <xss:element name="From" type="xs:string"/>
12                 <xss:element name="Header" type="xs:string"/>
13                 <xss:element name="Body" type="xs:string"/>
14             </xss:sequence>
15         </xss:complexType>
16     </xss:element>
17 </xss:schema>

```

© Aptech Limited XML Simplified/ Module 4/ 28 of 63

Schema Vocabulary 2-3

```

1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema"
4      targetNamespace="http://www.abc.com"
5      xmlns="http://www.abc.com"
6      elementFormDefault="qualified">
7      <xss:element name="Mail">
8          <xss:complexType>
9              <xss:sequence>
10                 <xss:element name="To" type="xs:string"/>
11                 <xss:element name="From" type="xs:string"/>
12                 <xss:element name="Header" type="xs:string"/>
13                 <xss:element name="Body" type="xs:string"/>
14             </xss:sequence>
15         </xss:complexType>
16     </xss:element>
17 </xss:schema>

```

This line of code points to "http://www.abc.com" as the default namespace.

It indicates that elements used by the XML instance document which were declared in this schema needs to be qualified by the namespace.

© Aptech Limited XML Simplified/ Module 4/ 29 of 63

Schema Vocabulary 3-3

It indicates the default namespace declaration. This declaration informs the schema-validator that all the elements used in this XML document are declared in the default ("http://www.abc.com") namespace.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <Mail xmlns="http://www.abc.com"
4      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5      xsi:schemaLocation="http://www.abc.com/mail_schema.xsd">
6      <To>John</To>
7      <From>Jordan</From>
8      <Header>Scheduler</Header>
9      <Body>3rd March Monday, 7:30 PM: board meeting!</Body>
10 </Mail>

```

This schemaLocation attribute has two values. The first value identifies the namespace. The second value is the location of the XML schema where it is stored.

This is the instance namespace available for the XML schema.

© Aptech Limited XML Simplified/ Module 4/ 30 of 63

Using slides 28 to 30, explain the schema vocabulary.

Creating a schema using XML schema vocabulary is like creating any other XML document using a specialized vocabulary. To understand the XML Schema vocabulary and the elements, the example discusses an XML document that will be validated against a schema.

Explanation of the different parts is as follows:

- **xs : schema xmlns:xs=http://www.w3.org/2001/XMLSchema**
Every XML Schema starts with the root element <schema>. The code indicates that the elements and data types used in the schema are derived from the "http://www.w3.org/2001/XMLSchema" namespace and prefixed with xs.
- **targetNamespace=http://www.abc.com**
It specifies that the elements defined in an XML document that refers this schema, come from the "http://www.abc.com" namespace.
- **xmlns=http://www.abc.com**
This line of code points to "http://www.abc.com" as the default namespace.
- **elementFormDefault="qualified"**
It indicates that elements used by the XML instance document which were declared in this schema needs to be qualified by the namespace.
- **xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance**
This is the instance namespace available for the XML schema.
- **xsi:schemaLocation="http://www.abc.com mail.xsd">**
This schemaLocation attribute has two values. The first value identifies the namespace. The second value is the location of the XML schema where it is stored.

Slide 31

Let us understand the objectives of lesson 3.



Lesson 3 – Working with Complex Types

In this third lesson, **Working with Complex Types**, you will learn to:

- Describe complex type elements.
- Describe `minOccurs` and `maxOccurs`.
- Explain element content and mixed content.
- Describe how grouping can be done.

© Aptech Limited XML Simplified/Module 4/31 of 63

Using slide 31, explain the topics covered in the lesson 3.

Slides 32 to 35

Let us understand the complex types.



Complex Types 1-4

- A schema assigns a type to each element and attribute it declares.
- Elements with complex type may contain nested elements and have attributes.
- Only elements can contain complex types. Complex type elements have four variations. They are:
 - Empty Elements
 - Only Elements
 - Only Text
 - Mixed

© Aptech Limited XML Simplified/Module 4/32 of 63



Complex Types 2-4

Empty elements

- Empty elements optionally specify attributes types, but do not permit content.

```
<xs:element name="Books">
  <xs:complexType>
    <xs:attribute name="BookCode" type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

Only Elements

- These elements can only contain elements and do not contain attributes.

```
<xs:element name="Books">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ISBN" type="xs:string"/>
      <xs:element name="Price" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

© Aptech Limited XML Simplified/Module 4/33 of 63

Complex Types 3-4

Only Text

- These elements can only contain text and optionally may or may not have attributes.

```
<xs:complexType name="Books">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="BookCode" type="xs:positiveInteger"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

© Aptech Limited XML Simplified/Module 4/ 34 of 63

Complex Types 4-4

Mixed

- These are elements that can contain text content as well as sub-elements within the element. They may or may not have attributes.

```
<xs:element name="Books">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="BookName" type="xs:string"/>
      <xs:element name="ISBN" type="xs:positiveInteger"/>
      <xs:element name="Price" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

A complex element can be defined in two different ways:

- By directly naming the element
- By using the name and type attribute of the complex type

© Aptech Limited XML Simplified/Module 4/ 35 of 63

Using slides 32 to 35, explain the complex types.

A complex element is an XML element that contains other elements and/or attributes. A schema assigns a type to each element and attribute that it declares. Only elements can contain complex types.

There are four kinds of complex elements:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

Explain the empty complex type elements in detail.

Empty elements

Empty elements optionally specify attributes types, but do not permit content as shown in the following example:

```
<xs:element name="Books">
<xs:complexType>
<xs:attributename="BookCode"
type="xs:positiveInteger"/>
</xs:complexType>
</xs:element>
```

Only Elements

These elements can only contain elements and do not contain attributes as shown in the following example:

```
<xs:element name="Books">
<xs:complexType>
<xs:sequence>
<xs:element name="ISBN" type="xs:string"/>
<xs:element name="Price" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Only Text

These elements can only contain text and optionally may or may not have attributes as shown in the following example:

```
<xs:complexType name="Books">
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attribute name="BookCode" type="xs: positiveInteger"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
```

Mixed

These are elements that can contain text content as well as sub-elements within the element as shown in the following example:

```
<xs:element name="Books">
<xs:complexType mixed="true">
<xs:sequence>
<xs:element name="BookName" type="xs:string"/>
<xs:element name="ISBN" type="xs:positiveInteger"/>
<xs:element name="Price" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

They may or may not have attributes.

Slides 36 and 37

Let us understand how to define complex types.



Defining Complex Types 1-2

- By directly naming the element

Code Snippet

```
<xs:element name="Student">
<xs:complexType>
<xs:sequence>
<xs:element name="FirstName" type="xs:string"/>
<xs:element name="MiddleName" type="xs:string"/>
<xs:element name="LastName" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

© Aptech Limited XML Simplified/ Module 4/ 36 of 63



Defining Complex Types 2-2

- By using the name and type attribute of the complex type

Code Snippet

```
<xs:element name="Student" type="PersonInfo"/>
<xs:complexType name="personinfo">
<xs:sequence>
<xs:element name="FirstName" type="xs:string"/>
<xs:element name="LastName" type="xs:string"/>
</xs:sequence>
</xs:complexType>
```

© Aptech Limited XML Simplified/ Module 4/ 37 of 63

Using slides 36 and 37, explain how to define complex types.

A complex element can be defined in two different ways:

By directly naming the element

Explain the code on slide 36 which defines the properties of complex element.

The element **Student** can be declared by directly mentioning it in the schema. The complex type mentions that the XML document contains nested XML elements. The sequence element indicates that the child elements **FirstName**, **MiddleName**, and **LastName** appear in the same order as they are declared.

By using the name and type attribute of the complex type

Explain the code on slide 37 which define the properties of complex element by using the name and type attribute of the complex type.

The Student element can have a type such as PersonInfo that refers to the name of the complex type. Several elements can reuse this complex type by referring to the name PersonInfo in their complex type declarations.

Slides 38 to 40

Let us understand the minOccurs and maxOccurs.

minOccurs and maxOccurs 1-3

- **minOccurs**
 - It specify the minimum number of occurrences of the element in an XML document.
- **maxOccurs**
 - It specify the maximum number of occurrences of the element in an XML document.

© Aptech Limited XML Simplified/Module 4/ 38 of 63

minOccurs and maxOccurs 2-3

Code Snippet

```
...
<xs:element name= "Books">
<xs:complexType>
<xs:sequence>
<xs:element name="ISBN" type="xs:string"/>
<xs:element name="Quantity" type="xs:string"
maxOccurs="100" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:element>
...
```

© Aptech Limited XML Simplified/Module 4/ 39 of 63



minOccurs and maxOccurs 3-3

minOccur	MaxOccur	Number of times an element can occur
0	1	0 or 1
1	1	1
0	*	Infinite
1	*	At least once
>0	*	At least minOccurs times
>maxOccurs	>0	0
Any value	<minOccurs	0

© Aptech Limited XML Simplified/Module 4/ 40 of 63

Using slides 38 to 40, explain the `minOccurs` and `maxOccurs`.

When working with DTDs, we used the markers *, ?, and + to indicate the number of times a particular child element could be used as content for an element. Similarly, XML schema allows specifying a minimum and maximum number of times an element can occur.

`minOccurs` specify the minimum number of occurrences of the element in an XML document. The default value for the `minOccurs` attribute is 1. If an element has a `minOccurs` value of 0, it is optional. An attribute is optional, if the `minOccurs` is 0. If `minOccurs` is set to 1, the attribute is required.

`maxOccurs` specify the maximum number of occurrences of the element in an XML document. The default value for the `maxOccurs` attribute is 1. If its value is kept unbounded, it means that the element can appear unlimited number of times. The `maxOccurs` attribute defaults to 1 unless it is specified.

Explain the code on slide 39 which demonstrates the use of `minOccurs` and `maxOccurs` attributes.

The example demonstrates that the `Quantity` element can occur a minimum of zero times and a maximum of hundred times in the `Books` element.

The relationship between the `minOccurs` and `maxOccurs` attributes is displayed in table which is shown on slide 40.

Slides 41 and 42

Let us understand the element content.



Element Content 1-2

Code Snippet

```
// Books.xml
<?xml version="1.0"?>
<Books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Books.xsd">
<Title>A cup of tea</Title>
<Author>
<Name>Dennis Compton</Name>
</Author>
<Author>
<Name>George Ford</Name>
</Author>
<Publisher>
<Name>Orange</Name>
</Publisher>
</Books>
```

@ Aptech Limited XML Simplified/Module 4/41 of 63



Element Content 2-2

Code Snippet

```
// Books.xsd
<?xml version="1.0"?>
<xss:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xss:element name= "Books" type= "BookType">
<xss:complexType name= "AuthorType">
<xss:sequence>
<xss:element name= "Name" type="xs:string"/>
</xss:sequence>
</xss:complexType>
<xss:complexType name= "PublisherType">
<xss:sequence>
<xss:element name= "Name" type= "xs:string"/>
</xss:sequence>
</xss:complexType>
<xss:complexType name= "BookType">
<xss:sequence>
<xss:element name= "Title" type= "xs:string"/>
<xss:element name= "Author" type="ComposerType"
maxOccurs="unbounded"/>
<xss:element name= "Publisher" type="PublisherType"
minOccurs="0" maxOccurs= "unbounded"/>
</xss:sequence>
</xss:complexType>
</xss:element>
```

@ Aptech Limited XML Simplified/Module 4/42 of 63

Using slide 41, explain the element content.

A complex type can mention element content. The Element content can contain only elements. There can be instances wherein the contained elements can also have child elements.

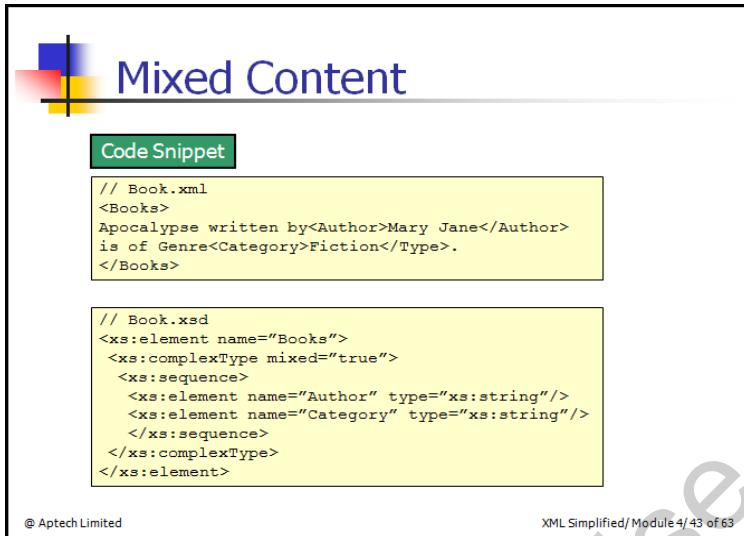
Explain the code on slide 42 which demonstrates the properties of element content.

This is an XML schema constrained document depicting book details.

In the XML document, the Author and the Publisher elements each contain Name elements. So built-in data types like xs:string cannot be used, instead AuthorName and PublisherName can be defined using top level xs:complexType elements.

Slide 43

Let us understand the mixed content.



The slide features a decorative graphic in the top-left corner consisting of overlapping colored squares (blue, red, yellow) forming a stylized 'X' or plus sign shape. The main title "Mixed Content" is centered at the top in a large, bold, blue font. Below the title is a green rectangular box labeled "Code Snippet". Inside this box are two code snippets: one for XML and one for XSD. The XML code defines a "Books" element containing text and elements for "Author" and "Category". The XSD code defines a "Books" complex type with a "mixed" attribute set to "true", allowing text and elements to be mixed together. At the bottom left is the copyright notice "@ Aptech Limited" and at the bottom right is the page number "XML Simplified/ Module 4/ 43 of 63".

```
// Book.xml
<Books>
Apocalypse written by<Author>Mary Jane</Author>
is of Genre<Category>Fiction</Type>.
</Books>

// Book.xsd
<x:element name="Books">
<x:complexType mixed="true">
<x:sequence>
<x:element name="Author" type="xs:string"/>
<x:element name="Category" type="xs:string"/>
</x:sequence>
</x:complexType>
</x:element>
```

Using slide 43, explain the mixed content.

A complex type can specify its content as a mixed content. Mixed content can contain text mixed with elements. The order and the number of elements that appears in the mixed content can also be specified in the schema. This type of content may or may not have attributes.

The mixed content is declared in exactly the same way, the declaration of element content is done. The only add-on to the declaration is, it takes a mixed attribute set to the true value.

Explain the code on slide 43 which demonstrates the properties of mixed content.

Slides 44 to 46

Let us understand how to group constructs.

Grouping Constructs 1-3

xs : all

- This grouping construct requires that each element in the group must occur at most once

Code Snippet

```
<xs:element name= "Books">
<xs:complexType>
<xs:all>
<xs:element name="Name" type="xs:string" minOccurs= "1"
maxOccurs= "1"/>
<xs:element name="ISBN" type="xs:string" minOccurs= "1"
maxOccurs= "1"/>
<xs:element name="items" type="Items" minOccurs= "1" />
</xs:all>
</xs:complexType>
</xs:element>
```

@ Aptech Limited

XML Simplified/Module 4/ 44 of 63

Grouping Constructs 2-3

xs : sequence

- It specifies each member of the sequence to appear in the same order

Code Snippet

```
<xs:element name="Books">
<xs:complexType>
<xs:sequence>
<xs:element name="Name" type="xs:string" />
<xs:element name="ISBN" type=" xs:string " />
<xs:element name="Price" type=" xs:string " />
</xs:sequence>
</xs:complexType>
</xs:element>
```

@ Aptech Limited

XML Simplified/Module 4/ 45 of 63

Grouping Constructs 3-3

xs : choice

- It will allow only one of the choices to appear instead of requiring all the elements to be present

Code Snippet

```
<xs:complexType name="AdressInfo">
<xs:group>
<xs:choice>
<xs:element name="Address" type="USAddress" />
<xs:element name="Address" type="UKAddress" />
<xs:element name="Address" type="FranceAddress" />
</xs:choice>
</xs:group>
```

@ Aptech Limited

XML Simplified/Module 4/ 46 of 63

Using slides 44 to 46, explain how to group constructs.

The XSL provides three grouping constructs that specify whether and how ordering of individual elements is important.

Explain `xs:all` - Grouping construct requires that each element in the group must occur at most once, but that order is not important. The only caution is that in this type of grouping the `minOccurs` attribute can be 0 or 1 and the `maxOccurs` attribute has to be 1.

Explain the code on slide 44 which demonstrates this concept.

The choice element `xs:choice` is the opposite of all elements. Instead of requiring all the elements to be present, it will allow only one of the choices to appear. The choice element provides an XML representation for describing a selection from a set of element types. The choice element itself can have `minOccurs` and `maxOccurs` attributes that establish exactly how many selections may be made from the choice.

Explain the code on slide 45 which demonstrates this concept.

In the code snippet, there are three child elements that are mutually exclusive. With the `xs:choice` element declared, only one element among the choices can be a child element of the parent element `AddressInfo`.

An `xs:sequence` element specifies each member of the sequence to appear in the same order in the instance document as mentioned in the `xs:sequence` element. The number of times each element is allowed to appear can be controlled by the element's `minOccurs` and `maxOccurs` attributes. Explain the code on slide 46 which demonstrates this concept.

Slide 47

Let us understand the objectives of lesson 4.



Lesson 4 – Working with Simple Types

In this last lesson, **Working with Simple Types**, you will learn to:

- Describe simple types.
- List and describe the data types used with simple types.
- Explain restrictions and facets.
- Identify the usage of attributes.

© Aptech Limited XML Simplified / Module 4 / 47 of 63

Using slide 47, explain the topics to be covered in the lesson 4.

Slide 48

Let us understand how to define simple type element.



Defining a Simple Type Element

- They are used to form the textual data and specify the type of data allowed within attributes and elements.

Syntax

```
<x:element name="XXXX" type="YYYY"/>
```

Code Snippet

```
Book.xml: XML Elements
<Book_name>The Da vinci code</Book_name>
<TotalNoOfPages>360</TotalNoOfPages>
<Author_name>Dan Brown</Author_name>

Book.xsd: Corresponding simple element definitions
<x:element name="Book_name" type="xs:string" />
<x:element name="TotalNoOfPages"
type="xs:integer"/>
<x:element name="Author_name" type="xs:string"/>
```

© Aptech Limited XML Simplified/ Module 4/ 48 of 63

Using slide 48, explain how to define simple type element.

A simple type is an XML element or attribute, which contains only text and no other elements or attributes.

The simple type declarations are used to form the textual data and specify the type of data allowed within attributes and elements.

Explain the code on slide 48 demonstrates that the element TotalNoOfPages can be specified as an integer type, which is again a positive integer with three digits.

Slide 49

Let us understand the data types used with simple types.



Data types used with Simple types

- Built-in simple type
- User-defined simple type

© Aptech Limited XML Simplified/ Module 4/ 49 of 63

Using slide 49, explain the data types used with simple types.

Elements of simple type tend to describe the content and the data type of a document, rather than its structure.

In XML Schema, one can mention the type of data an element can contain by assigning it a particular simple type definition. So, based upon the type of data it supports.

XML schema divides the elements of simple types into two broad categories:

- built-in simple type
- user-defined simple type

Tips:

You can also add restrictions (facets) to a data type in order to limit its content, or you can require the data to match a specific pattern.

Slides 50 and 51

Let us understand the built-in simple types.

Built-in Simple types 1-2

- There are several built-in simple types, such as integer, date, float and string.
- It can contain a default value or a fixed value.

Declaring simple type elements

xs:string
xs:integer
xs:boolean
...

default or fixed value specified for the element.

<xs:element name="name" type="simple-type" default/fixed="value"/>

© Aptech Limited XML Simplified/ Module 4/ 50 of 63

Built-in Simple types 2-2

Syntax

```
<xs:element name="XXXX" type="YYYY" default="ZZZZ"/>
```

Code Snippet

```
<xs:element name="AccountType" type="xs:string"  
fixed="Savings"/>  
<xs:element name="BalanceAmount" type="xs:integer"  
default="5000"/>
```

© Aptech Limited XML Simplified/ Module 4/ 51 of 63

Using slide 50, explain the built-in simple types.

There are several built-in simple types, such as integer, date, float, and string that one can use without further modifications.

A built-in simple element can contain a default value or a fixed value. A 'default' value is the value that is assigned automatically to the element when no other value has been specified. A 'fixed' value is assigned to an element, when there is no need to change the value for that element. Figure on slide 50 depicts built-in simple types.

Explain the syntax of built-in simple type using slide 51.

In this syntax, 'XXXX' is the name of the element, 'YYYY' is the data type of the element, and 'ZZZZ' is the default value specified for the element.

Explain the code on slide 51 which demonstrates the declaration of built-in simple type.

The code shows the declaration of built-in simple types "AccountType" and "BalanceAmount". The fixed value for the "AccountType" is "Savings" and the default value for "BalanceAmount" is "5000".

Slides 52 and 53

Let us understand the user-defined simple types.

User-defined Simple types 1-2

- Custom user defined datatype can be created using the `<simpleType>` definition.

Syntax

```
<xss:simpleType name="name of the simpleType">
  <xss:restriction base="built-in data type">
    <xss:constraint="set constraint to limit the
      content"/>
  </xss:restriction>
</xss:simpleType>
```

© Aptech Limited XML Simplified/ Module 4/ 52 of 63



User-defined Simple types 2-2

Code Snippet 1

```
<xss:simpleType name="AngleMeasure">
<xss:restriction base="xss:integer">
<xss:minInclusive value="0"/>
<xss:maxInclusive value="360"/>
</xss:restriction></xss:simpleType>
```

Code Snippet 2

```
<xss:simpleType name="triangle">
<xss:restriction base="xsd:string">
<xss:enumeration value="isosceles"/>
<xss:enumeration value="right-angled"/>
<xss:enumeration value="equilateral"/>
</xss:restriction>
</xss:simpleType>
```

© Aptech Limited XML Simplified/Module 4/ 53 of 63

Using slides 52 and 53, explain the user-defined simple types.

XML-schema supports a library of built-in data types. However, during the course of writing complex schema documents, a developer may need a type of data that is not defined in the schema recommendation. For instance, consider the declaration of an element `<AngleMeasure>`.

It implements the available restrictions to its content to accept non-negative numeric values, but still it accepts unwanted values like:

`<AngleMeasure>490</AngleMeasure>`

As per schema declaration, "490" is a valid non-negative numeric value, but still it needs numeric type that allows the values with a range from 0 to 360. Hence, a custom user defined data type can be created using the `<simpleType>` definition.

Explain the syntax of simple types.

Then, explain the codes on slide 53 which demonstrate the properties of user-defined simple types.

Slides 54 and 55

Let us understand the restrictions.

Restrictions 1-2

- It can be specified for the simpleType elements.
- They are declared using the <restriction> declaration.

Syntax

```
<restriction base="name of the simpleType you are deriving from">
```

In this <restriction> declaration, the base data type can be specified using the base attribute.

Code Snippet

```
<xs:simpleType name="Age">
  <xs:restriction base="xs:integer">
    ...
  </xs:restriction>
</xs:simpleType>
```

© Aptech Limited XML Simplified/Module 4/ 54 of 63

Restrictions 2-2

Syntax

```
<xs:simpleType name= "name">
  <xs:restriction base= "xs:source">
    <xs:facet value= "value"/>
    <xs:facet value= "value"/>
    ...
  </xs:restriction>
</xs:simpleType>
```

Code Snippet

```
<xs:simpleType name="triangle">
  <xs:restriction base="xs:string">
    <xs:enumeration value="isosceles"/>
    <xs:enumeration value="right-angled"/>
    <xs:enumeration value="equilateral"/>
  </xs:restriction>
</xs:simpleType>
```

© Aptech Limited XML Simplified/Module 4/ 55 of 63

Using slide 54, explain the restrictions.

Declaration of a data type puts certain limitations on the content of an XML element or attribute. If an XML element is of type "xs:integer" and contains a string like "Welcome", the element will not be validated. These limitations are called restrictions, which defines allowable values for XML elements and attributes.

Restrictions can be specified for the simpleType elements and restriction types are declared using the <restriction> declaration. Basically, the <restriction> declaration is used to declare a derived simpleType, which is a subset of its base simpleType. The value of the base attribute can be any existing simpleType, or built-in XML Schema data type.

Explain the syntax of restrictions using slide 54.

In this <restriction> declaration, the base data type can be specified using the base attribute.

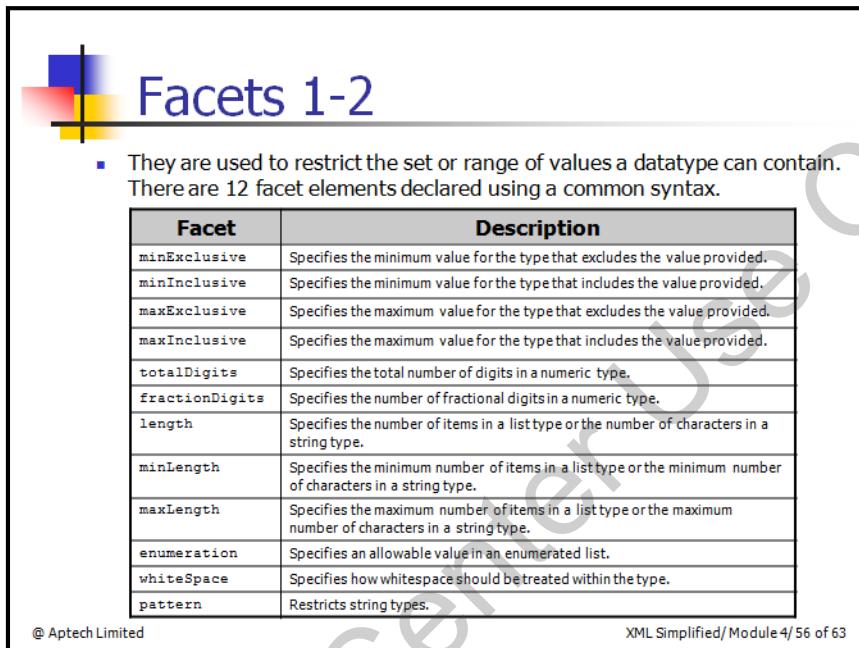
Explain the code on slide 55 which demonstrates a simpleType "Age" is derived by specifying the base as integer type.

Tips:

Restrictions on XML elements are called facets.

Slides 56 and 57

Let us understand the Facets.

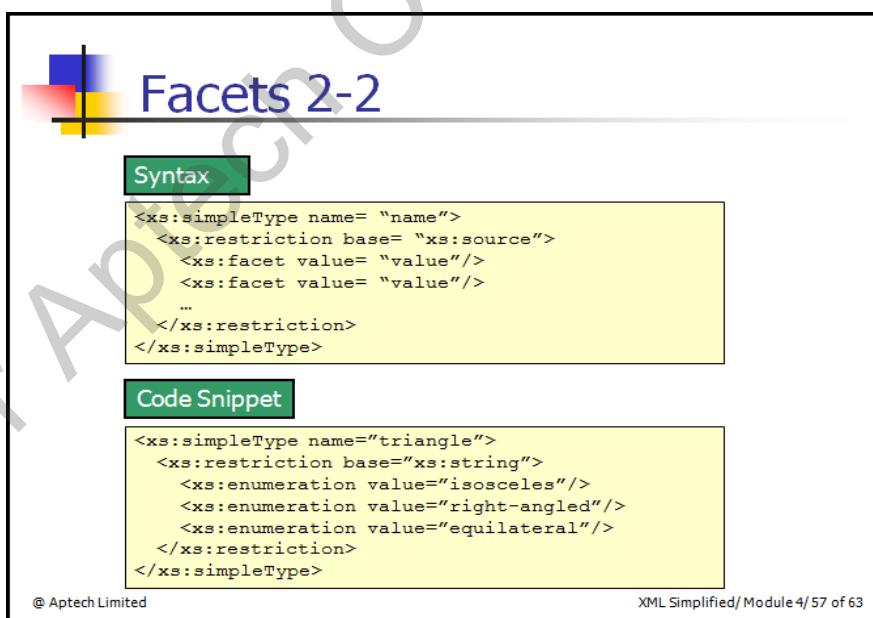


Facets 1-2

- They are used to restrict the set or range of values a datatype can contain. There are 12 facet elements declared using a common syntax.

Facet	Description
<code>minExclusive</code>	Specifies the minimum value for the type that excludes the value provided.
<code>minInclusive</code>	Specifies the minimum value for the type that includes the value provided.
<code>maxExclusive</code>	Specifies the maximum value for the type that excludes the value provided.
<code>maxInclusive</code>	Specifies the maximum value for the type that includes the value provided.
<code>totalDigits</code>	Specifies the total number of digits in a numeric type.
<code>fractionDigits</code>	Specifies the number of fractional digits in a numeric type.
<code>length</code>	Specifies the number of items in a list type or the number of characters in a string type.
<code>minLength</code>	Specifies the minimum number of items in a list type or the minimum number of characters in a string type.
<code>maxLength</code>	Specifies the maximum number of items in a list type or the maximum number of characters in a string type.
<code>enumeration</code>	Specifies an allowable value in an enumerated list.
<code>whiteSpace</code>	Specifies how whitespace should be treated within the type.
<code>pattern</code>	Restricts string types.

@ Aptech Limited XML Simplified/ Module 4/ 56 of 63



Facets 2-2

Syntax

```
<xs:simpleType name= "name">
  <xs:restriction base= "xs:source">
    <xs:facet value= "value"/>
    <xs:facet value= "value"/>
    ...
  </xs:restriction>
</xs:simpleType>
```

Code Snippet

```
<xs:simpleType name="triangle">
  <xs:restriction base="xs:string">
    <xs:enumeration value="isosceles"/>
    <xs:enumeration value="right-angled"/>
    <xs:enumeration value="equilateral"/>
  </xs:restriction>
</xs:simpleType>
```

@ Aptech Limited XML Simplified/ Module 4/ 57 of 63

Using slides 56 and 57, explain the facets.

With XML Schemas, custom restrictions can be specified on XML elements and attributes. These restrictions are called facets.

Facets are used to restrict the set or range of values a data type can contain. The value range defined by the facet must be equal to or narrower than the value range of the base type.

There are 12 facet elements, declared using a common syntax. They each have a compulsory value attribute that indicates the value for the facet. One restriction can contain more than one facet. Any values appearing in the instance and value spaces must conform to all the listed facets.

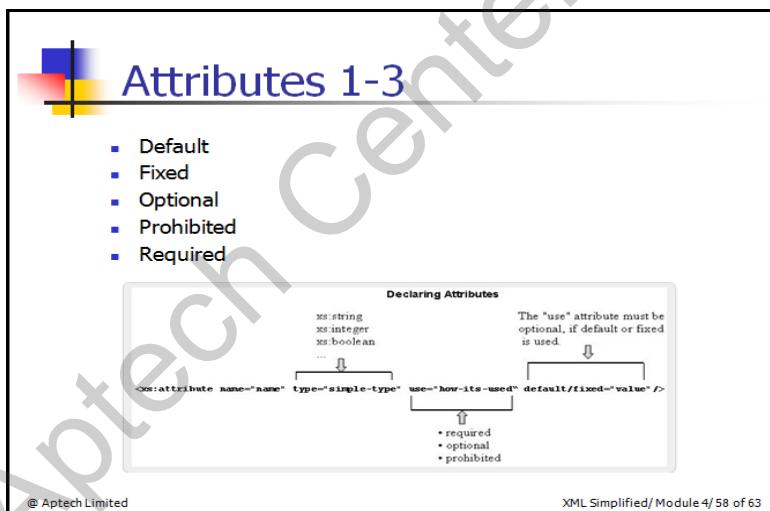
Table on slide 56 depicts the constraining facets.

Explain the syntax and code on slide 57 which demonstrates that the value attribute gives the value of that facet.

Here, the facet enumeration is added to the restriction with the value attribute as either `isosceles`, `right-angled`, or `equilateral`. So, an element declared to be of type triangle must be a string with a value of either `isosceles`, `right-angled`, or `equilateral`.

Slides 58 to 60

Let us understand the attributes.



Attributes 2-3

Syntax

```
<xs:attribute name="Attribute_name"
type="Attribute_datatype"/>
```

where,
Attribute_name is the name of the attribute.
Attribute_datatype specifies the data type of the attribute. There are lot of built in data types in XML schema such as string, decimal, integer, boolean, date, and time.

© Aptech Limited

XML Simplified/ Module 4/ 59 of 63

Attributes 3-3

Code Snippet

```
...
<xs:complexType name= "SingerType">
<xs:sequence>
<xs:element name= "Name">
<xs:complexType>
<xs:all>
<xs:element name= "FirstName" type="xs:string"/>
<xs:element name= "LastName" type="xs:string"/>
</xs:all>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name= "age" type="xs:positiveInteger"
use= "optional"/>
</xs:complexType>
...
```

© Aptech Limited

XML Simplified/ Module 4/ 60 of 63

Using slides 58 to 60, explain the attributes.

XML elements can contain attributes that describe elements. Within XML Schemas, attribute declarations are similar to element declarations. To declare an attribute, `<xs:attribute>` element is used.

An attribute can be indicated whether it is required or optional or whether it has a default value. A default value is automatically assigned to the attribute when no other value is specified.

The `use` attribute specifies whether the attribute is required or optional.

Explain different values of attributes. The figure on slide 58 depicts the attribute declaration.

Explain the syntax for defining an attribute using slide 59.

Explain the code on slide 60 which demonstrates the properties of attribute.

In the example, `xs:attribute` elements comes after `xs:sequence` and `xs:group` that forms the body of the element. The element `Name` can have an optional attribute named `age` with type `positiveInteger`.

Slides 61 to 63

Let us summarize the session.



Summary 1-3

- **XML Schema**
 - An XML Schema is an XML based alternative to DTDs, which describes the structure of an XML document.
 - An XML Schema can define elements, attributes, child elements and the possible values that can appear in a document.
 - Schemas overcome the limitations of DTDs and allow Web applications to exchange XML data robustly, without relying on adhoc validation tools.
- **Exploring XML Schemas**
 - XML schema offers built-in and user defined data types.
 - It supports built-in data types like string, boolean, number, date, dateTime, binary, and uri.
 - It also supports integer, decimal, time, and user-defined data types.

© Aptech Limited XML Simplified/ Module 4/ 61 of 63



Summary 2-3

- **Working with Complex Types**
 - Elements with complex type may contain nested elements and have attributes.
 - A complex element can be defined by directly naming the element and by using the name and the type attribute of the complex type.
 - `minOccurs` and `maxOccurs` specify the minimum and maximum number of occurrences of the element in an XML document respectively.
 - Element content in an XML document contains only XML elements and mixed content contains text mixed with elements.
 - The grouping constructs in XML schema specify the order of XML elements.

© Aptech Limited XML Simplified/ Module 4/ 62 of 63



Summary 3-3

- **Working with Simple Types**

- The elements of simple type describe the content and data type of the element rather than its structure.
- The simple type can have built-in and user defined data types.
- Simple type definition takes one of the two values, default or fixed as per the requirement.
- The user-defined data type can be derived from a built-in type or an existing simple type.
- Use of restrictions and facets restricts its content to user prescribed values.
- Schema supports usage of attributes in elements, which is declared with a simple type definition.

Using slides 61 to 63, summarize the session. End the session, with a brief summary of what has been taught in the session.

4.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Style Sheets that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 5 - Style Sheets

5.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

5.1.1 Objectives

By the end of this session, the learners will be able to:

- Style Sheets
- Selectors in CSS
- Properties and values
- Inheritance and Cascades in CSS

5.1.2 Teaching Skills

To teach this session, you should be well-versed with style sheets. The topics such as how to use Cascading Style Sheets to format XML document, and the cascading order of style rules are also covered.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Then give the students the overview of the current session in the form of Session Overview. Show the students slide 2 of the presentation.

Tell the students that this session introduces style sheets and how to create style sheets. They will also know understand how to use Cascading Style Sheets (CSS) to format XML document and the cascading order of style rules.

5.2 In-Class Explanations

Slide 3

Let us understand the objectives of lesson 1.

Lesson 1 – Style Sheets

In this first lesson, **Style Sheets**, you will learn to:

- Define and describe style sheets.
- Define and describe cascading style sheets (CSS).
- Explain how to implement styles with CSS.

@ Aptech Limited XML Simplified/ Module 5/ 3 of 53

Using slide 3, explain the objectives of lesson 1.

Slide 4

Let us understand the need for style sheets.

Need of Style Sheets

- They are a set of rules that describe the appearance of data in an XML document.

XML Document

Style Sheet

○	★
○	□
○	+
○	□

Formatted Document

@ Aptech Limited XML Simplified/ Module 5/ 4 of 53

Using slide 4, explain the need for style sheets.

Style sheets are a set of rules that describe the appearance of data in an XML document.

XML was inspired by the problems posed by presentational markup. Presentational markup does not describe data; it defines the appearance of data. In a document, if you had to change all proper nouns from bold to italics, you would have to do it manually for each proper noun. Presentation markup failed to provide same look and feel across multiple devices such as computers, Personal Digital Assistant (PDA) devices, and cell phones.

Style sheets and XML solve these problems. XML describes data. Style sheets define the appearance of data. However, both XML and style sheets are defined in separate files.

Explain the figure on slide 4 that depicts the formatted document with applied style and XML.

Slide 5

Let us understand the various style sheets.

The slide has a decorative graphic in the top-left corner consisting of overlapping colored squares (blue, red, yellow). The title 'Various Style Sheets' is centered at the top in a blue font. Below the title, there are two main sections: 'Cascading Style Sheet (CSS)' and 'Extensible Style Sheet (XSL)'. Each section contains a bulleted list of features. At the bottom left is the copyright notice '@ Aptech Limited' and at the bottom right is the page information 'XML Simplified/ Module 5/ 5 of 53'.

Various Style Sheets

Cascading Style Sheet (CSS)

- It allows you to control the appearance of data in HTML and XML documents.

Extensible Style Sheet (XSL)

- It is used to define the appearance of data contained only in XML documents.

@ Aptech Limited XML Simplified/ Module 5/ 5 of 53

Using slide 5, explain various style sheets.

There are several style sheets available. However, following two are the most popular style sheets:

- Cascading Style Sheet (CSS)
- Extensible Stylesheet (XSL)

Explain the CSS that allows you to control the appearance of data in HTML and XML documents by providing various properties to define:

- Position and size of data to be displayed
- Foreground and background color of data
- Font to be used to display data
- Spacing between data

Explain Extensible Stylesheet (XSL).

Mention XSL is a style sheet language used to define the appearance of data contained only in XML documents. However, it also allows you to transform XML documents.

Slide 6

Let us understand the structure of CSS.

The slide has a decorative graphic in the top-left corner consisting of overlapping colored squares (blue, red, yellow). The title 'Cascading Style Sheets' is centered at the top in a large blue font. Below the title is a bulleted list of four points:

- It comprises a set of rules for various elements in a document.
- Each rule defines how the data enclosed in the element should be displayed.
- The term cascading is derived from this ability to mix and match rules.

Below the list is a code block divided into two sections: 'XML Document' and 'Style Sheet'.

```
XML Document
<Library>
  <Films>
    <Film>
      <Name>Ghost Rider</Name>
      <Sound>Yes</Sound>
      <Year>2007</Year>
      <Cast>Nicolas Cage and others</Cast>
    </Film>
  </Films>
</Library>

Style Sheet
Name { font-style: bold }
Cast { color: aqua }
```

At the bottom left is the copyright notice '© Aptech Limited' and at the bottom right is 'XML Simplified/ Module 5 / 6 of 53'.

Using slide 6, explain the structure of CSS.

CSS is a rule-based language invented in 1996 to write formatting instructions for data contained in HTML documents.

A CSS style sheet comprises a set of rules for various elements in a document. Each rule defines how the data enclosed in the element should be displayed.

Style sheet rules can be combined from different sources, or subsets can be created or the rules can be overridden. The term **cascading** is derived from this ability to mix and match rules.

Figure the code on slide 6 that shows an example of CSS.

Slide 7

Let us understand the benefits of CSS.



Benefits of CSS

- Any style or presentation changes to data can be achieved faster as changes are to be implemented in one place.
- Device independence can be achieved by defining different style sheets for different device.
- Reduction in document code as the presentation information is stored in a different file and can be reused.

@ Aptech Limited XML Simplified/ Module 5/ 7 of 53

Using slide 7, explain the benefits of CSS.

Slide 8

Let us understand the style rules.



Style Rules

- They define how the content enclosed in an element will be displayed.
- These rules are applicable to all child elements within an element.
 - **Selector**
 - **Property**
 - **Value**

`selector {property : value}`

@ Aptech Limited XML Simplified/ Module 5/ 8 of 53

Using slide 8, explain the style rules.

A style sheet in CSS comprises a set of style rules. These rules define how the content enclosed in an element will be displayed. These rules are applicable to all child elements within an element. A style rule comprises a selector, a property, and a value. A property and a value separated with a colon are referred as property declaration. Figure on slide 8 depicts the style rules.

Explain in detail about selector, property, and value.

In CSS, style rules can comprise more than one selector. To include multiple selectors or group multiple selectors, one needs to provide a comma-separated list of element names.

Slide 9

Let us understand the ways of writing style rules.

Ways of Writing Style rules

- A single selector can have more than one property-value pairs associated with it.
- A collection of one or more property-value pairs can be associated with more than one selector.

Single selector, multiple property declarations

```
CD { font-family: sans-serif; color: black }
```

Multiple selectors, multiple property declarations

```
CD, Name, Title { font-family: times; color: black }
```

© Aptech Limited

XML Simplified/Module 5/ 9 of 53

Using slide 9, explain the ways of writing style rules.

A single selector can have more than one property-value pairs associated with it. For example, figure on slide 9 shows a CD element having two property declarations – one to set the font family to sans-serif, and other to set the color of text to black. Notice the property-value pairs are separated by a semi-colon.

Similarly, a collection of one or more property-value pairs can be associated with more than one selector. For example, figure on slide 9 shows two property declarations assigned to three elements namely CD, Title, and Name.

Slide 10

Let us understand the external style sheets.



External Style Sheets

- It should appear in the prolog section of an XML document, that is, it should appear before any tag of an element.
- One XML document can have more than one style sheet processing instructions, each linked to one .css file.

Syntax

```
<?xml-stylesheet href="headers.css" type="text/css"?>
```

where,

`xml-stylesheet` is the processing instruction.

`url` is the URL of a .css file; the .css file can be on a local system or anywhere on the Internet.

`type="text/css"` is optional; however if a browser does not support CSS, it informs the browser that it does not have to download a .css file.

Code Snippet

```
<?xml-stylesheet href="url" [type="text/css"]?>
```

@ Aptech Limited

XML Simplified/Module 5/10 of 53

Using slide 10, explain the external style sheets.

In CSS, the style rules are written in a file with the extension `.css`. This file is associated with an XML document using a style sheet processing instruction. A few points to note about style sheet processing instruction are:

- It should appear in the prolog section of an XML document, that is, it should appear before any tag of an element.
 - One XML document can have more than one style sheet processing instructions, each linked to one .css file.

Explain the syntax and code snippet on slide 10 for of external style sheet.

Slide 11

Let us understand the objectives of lesson 2.



Lesson 2 – Selectors in CSS

In this second lesson, **Selectors in CSS**, you will learn to:

- Identify simple selectors in CSS.
- State the use of universal selector in CSS.
- Describe ID selectors.

@ Aptech Limited XML Simplified/Module 5/ 11 of 53

Using slide 11, explain the topics covered in lesson 2.

Slide 12

Let us understand the simple selectors.



Simple Selectors

- It comprises an element name followed by one or more property declarations.
- It matches every occurrence of the element in a document.

Code Snippet

```
/* Simple selector */
CD { color: black }
/* Single element, multiple property declarations */
CD { color: white; background-color: blue }
/* Multiple elements, multiple property declarations */
CD, Name, Title { color: white; background-color: blue }
```

@ Aptech Limited XML Simplified/Module 5/ 12 of 53

Using slide 12, explain the simple selectors.

Simple selector comprises an element name followed by one or more property declarations. Same property declarations can be assigned to multiple elements by separating element names with a comma. Simple selectors match every occurrence of the element in a document.

Explain the code snippet on slide 12 demonstrating the simple selectors.

Slide 13

Let us understand the universal selectors.

Universal Selector

- It comprises an asterisk followed by property declarations.
- It matches all the elements in a document.

Code Snippet

```
* { color : blue }
```

Displays the content of all elements in a document in blue.

© Aptech Limited XML Simplified/Module 5/ 13 of 53

Using slide 13, explain the universal selectors.

A universal selector comprises an asterisk followed by property declarations. It is used when you want to assign the same style rules to all the elements in a document. A universal selector matches all the elements in a document. The code shown on the slide displays the content of all elements in a document in blue.

Explain the code snippet for the same using slide 13.

Slides 14 and 15

Let us understand the ID selectors.

ID Selector 1-2

- It comprises a hash (#) symbol, immediately followed by an attribute's value followed by property declarations.
- It is used to define styles for unique elements in a document.

Unique element

```
<CD>
  <Audio>
    <Name id="1001">2007 Hits</Name>
  </Audio>
</CD>
```

#1001 { color : blue }

ID selector

© Aptech Limited XML Simplified/Module 5/ 14 of 53



ID Selector 2-2

Syntax

```
#attribute_value { property_declarations }
```

Code Snippet

```
#1001 { color : blue }
```

Displays the content of an element in blue if its id attribute's value equals 1001.

@ Aptech Limited

XML Simplified/ Module 5/ 15 of 53

Using slides 14 and 15, explain the ID selectors.

An ID selector comprises a hash (#) symbol, immediately followed by an attribute's value followed by property declarations. It is used to define styles for unique elements in a document. For example, if you want the data of a unique element to be in a different style, you would define an ID selector for it. Unique element is one which has one of its attributes named as id as shown in figure on slide 14.

Explain the syntax and code snippet for ID selectors using slide 15.

Slide 16

Let us understand the properties and values.



Lesson 3 – Properties and Values

In this third lesson, **Properties and Values**, you will learn to:

- State and describe how to use color properties.
- Describe the font property.
- Describe the other properties such as margins, borders, padding.
- Explain briefly about positioning and alignment.

@ Aptech Limited

XML Simplified/ Module 5/ 16 of 53

Using slide 16, explain the topics covered in lesson3.

Slide 17

Let us understand the color properties.



Color Properties

Color Names	RGB Percentages	RGB Values	Hexadecimal Values
aqua	rgb(0%, 65%, 65%)	rgb(0, 160, 160)	#00a0a0
black	rgb(0%, 0%, 0%)	rgb(0, 0, 0)	#000000
blue	rgb(0%, 32%, 100%)	rgb(0, 80, 255)	#0050ff
gray	rgb(65%, 65%, 65%)	rgb(160, 160, 160)	#a0a0a0
green	rgb(0%, 100%, 0%)	rgb(0, 255, 0)	#00ff00
lime	rgb(0%, 65%, 0%)	rgb(0, 160, 0)	#00a000
maroon	rgb(70%, 0%, 32%)	rgb(176, 0, 80)	#b00050
navy	rgb(0%, 0%, 65%)	rgb(0, 0, 160)	#0000a0
olive	rgb(65%, 65%, 0%)	rgb(160, 160, 0)	#a0a000
purple	rgb(65%, 0%, 65%)	rgb(160, 0, 160)	#a000a0
red	rgb(100%, 0%, 32%)	rgb(255, 0, 80)	#ff0050
silver	rgb(90%, 90%, 90%)	rgb(225, 225, 225)	#d0d0d0
teal	rgb(0%, 65%, 100%)	rgb(0, 160, 255)	#00a0ff
white	rgb(100%, 100%, 100%)	rgb(255, 255, 255)	#ffffff
yellow	rgb(100%, 100%, 0%)	rgb(255, 255, 0)	#ffff00

@ Aptech Limited

XML Simplified/Module 5/ 17 of 53

Using slide 17, explain the color properties.

The CSS provides properties to set the foreground and background color of text. CSS uses color name, red-green-blue (RGB) values, RGB percentages and hexadecimal values to specify color values. The various ways in which color values are specified is shown in table on slide 17.

Slides 18 to 20

Let us understand how to set color properties.



Setting Color Properties 1-3

Syntax

```
color: colorValue
background-color: colorValue
```

where,

- color**
Property to set the foreground color of text in an element.
- colorValue**
colorValue can take up any value from the CSS color table.
- background-color**
Property to set the background color of text in an element.

@ Aptech Limited

XML Simplified/Module 5/ 18 of 53



Setting Color Properties 2-3

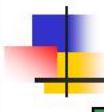
Code Snippet

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet href="Colors.css" type="text/css"?>
3 <Cars>
4   <Vehicle>
5     <Manufacturer>BMW</Manufacturer>
6     <Model>M3</Model>
7     <Color>Metallic Silver</Color>
8     <Price>40,000</Price>
9     <Location>Liverpool</Location>
10    </Vehicle>
11 </Cars>

```

© Aptech Limited XML Simplified/ Module 5/ 19 of 53



Setting Color Properties 3-3

Style Sheet

```

1 Cars { background-color: rgb(0%,32%,100%); color: #ffffff; }
2 Price { color: yellow; }

```

where,

```

Cars { background-color:rgb(0%,32%,100%);color: #ffffff; }
Causes the text enclosed in Cars element to be displayed in white color
with a background color of blue.
Price{color:yellow; }
Causes the text enclosed in Price element to be displayed in yellow color.

```

Output

BMW M3 Metallic Silver 40,000 Liverpool

© Aptech Limited XML Simplified/ Module 5/ 20 of 53

Using slide 18, explain how to set color properties.

The CSS specification provides the properties color and background-color to set the foreground and background color of text enclosed in elements.

Explain the figure on slide 18 that depicts the syntax for setting properties.

Explain the figure on slide 19 that shows the code for style rules defined in Colors.css file.

Explain the figure on slide 20 depicts the style sheet for Colors.css file. Also, explain the output for the same.

Slide 21

Let us understand the font properties.

Font Properties

Property Name	Description
<code>font-family</code>	To specify the font family
<code>font-size</code>	To specify the size of font
<code>font-style</code>	To specify the style of font
<code>font-weight</code>	To specify the weight of font

FONST STYLES

```

  FONST STYLES
    Font Style
      FONST Style
        Font Style
          font style
  
```

© Aptech Limited XML Simplified/Module 5/ 21 of 53

Using slide 21, explain the font properties.

The CSS specification defines several font properties to set the font family, font size, font style such as bold, italics, and so forth. Some of these properties are listed in table on slide 21.

Slides 22 and 23

Let us understand the `font-family` property.

Font-family Property 1-2

Syntax

```
font-family : "font-family name(s)"
```

where,

`font-family`
Property to specify the font-family to be used.
`font-family name (s)`
Comma separated list of font-family names such as serif, sans-serif, monospace, cursive, and fantasy. The list should start with the most specific font in which you want to display the data and end with the most generic font.

Code Snippet

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet href="FontFamily.css" type="text/css"?>
3  <Cars>
4  <Vehicle>
5   <Manufacturer>BMW</Manufacturer>
6   <Model>M3</Model>
7   <Color>Metallic Silver</Color>
8   <Price>40,000</Price>
9   <Location>Liverpool</Location>
10  </Vehicle>
11  </Cars>
  
```

© Aptech Limited XML Simplified/Module 5/ 22 of 53



Font-family Property 2-2

Style Sheet

```
1 Cars { font-family: "arial, times, serif" }
```

Output

BMW M3 Metallic Silver 40,000 Liverpool

@ Aptech Limited XML Simplified/ Module 5/ 23 of 53

Using slides 22 and 23, explain the `font-family` property.

The `font-family` property is used to specify the name of the font family to be applied to an element.

Explain the figure on slide 22 that shows the syntax for `font-family` property. Also, explain the code for style rules stored in `FontFamily.css` file.

Then, explain slide 23 that depicts the style sheet for `FontFamily.css` file.

Slides 24 to 26

Let us understand the `font-size` property.



Font-size Property 1-3

Syntax

```
font-size : "xx-small | x-small | small | medium | large | x-large | xx-large"
```

where,
`font-size`
 Property to specify the size of font.
`xx-small|x-small|small|medium|large|x-large|xx-large`
 One of various values that can be assigned to the property `font-size`.

@ Aptech Limited XML Simplified/ Module 5/ 24 of 53

Font-size Property 2-3

Code Snippet

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet href="FontSize.css" type="text/css"?>
3  <Cars>
4   <Vehicle>
5     <Manufacturer>BMW</Manufacturer>
6     <Model>M3</Model>
7     <Color>Metallic Silver</Color>
8     <Price>40,000</Price>
9     <Location>Liverpool</Location>
10    </Vehicle>
11 </Cars>

```

© Aptech Limited XML Simplified/Module 5/ 25 of 53

Font-size Property 3-3

Style Sheet

```

1 Cars { font-size: medium }

```

where,

```

Cars{font-size:medium}
All the text enclosed in Cars element and its child elements will
be displayed with medium font size.

```

Output

BMW M3 Metallic Silver 40,000 Liverpool

© Aptech Limited XML Simplified/Module 5/ 26 of 53

Using slides 24 to 26, explain the `font-size` property.

The `font-size` property is used to specify the size of font used to display element data. Explain the figure on slide 24 that shows the syntax for `font-size` property.

Then, explain the figure on slide 25 that shows the code for style rules defined in `FontSize.css` file.

Finally, explain the style sheet for `FontSize.css` file on slide 26.

The text enclosed in the element `Cars` will be displayed in either **Arial**, **Times New Roman**, or **Serif** font. The rule becomes applicable to all the elements enclosed in `Cars` element. If the system does not support **Arial** font, then next fonts will be selected in sequence.

Slides 27 to 29

Let us understand the font style and weight properties.

Font Style and Weight Properties 1-3

Syntax

```
font-style: normal | oblique | italic
font-weight: light | normal | bold
```

where,

- font-style**
Property to specify the style of text in an element.
`normal|oblique|italic`
One of the values that can be assigned to font-style property.
- font-weight**
Property to specify the weight style of the text in an element.
`light | normal | bold`
One of the values that can be assigned to font-weight property.

© Aptech Limited XML Simplified/Module 5/27 of 53

Font Style and Weight Properties 2-3

Code Snippet

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xmlstylesheet href="FontStyle.css" type="text/css"?>
3 <Cars>
4   <Vehicle>
5     <Manufacturer>BMW</Manufacturer>
6     <Model>M3</Model>
7     <Color>Metallic Silver</Color>
8     <Price>40,000</Price>
9     <Location>Liverpool</Location>
10    </Vehicle>
11 </Cars>
```

© Aptech Limited XML Simplified/Module 5/28 of 53

Font Style and Weight Properties 3-3

Style Sheet

```

1 Manufacturer { font-weight: bold }
2 Location { font-style: italic }
```

where,

- `Manufacturer { font-weight: bold }`
The text enclosed in Manufacturer element will be displayed in bold.
- `Location { font-style: italic }`
The text enclosed in Location element will be displayed in italics.

Output

BMW M3 Metallic Silver 40,000 Liverpool

© Aptech Limited XML Simplified/Module 5/29 of 53

Using slides 27 to 29, explain the font style and weight properties.

CSS provides two properties namely, `font-style` and `font-weight` to add emphasis or meaning to parts of data.

Explain the figure on slide 27 that shows the syntax for font style and weight properties.

Explain the figure on slide 28 that shows the code for style rules defined in `FontStyle.css` file.

Explain the figure on slide 29 that depicts the style sheet for `FontStyle.css` file.

Slides 30 to 32

Let us understand the margins in CSS.

Margins in CSS 1-3

Syntax

```
margin-left | margin-right | margin-top | margin-bottom =  
marginValue
```

where,

```
margin-left | margin-right | margin-top | margin-bottom  
The various margin properties to set left, right, top and bottom margins.  
marginValue  
The value to be assigned to one of the margin properties. This value can be  
a fixed value or a percentage.
```

© Aptech Limited XML Simplified/Module 5/30 of 53

Margins in CSS 2-3

Code Snippet

```
1 <?xml version="1.0" encoding="UTF-8"?>  
2 <?xml-stylesheet href="Margin.css" type="text/css"?>  
3 <Cars>  
4   <Vehicle>  
5     <Manufacturer>BMW</Manufacturer>  
6     <Model>M3</Model>  
7     <Color>Metallic Silver</Color>  
8     <Price>40,000</Price>  
9     <Location>Liverpool</Location>  
10    </Vehicle>  
11 </Cars>
```

© Aptech Limited XML Simplified/Module 5/31 of 53



Margins in CSS 3-3

Style Sheet

```
1 Manufacturer { margin-left: 20; margin-right: 50; }
2 Manufacturer { background-color: aqua; }
3 Vehicle { background-color: orange; }
```

where,

```
Manufacturer {margin-left:20; margin-right:50; }
Inserts a space of 20 pixels to the left and a space of 50 pixels to the right of text enclosed in element Manufacturer.
Manufacturer {background-color: aqua; }
Sets the background of text enclosed in element Manufacturer to aqua.
Vehicle {background-color: orange; }
Sets the background of text enclosed in element Vehicle to orange.
```

Output

BMW M3 Metallic Silver 40,000 Liverpool

@ Aptech Limited XML Simplified/Module 5/32 of 53

Using slide 30, explain margins in CSS.

Every element in an XML document is displayed in its own box. CSS provides four properties namely `margin-left`, `margin-right`, `margin-top`, and `margin-bottom` to insert space around the element's box.

Explain the figure on slide 30 that shows the syntax for margins in css.

Explain the figure on slide 31 that shows the code for style rules defined in `Margin.css` file.

Explain the figure on slide 32 that depicts the style sheet for `Margin.css` file.

Slides 33 to 35

Let us understand the `border` properties in CSS.

Border Properties in CSS 1-3

Syntax

```
border : border_width border_style border_color
```

where,

- `border`: Property to set the border of the box surrounding an element's data.
- `border_width`: Specifies the thickness of the border. Possible values are thin, medium and thick.
- `border_style`: Specifies the style of the border. Possible values are solid, dashed, dotted, groove, ridge, double, inset, outset.
- `border_color`: Specifies the color of the border. All values that are applicable to CSS color property are allowed.

© Aptech Limited XML Simplified/Module 5/33 of 53

Border Properties in CSS 2-3

Code Snippet

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xmlstylesheet href="Border.css" type="text/css"?>
3 <Cars>
4   <Vehicle>
5     <Manufacturer>BMW</Manufacturer>
6     <Model>M3</Model>
7     <Color>Metallic Silver</Color>
8     <Price>40,000</Price>
9     <Location>Liverpool</Location>
10    </Vehicle>
11 </Cars>
```

© Aptech Limited XML Simplified/Module 5/34 of 53

Border Properties in CSS 3-3

Style Sheet

```

1 Manufacturer {border: thick dashed magenta }
2 Model { border: thick solid olive }
3 Color { border: thick groove aqua }
4 Price { border: thick inset gray }
```

where,

- `Manufacturer {border: thick dashed magenta }`
Displays a thick and dashed magenta border around the content of Manufacturer element.
- `Model { border: thick solid olive }`
Displays a thick and solid olive border around the content of Model element.
- `Color {border: thick groove aqua}`
Displays a thick and groove aqua border around the content of Color element.
- `Price {border: thick inset gray}`
Displays a thick and inset gray border around the content of Price element.

Output Liverpool

© Aptech Limited XML Simplified/Module 5/35 of 53

Using slides 33 to 35, explain the `border` properties in CSS.

Borders are rectangular outlines surrounding an element's box. CSS provides properties to create dotted, solid, and groove borders to name a few.

Explain the syntax and the style rules for border properties explained on slides 33 and 34.

Then, explain the figure on slide 35 that depicts the style sheet for `Border.css` file.

Slides 36 and 37

Let us understand the padding properties in CSS.

Syntax

```
padding : padding_width
```

Code Snippet

- <?xml version="1.0" encoding="UTF-8"?>
- <?xmlstylesheet href="Padding.css" type="text/css"?>
- <Cars>
- <Vehicle>
- <Manufacturer>BMW</Manufacturer>
- <Model>M3</Model>
- <Color>Metallic Silver</Color>
- <Price>40,000</Price>
- <Location>Liverpool</Location>
- </Vehicle>
- </Cars>

© Aptech Limited XML Simplified/Module 5/36 of 53

Style Sheet

```

1 Manufacturer {border: thick dashed magenta }
2 Model { border: thick solid olive }
3 Color { border: thick groove aqua }
4 Price { border: thick inset gray }
5 Manufacturer { padding: 2 0 }
6 Model { padding: 2 0 }
7 Color { padding: 2 5 0 1 }
8 Price { padding: 2 5 8 10 }

```

where,

- Manufacturer { padding: 2 } Inserts padding of 2 pixels between the four borders and text of Manufacturer element. The four borders applicable are top, right, bottom, and left.
- Model { padding: 2 0 } Inserts padding between the borders and text of Model element. The value 2 is applied to top and bottom borders and 0 is applied to left and right borders.
- Color { padding: 2 5 0 1 } Inserts padding between the borders and text of Color element. The value 2 is applied to top border, 5 to left and right borders, and value 0 to bottom border.
- Price { padding: 2 5 8 10 } Inserts padding between the borders and text of Price element. The value 2 is applied to top border, 5 to right border, 8 to bottom border, and 10 to left border.

Output

BMW M3 Metallic Silver 40,000 Liverpool

© Aptech Limited XML Simplified/Module 5/37 of 53

Using slides 36 and 37, explain the padding properties in CSS.

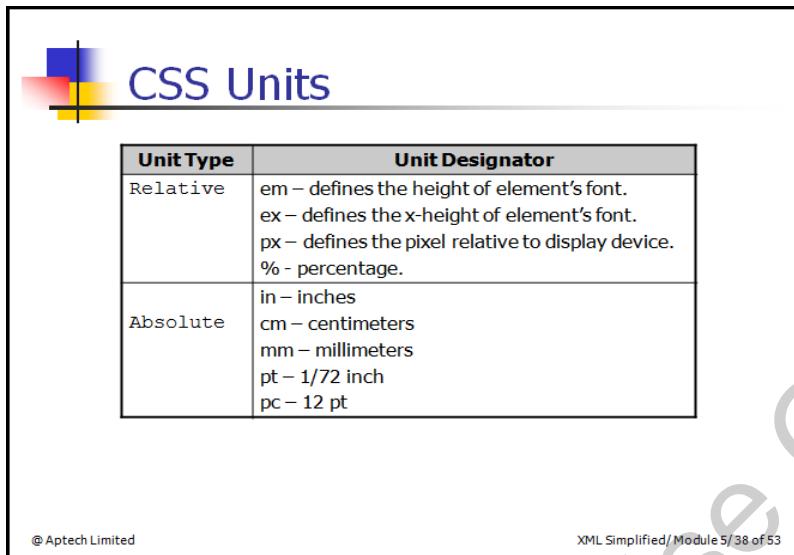
The border property surrounds the text in an element with an outline. To insert space between the border and the text of an element, CSS provides the `padding` property.

Explain the figure showing the syntax for `padding` property and the code for style rules defined in `Padding.css` file explained on slides 36 and 37.

Then, explain the figure on slide 37 that depicts the style sheet for `Padding.css` file.

Slide 38

Let us understand the CSS units.



CSS Units

Unit Type	Unit Designator
Relative	em – defines the height of element's font. ex – defines the x-height of element's font. px – defines the pixel relative to display device. % - percentage.
Absolute	in – inches cm – centimeters mm – millimeters pt – 1/72 inch pc – 12 pt

© Aptech Limited XML Simplified/ Module 5/ 38 of 53

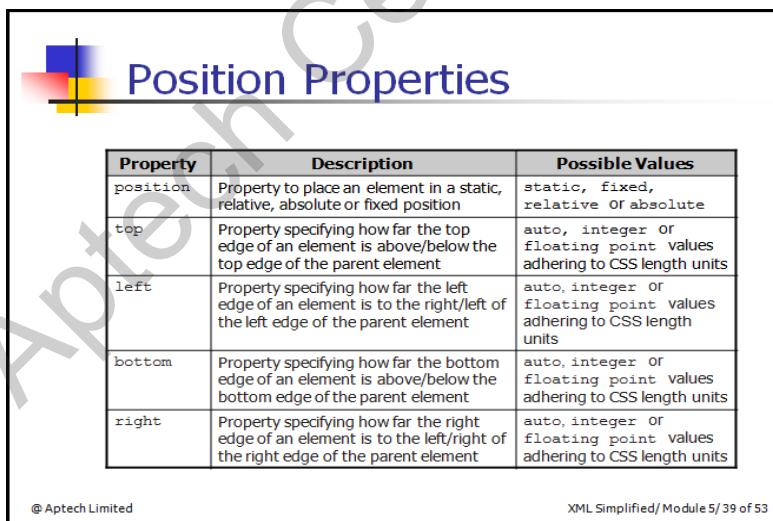
Using slide 38, explain the units in CSS.

The values assigned to CSS properties are expressed in length units.

Explain the table that lists the units, in slide 38.

Slide 39

Let us understand the position properties.



Position Properties

Property	Description	Possible Values
position	Property to place an element in a static, relative, absolute or fixed position	static, fixed, relative OR absolute
top	Property specifying how far the top edge of an element is above/below the top edge of the parent element	auto, integer or floating point values adhering to CSS length units
left	Property specifying how far the left edge of an element is to the right/left of the left edge of the parent element	auto, integer or floating point values adhering to CSS length units
bottom	Property specifying how far the bottom edge of an element is above/below the bottom edge of the parent element	auto, integer or floating point values adhering to CSS length units
right	Property specifying how far the right edge of an element is to the left/right of the right edge of the parent element	auto, integer or floating point values adhering to CSS length units

© Aptech Limited XML Simplified/ Module 5/ 39 of 53

Using slide 39, explain the position properties.

Every element's text is placed in a box of its own. Table lists the CSS positioning properties to position the text inside the box. Note that the properties top, left, bottom, and right are used only if value of position property is not static.

Slides 40 to 42

Let us understand the position properties example.

Position Properties Example 1-3

Code Snippet

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet href="Position.css" type="text/css"?>
3 <Cars>
4   <Vehicle>
5     <Manufacturer>BMW</Manufacturer>
6     <Model>M3</Model>
7     <Color>Metallic Silver</Color>
8     <Price>40,000</Price>
9     <Location>Liverpool</Location>
10    </Vehicle>
11 </Cars>
```

© Aptech Limited XML Simplified/Module 5/ 40 of 53

Position Properties Example 2-3

Style Sheet

```

1 Location { position: relative; top: 20; left: 20 }
2 Price { position: absolute; top: 40; left: 20 }
```

where,
 Location {position:relative;top:20;left:20}
 Positions the text of Location element relative to the previous element.
 However, inside the box, the content is placed at 20 pixels from top and 20 pixels from left.
 Price {position:absolute;top:40;left:20}
 Positions the text of Price element at an absolute location of 40 pixels from the top and 20 pixels from the left.

© Aptech Limited XML Simplified/Module 5/ 41 of 53

Position Properties Example 3-3

Output

BMW M3 Metallic Silver
 40,000 Liverpool

where,
 40,000
 Text displayed using absolute positioning
 Liverpool
 Text displayed using relative positioning

© Aptech Limited XML Simplified/Module 5/ 42 of 53

Using slides 40 to 42, explain the example of position properties.

Explain the figure on slide 40 that explains the style rules defined in Position.css file.
 Also, explain the figure on slide 41 that depicts the style sheet for Position.css file.

Slides 43 to 45

Let us understand the display properties.

Display Property 1-3

Syntax

```
display : value
```

where value = none | inline | block

where,

- display Property to specify how the element is to be rendered.
- none No rendering is applied to the element.
- inline Displays the element text as in line. This is the default value if no value is specified.
- block Displays the element text on a new line in a block of its own.

© Aptech Limited XML Simplified/Module 5/ 43 of 53

Display Property 2-3

Code Snippet

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet href="Display.css" type="text/css"?>
3 <Cars>
4   <Vehicle>
5     <Manufacturer>BMW</Manufacturer>
6     <Model>M3</Model>
7     <Color>Metallic Silver</Color>
8     <Price>40,000</Price>
9     <Location>Liverpool</Location>
10    </Vehicle>
11   <Vehicle>
12     <Manufacturer>TOYOTA</Manufacturer>
13     <Model>INNOVA</Model>
14     <Color>Gray</Color>
15     <Price>38,000</Price>
16     <Location>California</Location>
17   </Vehicle>
18 </Cars>
```

© Aptech Limited XML Simplified/Module 5/ 44 of 53

Display Property 3-3

Style Sheet

```
1 Price { display: block }
```

where,

```
Price { display: block }
```

Inserts a new line before and after the text of element Price.

Output

```
BMW
M3 Metallic Silver
40,000
Liverpool
TOYOTA
INNOVA Gray
38,000
California
```

© Aptech Limited XML Simplified/Module 5/ 45 of 53

Using slides 43 to 45, explain the display properties.

In HTML, if you wanted the text to appear as new paragraph, you would use the `<P>` tag. The same can be achieved in XML by using CSS display property.

Explain the figure that shows the syntax for `display` property and also explain the code for style rules defined in `Display.css`.

Then, explain the figure on slide 45 that depicts the style sheet for `Display.css` file.

Data in each element is displayed on a separate row in block of its own.

Slides 46 to 48

Let us understand the text alignment and indentation.

Syntax

```
text-align : alignment_value
text-indent : value
```

where,

- text-align**
Property to align the text in a block.
- alignment_value**
Can be one of the following: left(default), right, center and justify.
- text-indent**
Property to indent the text in a block.
- value**
Floating point value followed by absolute units designators or relative units designators; or an integer value followed by percentage (%) symbol.

© Aptech Limited XML Simplified/ Module 5/ 46 of 53

Code Snippet

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet href="Align.css" type="text/css"?>
3  <Cars>
4    <Vehicle>
5      <Manufacturer>BMW</Manufacturer>
6      <Model>M3</Model>
7      <Color>Metallic Silver</Color>
8      <Price>40,000</Price>
9      <Location>Liverpool</Location>
10     </Vehicle>
11    <Vehicle>
12      <Manufacturer>TOYOTA</Manufacturer>
13      <Model>INNOVA</Model>
14      <Color>Gray</Color>
15      <Price>38,000</Price>
16      <Location>California</Location>
17    </Vehicle>
18  </Cars>

```

© Aptech Limited XML Simplified/ Module 5/ 47 of 53



Text Alignment and Indentation 3-3

Style Sheet

```

1 Price { display: block }
2 Price { text-align: left }
3 Price { text-indent: 20 }

```

where,
 Price {display: block}
 Inserts a new line before and after the text of element Price and displays the text in a separate block.
 Price {text-align: left}
 Left aligns the text of Price element.
 Price {text-indent: 20}
 Indents the text of Price element at 20 pixels.

Output

BMW M3 Metallic Silver	40,000
Liverpool TOYOTA INNOVA Gray	38,000
	California

where,
 40,000
 The text in element Price is displayed on a separate row and left indented at 20 pixels.

© Aptech Limited

XML Simplified/Module 5/ 48 of 53

Using slides 46 to 48, explain the text alignment and indentation.

The `display:block` property renders the text of an element in a separate block. Inside this block, you can align and indent the text using the CSS properties `text-align` and `text-indent` respectively.

Explain the figure on slide 46 that shows the syntax for text alignment and indentation.

Then, explain the figure on slide 47 shows the code for style rules defined in `Align.css`. Also, the figure on slide 48 depicts the style sheet for `Align.css` file.

The text in element `Price` is displayed on a separate row and left indented at 20 pixels.

Slide 49

Let us understand the objectives of lesson 4.



Lesson 4 - Inheritance and Cascades in CSS

In this last lesson, **Inheritance and Cascades in CSS**, you will learn to:

- Define the process of cascading.
- Explain inheritance.

© Aptech Limited

XML Simplified/Module 5/ 49 of 53

Using slide 49, explain the objectives of lesson 4.

Slide 50

Let us understand the cascading in CSS.



Cascading in CSS

- All declarations that apply to an element and property are grouped.
- Declarations are sorted by weight and origin.
- Declarations are sorted by specificity of selector.
- Declarations are sorted by the order specified.

© Aptech Limited

XML Simplified/ Module 5/ 50 of 53

Using slide 50, explain the cascading in CSS.

There are several ways in which style sheets can be defined. For example, one document can have more than one style sheet defined for it. A browser has its own style sheet. In such cases, it is possible that more than one style rule may exist for an element. Hence, the World Wide Web Consortium (W3C) defines the following rules to determine what style to apply to an element. The sorting of style rules begins with rule number 1. If the style rule matches rule number 1, then the search is over; otherwise the search continues with rule number 2 and so on.

1. Find all property declarations for an element in question. Apply the style rules if the element name matches the element in the selector. If there is no style rule defined for an element, the element inherits the style rules defined for the parent element. However, if the parent element does not have any style rules defined, then the element is rendered with default values.
2. Style rules declared as important are considered next. A style rule can be declared important in the following manner:

```
Threat { display: block ! important }
```

Important property declarations have a higher precedence over normal property declarations.

3. Next, the origin of style sheet is determined. A style sheet can have following sources:
 - **Author:** The author of the XML document can define a style sheet in an external document to format the XML document.
 - **User/Reader:** The end-user viewing the XML document can specify his/her personal style sheet to be used to format the XML document.

- **User-agent:** The browser is referred to as user-agent. A browser has its own default style sheet.

The style rules defined in the author's style sheet override the style rules defined in the user's style sheet. The author's and the user's style sheet both override the user-agent's style sheet.

4. Next, the specificity of a selector is determined in that a more specific selector will override the general selector. The specificity is determined by carrying out the following three activities:
 - Count the number of ID attributes in the selector.
 - Count the number of class attributes in a selector.
 - Count the number of element names in the selector.
5. Next, write the three numbers with no commas or spaces and in the same sequence as shown earlier. Higher the number, higher is the specificity. Rules with higher specificity override the ones with lower specificity. For example, following is a list of selectors sorted by specificity:

```
#favorite {...}          /* a=1;b=0;c=0; specificity = 100 */
Name, CD, Artist.caps {...} /* a=0;b=1;c=3; specificity = 013 */
Artist.caps {...}           /* a=0;b=0;c=1; specificity = 001 */
```

6. If two rules have the same weight, then the one specified last wins.

Slides 51 and 52

Let us understand the inheritance in CSS.

Inheritance in CSS 1-2

- Inheritance is the ability of one entity to acquire the characteristics of another entity.
- In CSS, the child elements inherit the styles rules defined for parent element.
- However, this is applicable only if the child has no explicit style rule defined for it.
- Otherwise, the style rule defined for child element overrides the style rule defined for parent element.



Inheritance in CSS 2-2

XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="CD.css" type="text/css"?>
<CD>
  <Audio>
    <Name>2007 Hits</Name>
  </Audio>
</CD>
```

Style Sheet

```
CD { font-style: italic }
```

Output

2007 Hits

@ Aptech Limited XML Simplified/ Module 5/ 52 of 53

Using slides 51 and 52, explain the inheritance in CSS.

Inheritance is the ability of one entity to acquire the characteristics of another entity. In CSS, the child elements inherit the style rules defined for parent element. However, this is applicable only if the child has no explicit style rule defined for it. Otherwise, the style rule defined for child element overrides the style rule defined for parent element.

Figure on slide 52 shows that the text in element Name is displayed in italics. This is because there are no style rules defined for elements Audio and Name. Hence, the style rule defined for CD is inherited by child element Name.

Slide 53

Let us summarize the session.



Summary

- **Style Sheets**
 - Style Sheets are a set of rules that define the appearance of data.
 - These rules are written in a file with the extension .css.
 - The .css file is associated with an XML document using the xml-processing instruction.
- **Selectors in CSS**
 - Selectors define the elements to which the styles will be applied.
 - The various types of selectors are simple, universal and ID selectors.
- **Properties and Values**
 - CSS provides several properties to define the appearance of data.
 - Some of these properties are color, background-color, position, padding, font, text-align to name a few.
- **Inheritance and Cascades in CSS**
 - In CSS, a child element inherits the styles rules applied to its ancestor element.
 - Also there are several sources of style sheets, hence CSS follows W3C defined cascading order when applying style rules to elements.

@ Aptech Limited XML Simplified/ Module 5/ 53 of 53

In slide 53, summarize the session. End the session, with a brief summary of what has been taught in the session.

5.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the XSL and XSLT that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 6 - XSL and XSLT

6.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

6.1.1 Objectives

By the end of this session, the learners will be able to:

- Introduction to XSL
- Working with XSL

6.1.2 Teaching Skills

To teach this session, you should be well-versed with XSL and XSLT. You should aware yourself with the techniques of transforming XML documents into XML, HTML, and Text documents using XSL stylesheets.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Then give the students the overview of the current session in the form of Session Overview. Show the students slide 2 of the presentation.

Tell the students that this session introduces the process of transforming XML documents into XML, HTML, and text documents using Extensible Stylesheet Language (XSL). It also explains the difference between XSL and CSS.

6.2 In-Class Explanations

Slide 3

Let us understand the objectives of lesson 1.



Lesson 1 – Introduction to XSL

In this first lesson, **Introduction to XSL**, you will learn to:

- Define Extensible Stylesheet Language (XSL), Extensible Stylesheet Language Transformations (XSLT) and their purpose.
- Explain the structure and syntax of XSL.
- Distinguish between Cascading Style Sheet (CSS) and XSL.

© Aptech Limited XML Simplified/ Module 6/ 3 of 53

Using slide 3, explain the topics covered in lesson1.

Slides 4 and 5

Let us understand the stylesheets.

Stylesheets 1-2

- It is a collection of commands that tells a processor how to render the visual appearance of content in a web page.
- Stylesheets typically contain instructions like:
 - Number figures sequentially throughout the document.
 - Display hypertext links in blue.
 - Position text and images on a Web page.
 - Create a layout optimized for small displays for mobile phones.

© Aptech Limited XML Simplified/ Module 6/ 4 of 53

Stylesheets 2-2

- CSS is used for HTML content formatting.
- XSL is used to describe how the XML document should be displayed.

© Aptech Limited XML Simplified/ Module 6/ 5 of 53

Using slides 4 and 5, explain the stylesheets.

A stylesheet is a collection of commands that tells a processor (such as a Web browser, or a document reader) how to render the visual appearance of content in a Web page.

Stylesheets typically contain instructions such as:

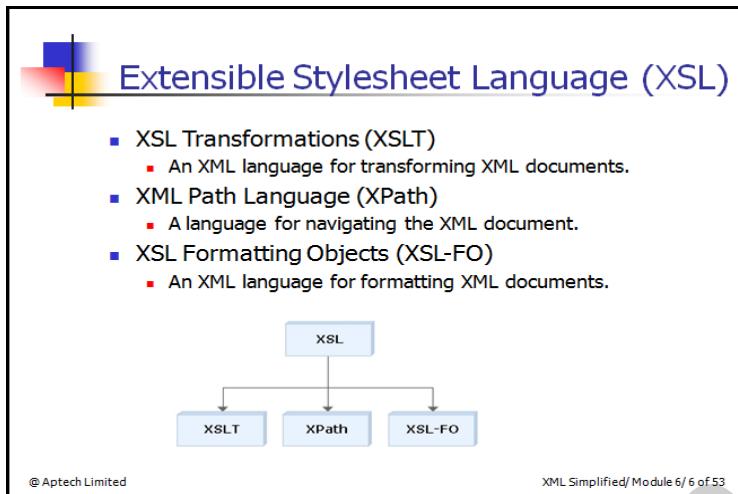
- Number figures sequentially throughout the document.
- Display hypertext links in blue.
- Position text and images on a Web page.
- Create a layout optimized for small displays for mobile phones.

CSS is a stylesheet technology that is used for HTML content formatting.

Extensible Stylesheet Language (XSL) is developed by W3C to describe how the XML document should be displayed.

Slide 6

Let us understand the XSL.



Using slide 6, explain XSL.

XSL is an XML-based language used to create stylesheets. XSL is designed to format XML content for display purposes and also has the ability to completely transform XML documents. XSL is a specification from the World Wide Web Consortium (W3C). XSL Transformation Recommendation describes the process of transforming an XML document, using a transformation engine and XSL. The XML document and XSL stylesheet are provided as input to the XML transformation engine, also known as the XSLT processor. The output of the processor is in the form of a result tree in which the elements of the XML document are converted into nodes having specific text values and attributes. XSLT processors use XSL stylesheets to gather instructions for transforming source XML documents to output XML documents. Stylesheets are used to change the structure of the XML document and modify the output.

XSL consists of three languages as follows:

- XSL Transformations (XSLT)

XSLT transforms an XML document into another XML document that can actually be understood and displayed by a computer.

- XML Path Language (XPath)

XPath is used as the navigator for XSL. XSL uses XPath to find parts of the source document that should match a certain predefined template. When XPath finds the node, XSLT takes over and performs a transformation, turning the source document into a result document.

- XSL Formatting Objects (XSL-FO)

An XML language for formatting XML documents. Once XPath has searched through the source document and used XSLT to transform the source document into the result document, the document then needs to be formatted so that the Web browser will be able

to present the document with the proper layout and structure. Simply put, XSL-FO is the part of XSL that produces the final output.

Figure on slide 6 depicts the structure of XSL.

Slide 7

Let us understand the XSL transformations.

XSL Transformations

- The transformation component of the XSL stylesheet technology is XSLT.
- It describes the process of transforming an XML document, using a transformation engine and XSL.

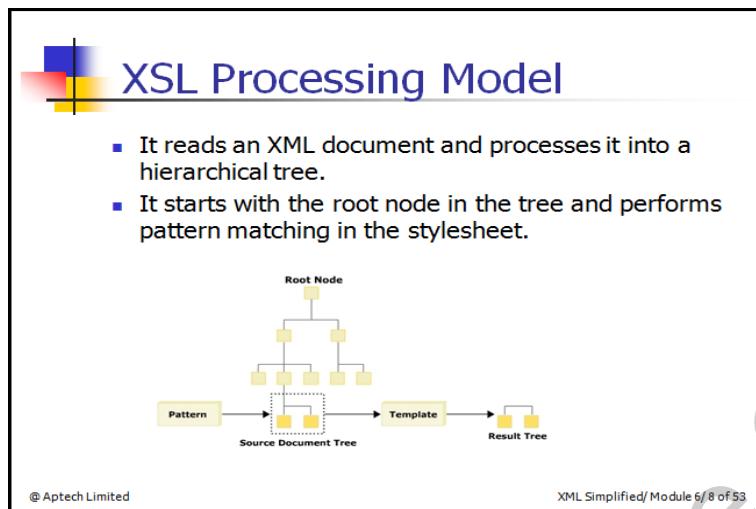
@ Aptech Limited XML Simplified/ Module 6/ 7 of 53

Using slide 7, explain the XSL transformations.

The transformation component of the XSL stylesheet technology is XSLT. Its purpose is to transform XML documents. It describes the process of transforming an XML document, using a transformation engine and XSL. The XML document and XSL stylesheet are provided as input to the XML transformation engine, also known as the XSLT processor. The output of the processor is in the form of a result tree in which the elements of the XML document are converted into nodes having specific text values and attributes.

Slide 8

Let us understand the XSL processing model.



Using slide 8, explain the XSL processing model.

The XML processor reads an XML document and processes it into a hierarchical tree containing nodes for each piece of information in a document. After a document has been processed into a tree, the XSL processor begins applying the rules of an XSL stylesheet to the Document tree.

The XSL processor starts with the root node in the tree and performs pattern matching in the stylesheet. These patterns are stored within constructs known as templates in XSL stylesheets. The XSL processor analyzes templates and the patterns associated with them to process different parts of the Document tree. When a match is made, the portion of the tree matching the given pattern is processed by the appropriate stylesheet template. At this point, the rules of the template are applied to the content to generate a result tree. The result tree is itself a tree of data and the data has been transformed by the stylesheet.

Figure on slide 8 displays the XSL processing model.

Slide 9

Let us understand the XSLT structure and syntax.



XSLT Structure and Syntax

Using slide 9, explain the XSLT structure and syntax.

XSLT follows normal rules of XML syntax. It uses a standard document introduction, matching closing tags for any opening tags that contain content, and a proper syntax for empty elements.

In XSL, the style rules are written in a file with the extension .xsl. This file is associated with an XML document by using the statement <?xml-stylesheet href="xsl file" type="text/xsl"?>. Here, the xml-stylesheet is the processing instruction.

Slide 10

Let us understand the top-level XSLT elements.

Top Level XSLT Elements

- An element occurring as a child of an `xsl:stylesheet` element is called a top-level element.
- It can occur directly inside the `xsl:stylesheet` element.

Element Name	Description
<code>xsl:attribute-set</code>	Adds a list of attributes to the output node tree
<code>xsl:import</code>	Used to import contents of one stylesheet into another. The importing stylesheet takes precedence over the imported stylesheet
<code>xsl:namespace-alias</code>	Replaces source document Namespace with a new Namespace in the output tree node
<code>xsl:output</code>	Specifies the output for the result tree. It contains a list of attributes. The most important one is the <code>method</code> attribute which dictates if the type of output is <code>HTML</code> , <code>text</code> , or <code>XML</code> .
<code>xsl:template</code>	Used to define a template that can be applied to a node to produce a desired output display
<code>xsl:variable</code>	Defines a <code>variable</code> in a <code>stylesheet</code> or <code>template</code> , and to assign it a value

© Aptech Limited

XML Simplified/ Module 6/ 10 of 53

Using slide 10, explain the top-level XSLT elements.

The top-level XSLT elements can occur directly inside the `xsl:stylesheet` element. An element occurring as a child of an `xsl:stylesheet` element is called a top-level element. These elements provide the building blocks for creating XSLT documents.

The top-level elements in the XSLT syntax are listed in table on slide 10. These elements can occur directly inside the `xsl:stylesheet` or `xsl:transform` elements.

Slide 11

Let us understand the CSS and XSL.



CSS and XSL

- They are two different style languages recommended by the World Wide Web Consortium (W3C).
- XSL is more powerful and complex than CSS.

CSS	XSL
Stylesheet language to create a style for HTML and XML documents	Stylesheet language to create a style for XML documents
Determines the visual appearance of a page, but does not alter the structure of the source document	Provides a means of transforming XML documents
Does not support decision structures and it cannot calculate quantities or store values in variables	Supports decision structures and can calculate quantities or store values in variables
Uses its own notation	Uses an XML notation
Highly effective and easy to learn for simple applications	Designed to meet the needs of more complex applications for richer style sheets

© Aptech Limited

XML Simplified/ Module 6/ 11 of 53

Using slide 11, explain the CSS and XSL.

XSL and CSS are two different style languages recommended by the W3C. XSL is more powerful and complex than CSS. Some of the differences between CSS and XSL are listed in table on slide 11.

Slide 12

Let us understand the objectives of lesson 2.



Lesson 2 – Working with XSL

In this last lesson, **Working with XSL**, you will learn to:

- Explain XSL templates.
- Describe the use of `select` attribute.
- State how to use `xsl:value-of` element.
- Describe how to use `xsl:for-each` element.
- Explain briefly how to use `xsl:text` element.
- Describe how to use `xsl:number` element.
- Describe how to use `xsl:if` element.
- Describe how to use `xsl:choose` element.
- Explain how to perform sorting using XSL.

© Aptech Limited

XML Simplified/ Module 6/ 12 of 53

Using slide 12, explain the topics covered in lesson 2.

Slide 13

Let us understand the XSL templates.

The slide title is "XSL Templates". It displays an XML document and its corresponding XSLT transformation. The XML document contains customer information. The XSLT code uses `<xsl:for-each>` loops to iterate over customers and their orders, applying templates to output HTML table rows. A red box highlights the XSLT template rule for the first name. To the right, the resulting HTML table is shown, displaying two rows of customer data: Blake, David and Honai, John.

```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="customer.xsl"?>
<CustomerList>
  <Customer>
    <Name>
      <First>David</First>
      <Last>Blake</Last>
    </Name>
    <Order>100 brown suitcases</Order>
    <Order>12 bottles wine</Order>
  </Customer>
<CustomerList>
  <Customer>
    <Name>
      <First>Honai</First>
      <Last>John</Last>
    </Name>
    <Order>120 red T-shirts</Order>
    <Order>120 bottles mango juice</Order>
  </Customer>
</CustomerList>

```

Blake , David
100 brown suitcases
12 bottles wine
Honai , John
120 red T-shirts
120 bottles mango juice

Using slide 13, explain the XSL templates in details.

A template is the main component of a stylesheet. Templates are defined with the help of rules. A template rule is used to control the output of the XSLT processor. It defines the method by which an XML element node is transformed into an XSL element node. A template rule consists of a pattern that identifies the XML node and an action that selects and transforms the node.

Each template rule is represented by the `xsl:template` element. The `xsl:template` is an element that defines an action for producing output from a source document.

Explain the figure on slide 13 shows an example of XSL template.

Slides 14 and 15

Let us understand the `xsl:template` element.

The slide title is "The xsl:template Element 1-2". It includes a bulleted list stating that it is used to define a template that can be applied to a node to produce desired output. Below this is a "Syntax" section with the XML structure of the `<xsl:template>` element. A "where" section provides detailed explanations for each attribute: `match`, `mode`, `name`, and `priority`.

```

<xsl:template
  match="pattern"
  mode="mode"
  name="name"
  priority="number"
>
</xsl:template>

```

where,

- `match`: Is a pattern that is used to define which nodes will have which template rules applied to them. If this attribute is omitted there must be a `name` attribute.
- `mode`: Allows the same nodes to be processed more than once.
- `name`: Specifies a name for the template. If this attribute is omitted there must be a `match` attribute.
- `priority`: Is a real number that sets the priority of importance for a template. The higher the number, the higher the priority.



The xsl:template Element 2-2

Code Snippet

```

1  <?xml version="1.0"?
2  <xsl:stylesheet version="1.0"
3      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4  <xsl:template match="/">
5      <html>
6          <body>
7              <h1> XSL TEMPLATE SAMPLE</h1>
8          </body>
9      </html>
10     </xsl:template>
11 </xsl:stylesheet>

```

© Aptech Limited XML Simplified/ Module 6/ 15 of 53

Using slides 14 and 15, explain the `xsl:template` element.

The `xsl:template` element is used to define a template that can be applied to a node to produce desired output.

The `match` attribute in `xsl:template` is used to associate the template with an XML element. You can also define a template for a whole branch of the XML document by using the `match` attribute (for example, `match="/"` defines the whole XML document).

Explain the syntax of the `xsl:template` element using slide 14.

Explain the code snippet which demonstrates the usage of `xsl:template` element in stylesheets on slide 15.

An XSL stylesheet is an XML document itself, it always begins with the XML declaration : `<?xml version="1.0" ?>`. The next element, `<xsl:stylesheet>`, defines that this document is an XSLT stylesheet document. It also contains the version number and XSLT namespace attributes.

The `<xsl:template>` element defines a template. The `match="/"` attribute associates the template with the root of the XML source document.

The content inside the `<xsl:template>` element defines some HTML to write the output. The last two lines define the end of the template and the end of the stylesheet.

Slides 16 to 19

Let us understand the `xsl:apply-templates` element.

The `xsl:apply-templates` element 1-4

- It defines a set of nodes to be processed.

Syntax

```
<xsl:apply-templates
    select="expression"
    mode="name"
>
</xsl:apply-templates>
```

where,
`select`
Used to process nodes selected by an expression.
`mode`
Allows the same nodes to be processed more than once. Each time the nodes are processed, they can be displayed in a different manner.

The `xsl:apply-templates` element 2-4

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xmlstylesheet type="text/xsl" href="GEM_Stylesheet.xsl"?>
3  <GEMEmployees>
4      <Designer>
5          <Name>David Blake</Name>
6          <DOJ>18/11/1973</DOJ>
7          <Address>512-B Lamington Road</Address>
8          <Phone>1564-754-111</Phone>
9      </Designer>
10     <Designer>
11         <Name>Susan Jones</Name>
12         <DOJ>03/05/1953</DOJ>
13         <Address>Palm Beach Road</Address>
14         <Phone>8755-211-111</Phone>
15     </Designer>
16 </GEMEmployees>
```



The xsl:apply-templates element 3-4

Style Sheet

```

1 <xslstylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
2   <xsltemplate match="GEMEmployees/Designer">
3     <body>
4       <xslapply-templates select="Name"/>
5       <xslapply-templates select="DOJ"/>
6     </body>
7   </xsltemplate>
8   <xsltemplate match="Name">
9     Name:
10    <span style="font-size:22px; color:green">
11      <xslvalue-of select="."/>
12    </span>
13  </xsltemplate>
14  <xsltemplate match="DOJ">
15    Date of Join:
16    <span style="color:blue;">
17      <xslvalue-of select="."/>
18    </span>
19  </xsltemplate>
20 </xslstylesheet>

```

where:
 match="GEMEmployees/Designer"
 Represents the Designer child element by specifying the GEMEmployees parent element.
 xsl:apply-templates select="Name"
 Applies the template Name element.
 xsl:apply-templates select="DOJ"
 Applies the template on DOJ element.
 xsl:value-of
 Used to extract the value of selected node.
 xsl:template match="Name"
 If the template is matched with Name element, the value of the Name element is displayed in green color with font size 22 pixels.
 xsl:template match="DOJ"
 If the template is matched with DOJ element, the value of the DOJ element is displayed in blue color.

© Aptech Limited XML Simplified/ Module 6/ 18 of 53



The xsl:apply-templates element 4-4

Output

Name: David Blake
Date of Join: 18/11/1973

Name: Susan Jones
Date of Join: 03/05/1953

© Aptech Limited XML Simplified/ Module 6/ 19 of 53

Using slides 16 to 19, explain the `xsl:apply-templates` element.

The `xsl:apply-templates` element defines a set of nodes to be processed. This element, by default, selects all child nodes of the current node being processed, and finds a matching template rule to apply to each node in the set.

Explain the figure on slide 16 that shows the syntax for `xsl:apply-templates`.

Explain the figure on slide 17 that shows the code for style rules that are defined in `GEM_Stylesheet.xsl` file.

Explain the figure on slide 18 depicts the `GEM_Stylesheet.xsl` file.

Slides 20 to 23

Let us understand the `select` attribute.

The select attribute 1-4

- It can be used to process nodes selected by an expression instead of processing all children.

Syntax

```
<xsl:template match = "element">
  <xsl:apply-templates select = "name of the element"/>
</xsl:template>
```

where,
select
 Uses the same kind of patterns as the `match` attribute of the `xsl:template` element. If `select` attribute is not present, all child element, comment, text, and processing instruction nodes are selected.

© Aptech Limited XML Simplified/ Module 6/ 20 of 53

The select attribute 2-4

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xslstylesheet type="text/xsl" href="Book_stylesheet.xsl">
3  <xsl:catalog>
4  <xsl:book>
5    <xsl:book>
6      <xsl:title>XML By Example</xsl:title>
7      <xsl:author>David Blake</xsl:author>
8      <xsl:price>$20.90</xsl:price>
9      <xsl:year>1990</xsl:year>
10 </xsl:book>
11 <xsl:book>
12   <xsl:title>XML Bible 1.1</xsl:title>
13   <xsl:author>David Trott</xsl:author>
14   <xsl:price>$53</xsl:price>
15   <xsl:year>2004</xsl:year>
16 </xsl:book>
17 <xsl:book>
18   <xsl:title>XML Cookbook</xsl:title>
19   <xsl:author>Susan Jones</xsl:author>
20   <xsl:price>$11.10</xsl:price>
21   <xsl:year>1995</xsl:year>
22 </xsl:book>
23 </xsl:catalog>
```

© Aptech Limited XML Simplified/ Module 6/ 21 of 53

The select attribute 3-4

Style Sheet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xslstylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3  <xsl:template match="/">
4    <html>
5      <body>
6        <h1>Popular XML Books</h1>
7        <xsl:apply-templates/>
8      </body>
9    </html>
10 </xsl:template>
11 <xsl:template match="Book">
12   <p>
13     <xsl:apply-templates select="Title"/>
14     <xsl:apply-templates select="Author"/>
15   </p>
16 </xsl:template>
17 <xsl:template match="Title">
18   Title:
19   <span style="color:green;font-size:20pt">
20     <xsl:value-of select=""/>
21   </span>
22   <br/>
23 </xsl:template>
24 <xsl:template match="Author">
25   Author:
26   <span style="color:red;font-size:15pt">
27     <xsl:value-of select=""/>
28   </span>
29   <br/>
30 </xsl:template>
31 </xslstylesheet>
```

where,
select="Title"
 Applies the template on `Title` element.
select="Author"
 Applies the template on `Author` element.

© Aptech Limited XML Simplified/ Module 6/ 22 of 53

The slide has a decorative graphic in the top-left corner consisting of overlapping colored squares (blue, red, yellow). The title 'The select attribute 4-4' is in large blue font. Below it is a green button-like box containing the word 'Output'. The main content is titled 'Popular XML Books' in bold black font. It lists three books with their titles in green and authors in red:

- Title: XML By Example
Author: David Blake
- Title: XML Bible 1.1
Author: David Troff
- Title: XML Cookbook
Author: Susan Jones

At the bottom left is the copyright notice '@ Aptech Limited' and at the bottom right is 'XML Simplified/ Module 6/ 23 of 53'.

Using slide 20, explain the `select` attribute.

The `select` attribute can be used to process nodes selected by an expression instead of processing all children. The `xsl:apply-templates` with a `select` attribute can be used to choose a particular set of children instead of all children.

Explain the figure on slide 20 shows the syntax for `select` attribute.

It uses the same kind of patterns as the `match` attribute of the `xsl:template` element. If `select` attribute is not present, all child element, comment, text, and processing instruction nodes are selected.

Explain the figure on slide 21 shows the code for style rules that are defined in `book_stylesheet.xsl` file. Also the figure on slide 22 that depicts the stylesheet for `book_stylesheet.xsl` file. Explain the code.

Slides 24 to 27

Let us understand the `xsl:value-of` element.



The `xsl:value-of` element 1-4

- It is used to write or display text string representing the value of the element specified by the `select` attribute.

Syntax

```
<xsl:value-of
    select="expression"
    disable-output-escaping="yes" | "no"
/>
```

where,
select
A mandatory attribute that assigns the node (by name) to the element.
disable-output-escaping
Specifies how special characters should appear in the output string.
yes
Indicates that special characters should be displayed as is (for example, a < or >).
no
Indicates that special characters should not be displayed as is (for example, a > is displayed as >).

@ Aptech Limited XML Simplified/ Module 6/ 24 of 53



The `xsl:value-of` element 2-4

Code Snippet

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="person_stylesheet.xsl"?>
3  <House>
4      <Person>
5          <FirstName age="20">David</FirstName>
6          <LastName>Blake</LastName>
7      </Person>
8      <Person>
9          <FirstName age="34">Susan</FirstName>
10         <LastName>Jones</LastName>
11     </Person>
12     <Person>
13         <FirstName>Martin</FirstName>
14         <LastName>King</LastName>
15     </Person>
16     <Person>
17         <FirstName>Justin</FirstName>
18         <LastName>Nora</LastName>
19     </Person>
20 </House>

```

@ Aptech Limited XML Simplified/ Module 6/ 25 of 53



The xsl:value-of element 3-4

Style Sheet

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3    <xsl:template match="Person">
4      <p>
5        <xsl:value-of select="FirstName"/>
6        ,
7        <xsl:value-of select="LastName"/>
8      </p>
9    </xsl:template>
10   </xsl:stylesheet>

```

where,
`xsl:value-of select="FirstName"`
 Display the value of the element FirstName.
`xsl:value-of select="LastName"`
 Display the value of the element LastName.

@ Aptech Limited XML Simplified/ Module 6/ 26 of 53



The xsl:value-of element 4-4

Output

```

David , Blake
Susan , Jones
Martin , King
Justin , Nora

```

@ Aptech Limited XML Simplified/ Module 6/ 27 of 53

Using slides 24 to 27, explain the `xsl:value-of` element.

The `xsl:value-of` element is used to write or display in the result tree a text string representing the value of the element specified by the `select` attribute. A `xsl:value-of` element can only have one node assigned to it.

Explain the figure on slide 24 that shows the syntax for `xsl:value-of` element.

Explain the figure on slide 25 shows the code for style rules that are defined in `person_stylesheet.xsl` file. Explain the figure on slide 26 depicts the stylesheet for `person_stylesheet.xsl` file.

Slides 28 to 31

Let us understand the `xsl:for-each` element.



The `xsl:for-each` element 1-4

- It can be used to iterate through the XML elements of a specified node set.

Syntax

```
<xsl:for-each select="expression">
</xsl:for-each>
```

where,
`select="expression"`
 The expression is evaluated on the current context to determine
 the set of nodes to iterate over.

@ Aptech Limited XML Simplified/Module 6/ 28 of 53



The `xsl:for-each` element 2-4

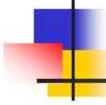
Code Snippet

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="APTEmployees_stylesheet.xsl"?>
3  <Employees>
4  <Employee>
5   <Name>John Thorp</Name>
6   <Department>Marketing</Department>
7   <Language>EN</Language>
8   <Salary>$1000</Salary>
9  </Employee>
10 <Employee>
11  <Name>David Blake</Name>
12  <Department>Admin</Department>
13  <Language>GR</Language>
14  <Salary>$1200</Salary>
15 </Employee>
16 <Employee>
17  <Name>Ben Johns</Name>
18  <Department>Physical Education</Department>
19  <Language>TR</Language>
20  <Salary>$800</Salary>
21 </Employee>
22 <Employee>
23  <Name>Susan Lopez</Name>
24  <Department>External Affairs</Department>
25  <Language>EN</Language>
26  <Salary>$2800</Salary>
27 </Employee>
28 </Employees>

```

@ Aptech Limited XML Simplified/Module 6/ 29 of 53



The xsl:for-each element 3-4

Style Sheet

```

1 <xml version="1.0" encoding="UTF-8">
2 <xslstylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3   <xsloutput method="html"/>
4   <!-- Template for "employees" elements -->
5   <xsltemplate match="Employees">
6     <h1>Employee Information</h1>
7     <table border="1">
8       <tr>
9         <th>Department</th>
10        <th>Name</th>
11        <th>Salary</th>
12        <th>Language</th>
13      </tr>
14      <xslfor-each select="Employee">
15        <tr>
16          <td>
17            <xslvalue-of select="Department"/>
18          </td>
19          <td>
20            <xslvalue-of select="Name"/>
21          </td>
22          <td>
23            <xslvalue-of select="Salary"/>
24          </td>
25          <td>
26            <xslvalue-of select="Language"/>
27          </td>
28        </tr>
29      </xslfor-each>
30    <!-- End of Table -->
31  </table>
32 </xsltemplate>
33 </xslstylesheet>
34

```

where,
xsl:for-each select="Employee"
 Iterates through the Employee node and applies a template.
xsl:value-of select="Department"
 Displays the value of Department element.
xsl:value-of select="Name"
 Displays the value of Name element.
xsl:value-of select="Salary"
 Displays the value of Salary element.
xsl:value-of select="Language"
 Displays the value of Language element.
</xsl:for-each>
 End of for-each loop.

XML Simplified/ Module 6/ 30 of 53

@ Aptech Limited



The xsl:for-each element 4-4

Output

Employee Information

Department	Name	Salary	Language
Marketing	John Thorp	\$1000	EN
Admin	David Blake	\$1200	GR
Physical Education	Ben Johns	\$800	TR
External Affairs	Susan Lopez	\$2800	EN

XML Simplified/ Module 6/ 31 of 53

@ Aptech Limited

Using slides 28 to 31, explain the **xsl:for-each** element.

The **xsl:for-each** element can be used to iterate through the XML elements of a specified node set. It applies a template repeatedly to each node in a set.

Explain the figure on slide 28 that shows the syntax for **xsl:for-each** element. Mention the expression is evaluated on the current context to determine the set of nodes to iterate over.

Explain the figure on slide 29 shows the code for style rules that are defined in **APTEmployees_stylesheet.xsl** file. Explain the figure on slide 30 that depicts the stylesheet for **APTEmployees_stylesheet.xsl** file.

Slides 32 to 35

Let us understand the `xsl:text` element.



The `xsl:text` element 1-4

- It is used to add literal text to the output.

Syntax

```
<xsl:text
    disable-output-escaping="yes"|"no">
</xsl:text>
```

where,
`disable-output-escaping`
 Turns on or off the ability to escape special characters.
`yes`
 If the value is yes, a `>` will appear as a `>`.
`no`
 If the value is no, a `>` will appear as a `>` in the text.

© Aptech Limited XML Simplified/ Module 6/ 32 of 53



The `xsl:text` element 2-4

Code Snippet

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="Orders_stylesheet.xsl"?>
3  <Orders>
4      <Item>
5          <Name>Dell Laptop</Name>
6          <Price>$45000</Price>
7          <ShippingDate>10-Mar-07</ShippingDate>
8      </Item>
9      <Item>
10         <Name>Mouse</Name>
11         <Price>$450</Price>
12         <ShippingDate>10-Mar-07</ShippingDate>
13     </Item>
14     <Item>
15         <Name>Dell Keyboard</Name>
16         <Price>$150</Price>
17         <ShippingDate>10-Mar-07</ShippingDate>
18     </Item>
19 </Orders>

```

© Aptech Limited XML Simplified/ Module 6/ 33 of 53

The xsl:text element 3-4

Style Sheet

```

1  <xsl:stylesheet version="1.0" encoding="UTF-8">
2  <xsl:output method="html"/>
3  <xsl:template match="/">
4  <html>
5  <body>
6  <xsl:text>
7  Following are the items which are shipped on 10th March
8  </xsl:text>
9  <br/>
10 <xsl:for-each select="Orders/Item">
11   <xsl:value-of select="Name"/>
12   <xsl:text>, </xsl:text>
13   </xsl:for-each>
14   <xsl:text>!!!!</xsl:text>
15   </body>
16 </html>
17 </xsl:template>
18 </xsl:stylesheet>
19
where,
  xsl:for-each select="Orders/Item"
  Iterates through the Item element.
  xsl:value-of select="Name"
  Displays the value of Name element.
  <xsl:text>,</xsl:text>
  Inserts a comma (,) after each name value.
  <xsl:text>!!!!</xsl:text>
  Inserts four exclamation marks (!) at the end of the output.

```

@ Aptech Limited

XML Simplified/ Module 6 / 34 of 53

The xsl:text element 4-4

Output

Following are the items which are shipped on 10th March
Dell Laptop,Mouse,Dell Keyboard,!!!!

@ Aptech Limited

XML Simplified/ Module 6 / 35 of 53

Using slides 32 to 35, explain the `xsl:text` element.

The `xsl:text` element is used to add literal text to the output. This element cannot contain any other XSL elements. It can contain only text.

Explain the figure on slide 32 that shows the syntax for `xsl:text` element.

Explain the figure on slide 33 that shows the code for style rules that are defined in `Orders_stylesheet.xsl` file.

Explain the figure on slide 34 depicts the stylesheet for `Orders_stylesheet.xsl` file.

Slides 36 to 39

Let us understand the `xsl:number` element.

The `xsl:number` element 1-4

- It can be used to determine the sequence number for the current node.

Syntax

```
<xsl:number
    count="pattern"
    format="{ string }"
    value="expression"
>
</xsl:number>
```

where,
count="pattern"
 Indicates what nodes are to be counted. Only nodes that match the pattern are counted.
format="{ string }"
 Sequence of tokens that specifies the format to be used for each number in the list.
value="expression"
 Specifies the expression to be converted to a number and output to the result tree.

@ Aptech Limited

XML Simplified/ Module 6/ 36 of 53

The `xsl:number` element 2-4

Code Snippet

```
1 | <?xml version="1.0" encoding="UTF-8"?>
2 | <?xmlstylesheet type="text/xsl" href="item_stylesheet.xsl" ?>
3 | <Items>
4 |   <Item>Water Bottle</Item>
5 |   <Item>Chocolates</Item>
6 |   <Item>Computer Book</Item>
7 |   <Item>Mobile Phone</Item>
8 |   <Item>Personal Computer</Item>
9 | </Items>
```

@ Aptech Limited

XML Simplified/ Module 6/ 37 of 53

The `xsl:number` element 3-4

Style Sheet

```
1 | <?xml version="1.0"?>
2 | <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3 |
4 |   <xsl:template match="Items">
5 |     <h3> Items numbered in Western and then upper-case Roman numbering</h3>
6 |     <xsl:for-each select="Item">
7 |       <xsl:number value="position()" format="1." />
8 |       <xsl:value-of select="." />
9 |
10 |       <xsl:number value="position()" format="I." />
11 |       <xsl:value-of select="." />
12 |     </xsl:for-each>
13 |   </xsl:template>
14 | </xsl:stylesheet>
```

where,
`position()`
 The current node's position in the source document.
`format="1."`
 User-provided number starts with 1.
`format="I."`
 User-provided roman number starts with I.

@ Aptech Limited

XML Simplified/ Module 6/ 38 of 53



The xsl:number element 4-4

Output

Items numbered in Western and then upper-case Roman numbering

1.Water Bottle , I.Water Bottle
2.Chocolates , II.Chocolates
3.Computer Book , III.Computer Book
4.Mobile Phone , IV.Mobile Phone
5.Personal Computer , V.Personal Computer

where,
1.Water Bottle, I.Water Bottle
The first item is numbered with number 1 and roman number I.
4.Mobile Phone, IV. Mobile Phone
The fourth item is numbered with number 4 and roman number IV.

@ Aptech Limited XML Simplified/ Module 6/ 39 of 53

Using slides 36 to 39, explain the `xsl:number` element.

The `xsl:number` element can be used to determine the sequence number for the current node. It can also be used to format a number for display in the output.

Explain the figure on slide 36 shows the syntax for `xsl:number` element.

Explain the figure on slide 37 shows the code for style rules that are defined in `item_stylesheet.xsl` file.

Explain the figure on slide 38 that depicts the stylesheet for `item_stylesheet.xsl` file. Explain the code snippet.

Slides 40 to 43

Let us understand the `xsl:if` element.



The `xsl:if` element 1-4

- Evaluates a conditional expression against the content of the XML file.

Syntax

```
<xsl:if
    test="expression"
>
</xsl:if>
```

where,
`test=expression`
The condition in the source data to test with either a true or false.

© Aptech Limited XML Simplified/ Module 6/ 40 of 53



The `xsl:if` element 2-4

Code Snippet

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <xslstylesheet type="text/xsl" href="Librarybooks_stylesheet.xsl"?>
3  <Catalog>
4      <Book>
5          <Title>XML By Example</Title>
6          <Author>David Blake</Author>
7          <Price>20.90</Price>
8          <Year>1990</Year>
9      </Book>
10     <Book>
11         <Title>XML Bible 1.1</Title>
12         <Author>David Trott</Author>
13         <Price>53</Price>
14         <Year>2004</Year>
15     </Book>
16     <Book>
17         <Title>XML Cookbook</Title>
18         <Author>Susan Jones</Author>
19         <Price>11.10</Price>
20         <Year>1995</Year>
21     </Book>
22     <Book>
23         <Title>XML Complete Reference </Title>
24         <Author>Andrew Neil</Author>
25         <Price>193</Price>
26         <Year>2001</Year>
27     </Book>
28 </Catalog>
```

© Aptech Limited XML Simplified/ Module 6/ 41 of 53

The xsl:if element 3-4

Style Sheet

```

1 <xsl:stylesheet version="1.0" encoding="UTF-8">
2   <xsl:template version="1.0"
3     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4     <xsl:template match="/">
5       <html>
6         <body>
7           <h2>Library Books</h2>
8           <table border="1">
9             <tbo border="1">
10               <th>Book</th>
11               <th>Author</th>
12               <th>Year</th>
13             </tbo>
14             <xsl:for-each select="Catalog/Book">
15               <xsl:if test="Price > 50">
16                 <tbo>
17                   <td><xsl:value-of select="Title"/></td>
18                   <td><xsl:value-of select="Author"/></td>
19                   <td><xsl:value-of select="Year"/></td>
20                 </tbo>
21               </xsl:if>
22             </xsl:for-each>
23           </table>
24         </body>
25       </html>
26     </xsl:template>
27   </xsl:stylesheet>

```

where,
 <xsl:for-each select="Catalog/Book">
 Iterates through the Booknode.
 <xsl:if test="Price > 50">
 Checks if the price of the book is greater than 50. If true, Author, Title and Year are displayed.

@ Aptech Limited

XML Simplified/ Module 6/ 42 of 53

The xsl:if element 4-4

Output

Library Books

Book Title	Author	Year
XML Bible 1.1	David Troff	2004
XML Complete Reference	Andrew Nel	2001

@ Aptech Limited

XML Simplified/ Module 6/ 43 of 53

Using slides 40 to 43, explain the `xsl:if` element.

The `xsl:if` element evaluates a conditional expression against the content of the XML file. The `test` attribute of `xsl:if` element contains a conditional expression that evaluates to a boolean value of **true** or **false**.

Explain the figure on slide 40 shows the syntax for `xsl:if` element.

Explain the figure on slide 41 shows the code for style rules are that defined in `Librarybooks_stylesheet.xsl` file.

Explain the figure on slide 42 depicts the stylesheet for `Librarybooks_stylesheet.xsl` file.

Slides 44 to 48

Let us understand the `xsl:choose` element.



The `xsl:choose` element 1-5

- It is used in conjunction with `xsl:when` and `xsl:otherwise` to express multiple conditional tests.

Syntax

```
<xsl:choose>
  <xsl:when test="expression">
    template body
  </xsl:when>
  ...
  <xsl:otherwise>
    template body
  </xsl:otherwise>
</xsl:choose>
```

where,

`xsl:when test="expression"`
The `xsl:when` element is examined in the order of occurrence. If the test expression is true, the code contained in that element is executed.
`xsl:otherwise`
If all the test conditions in any `xsl:when` element are false, then the `xsl:otherwise` element is automatically selected and the code associated with that element is executed.

@ Aptech Limited XML Simplified/ Module 6/ 44 of 53



The `xsl:choose` element 2-5

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="publisherbooks_stylesheet.xsl"?>
3  <Publisher>
4    <Book>
5      <Title>XML By Example</Title>
6      <Author>David Blake</Author>
7      <Price>20.90</Price>
8      <Year>1990</Year>
9    </Book>
10   <Book>
11     <Title>XML Cookbook</Title>
12     <Author>Susan Jones</Author>
13     <Price>11.10</Price>
14     <Year>1995</Year>
15   </Book>
16   <Book>
17     <Title>XML Complete Reference </Title>
18     <Author>Andrew Nel</Author>
19     <Price>19.95</Price>
20     <Year>2001</Year>
21   </Book>
22 </Publisher>
```

@ Aptech Limited XML Simplified/ Module 6/ 45 of 53

The xsl:choose element 3-5

Style Sheet

```

1 <xsl:stylesheet version="1.0" encoding="UTF-8">
2 <xsl:template match="/">
3   <html>
4     <body>
5       <h2>Publisher Books</h2>
6       <table border="1">
7         <tr bgcolor="pink">
8           <th>Title</th>
9           <th>Author</th>
10          <th>Price</th>
11          <th>Year</th>
12        </tr>
13        <xsl:for-each select="Publisher/Book">
14          <tr>
15            <td>
16              <xsl:value-of select="Title"/>
17            </td>
18            <xsl:choose>
19              <xsl:when test="Price > 100">
20                <td bgcolor="magenta">
21                  <xsl:value-of select="Author"/>
22                </td>
23                <td bgcolor="magenta">
24                  <xsl:value-of select="Price"/>
25                </td>
26                <td bgcolor="magenta">
27                  <xsl:value-of select="Year"/>
28                </td>
29              </xsl:when>
30            </td>
31          </tr>
32        </xsl:for-each>
33      </table>
34    </body>
35  </html>
36 </xsl:template>
37 </xsl:stylesheet>

```

@ Aptech Limited

XML Simplified/ Module 6/ 46 of 53

The xsl:choose element 4-5

Style Sheet

```

31   <xsl:otherwise>
32     <td>
33       <xsl:value-of select="Author"/>
34     </td>
35     <td>
36       <xsl:value-of select="Price"/>
37     </td>
38     <td>
39       <xsl:value-of select="Price"/>
40     </td>
41     <td>
42       <xsl:value-of select="Year"/>
43     </td>
44   </xsl:otherwise>
45 </xsl:choose>
46 </tr>
47 </xsl:for-each>
48 </table>
49 </body>
50 </html>
51 </xsl:template>
52 </xsl:stylesheet>

```

where,

`xsl:when test="Price > 100"`
 Checks whether the price of the book is greater than 100. If true, the details like Author, Price and Year are displayed with magenta as the background color.
`xsl:otherwise`
 If all the conditions are false, this block is executed. Here, all other book details are printed in normal background color.

@ Aptech Limited

XML Simplified/ Module 6/ 47 of 53

The xsl:choose element 5-5

Output

Publisher Books

Title	Author	Price	Year
XML By Example	David Blake	20.90	1990
XML Cookbook	Susan Jones	11.10	1995
XML Complete Reference	Andrew Net	19.9	2001

@ Aptech Limited

XML Simplified/ Module 6/ 48 of 53

Using slides 44 to 48, explain the `xsl:choose` element.

The `xsl:choose` element is used to make a decision when there are two or more possible courses of action. The `xsl:choose` element is used in conjunction with `xsl:when` and `xsl:otherwise` to express multiple conditional tests.

Explain the figure on slide 44 shows the syntax for `xsl:choose` element.

Explain the figure on slides 45 and 46 shows the code for style rules that are defined in `Publisherbooks_stylesheet.xsl` file.

Then, explain the figure on slide 47 depicts the stylesheet for `Publisherbooks_stylesheet.xsl` file.

Slides 49 to 52

Let us understand the sorting in XSLT.

Sorting in XSLT 1-4

- It can be used to sort a group of similar elements.

Syntax

```
<xsl:sort
  case-order="upper-first" | "lower-first"
  data-type="number" | "qname" | "text"
  order="ascending" | "descending"
  select="expression"
>
</xsl:sort>
```

where,

<code>case-order</code>	Indicates whether the sort will have uppercase or lowercase letters listed first in the sort output. The default option is to list uppercase first.
<code>data-type</code>	Specifies the data type of the strings.
<code>Number</code>	Sort key is converted to a number.
<code>Qname</code>	Sort is based upon a user-defined datatype.
<code>Text</code>	Specifies that the sort keys should be sorted alphabetically.
<code>Order</code>	The sort order for the strings. The default value is "ascending".
<code>Select</code>	Expression that defines the key upon which the sort will be based. The expression is evaluated and converted to a string that is used as the sort key.

@ Aptech Limited XML Simplified/ Module 6/ 49 of 53

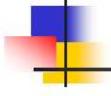
Sorting in XSLT 2-4

Code Snippet

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="Products_stylesheet.xsl"?>
3  <Products>
4    <Item>
5      <ItemCode>CD01</ItemCode>
6      <ItemName>Music CD</ItemName>
7      <UnitPrice>9.90</UnitPrice>
8    </Item>
9    <Item>
10      <ItemCode>PN01</ItemCode>
11      <ItemName>Parker Pens</ItemName>
12      <UnitPrice>12.50</UnitPrice>
13    </Item>
14    <Item>
15      <ItemCode>CK01</ItemCode>
16      <ItemName>Coca Cola</ItemName>
17      <UnitPrice>2.20</UnitPrice>
18    </Item>
19    <Item>
20      <ItemCode>BK01</ItemCode>
21      <ItemName>Computer Books</ItemName>
22      <UnitPrice>8.76</UnitPrice>
23    </Item>
24  </Products>
```

@ Aptech Limited XML Simplified/ Module 6/ 50 of 53



Sorting in XSLT 3-4

Style Sheet

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3   <xsl:template match="/">
4     <html>
5       <body>
6         <xsl:for-each select="Products/Item">
7           <xsl:sort data-type="number" select="UnitPrice" order="descending"/>
8           <xsl:value-of select="ItemName"/>
9           <xsl:text></xsl:text>
10          <xsl:value-of select="UnitPrice"/>
11          <br/>
12        </xsl:for-each>
13      </body>
14    </html>
15  </xsl:template>
16 </xsl:stylesheet>

```

where,
`select="UnitPrice"`
Sort is based on UnitPrice element.
`order="descending"`
The sorting order for UnitPrice is descending in that the higher value will be displayed first.

@ Aptech Limited XML Simplified/ Module 6/ 51 of 53



Sorting in XSLT 4-4

Output

```

Parker Pens-12.50
Music CD-9.90
Computer Books-8.76
Coca Cola-2.20

```

@ Aptech Limited XML Simplified/ Module 6/ 52 of 53

Using slides 49 to 52, explain the sorting in XSLT.

The `xsl:sort` element in XSLT can be used to sort a group of similar elements. The sorting can be done in various ways by using the attributes of this element.

Explain the figure on slide 49 that shows the syntax for `xsl:sort` element.

Explain the figure on slide 50 shows the code for style rules that are defined in `Products_stylesheet.xsl` file.

Then, explain the figure on slide 51 depicts the stylesheet for `Products_stylesheet.xsl` file.

The sorting order for `UnitPrice` is descending in that the higher value will be displayed first.

Slide 53

Let us summarize the session.



Summary

- **Introduction to XSL**
 - XML provides the ability to format document content.
 - XSL provides the ability to define how the formatted XML content is presented.
 - An XSL Transformation applies rules to a source tree read from an XML document to transform it into an output tree written out as an XML document.
- **Working with XSL**
 - An XSL template rule is represented as an `xsl:template` element.
 - You can process multiple elements in two ways: using the `xsl:apply-templates` element and the `xsl:for-each` element.
 - The `xsl:stylesheet` element allows you to include a stylesheet directly in the document it applies to.
 - The `xsl:if` element produces output only if its `test` attribute is `true`.

© Aptech Limited XML Simplified/ Module 6/ 53 of 53

Using slide 53, summarize the session. End the session, with a brief summary of what has been taught in the session.

6.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the more on XSLT that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 7 - More on XSLT

7.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

7.1.1 Objectives

By the end of this session, the learners will be able to:

- XPath
- XPath Expressions and Functions
- Working with different Styles

7.1.2 Teaching Skills

To teach this session, you should be well-verses with XPath and identifying the various nodes of XPath. You should also aware yourself with the different operators used with XPath and learn the various XPath expressions and functions. Finally, this session explains how to switch between styles and how to transform XML documents into HTML using XSLT.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Then give the students the overview of the current session in the form of Session Overview. Show the students slide 2 of the presentation.

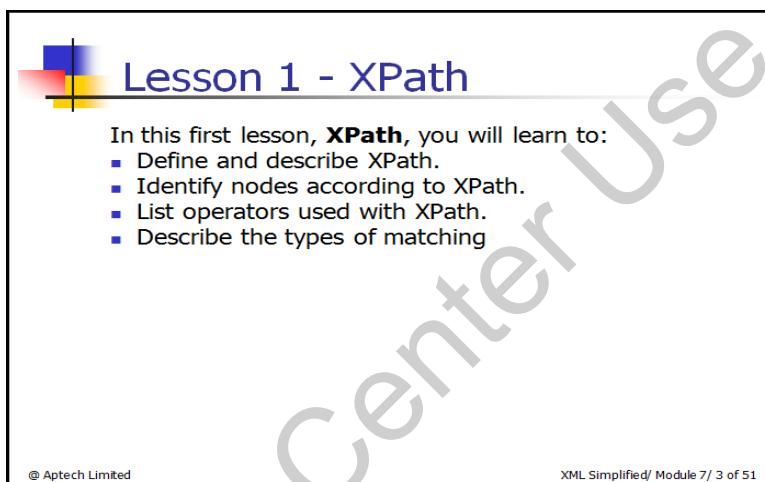
Tell the students that this session introduces XPath and identifying the various nodes of XPath. They will also learn the different operators used with XPath. They will understand and will be able to work with the various XPath expressions and functions.

They will also learn how to switch between styles and how to transform XML documents into HTML using XSLT.

7.2 In-Class Explanations

Slide 3

Let us understand the objectives of lesson 1.



The slide has a decorative graphic in the top-left corner consisting of overlapping colored squares (blue, red, yellow). The title "Lesson 1 - XPath" is centered at the top in a blue font. Below the title, the text "In this first lesson, **XPath**, you will learn to:" is followed by a bulleted list of five items. At the bottom left is the copyright notice "@ Aptech Limited" and at the bottom right is the page information "XML Simplified/ Module 7 / 3 of 51".

In this first lesson, **XPath**, you will learn to:

- Define and describe XPath.
- Identify nodes according to XPath.
- List operators used with XPath.
- Describe the types of matching

@ Aptech Limited XML Simplified/ Module 7 / 3 of 51

Using slide 3, explain the topics covered in the lesson 1.

Slides 4 and 5

Let us understand the XPath.



XPath 1-2



XPath 2-2

- XSLT
 - Language for transforming XML documents into XML, HTML or text.
- XQuery
 - Builds on XPath.
 - Language for extracting information from XML documents.

© Aptech Limited

XML Simplified/ Module 7 / 5 of 51

Using slides 4 and 5, explain the XPath.

XPath is used to navigate through elements and attributes in an XML document. It contains a library of standard functions that form a major element in XSLT. It is a major element in XSLT standards recommended by W3C.

XPath can be thought of as a query language like SQL. However, rather than extracting information from a database, it extracts information from an XML document. XPath is a language for retrieving information from a XML document. XPath is used to navigate through elements and attributes in an XML document. Thus, XPath allows identifying parts of an XML document.

XPath provides a common syntax as shown in figure on slide 4 for features shared by XSLT and XQuery.

- **XSLT** – It is a language for transforming XML documents into XML, HTML, or text.
- **XQuery** – It is built on XPath and is a language for extracting information from XML documents.

In-Class Question:

After you finish explaining the XPath, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which language can be used for querying XML data?

Answer:

XQuery.

Slide 6

Let us understand the benefits of XPath.

Benefits of XPath

- Provides single syntax that you can use for:
 - Queries
 - Addressing
 - Patterns
- Concise, simple, and powerful.
- Benefits:
 - Syntax is simple for the simple and common cases.
 - Any path that can occur in an XML document.
 - Any set of conditions for the nodes in the path can be specified.
 - Any node in an XML document can be uniquely identified.

© Aptech Limited XML Simplified/ Module 7/ 6 of 51

Using slide 6, explain the benefits of XPath.

XPath is designed for XML documents. It provides a single syntax that you can use for queries, addressing, and patterns. XPath is concise, simple, and powerful.

Explain the benefits of XPath in details.

XPath is designed to be used in many contexts. It is applicable for providing links to nodes, searching repositories, and for many other applications.

Tips:

Knowledge of XPath is important for:

- Business analysts
 - Project managers
 - Report developers
 - Programmers

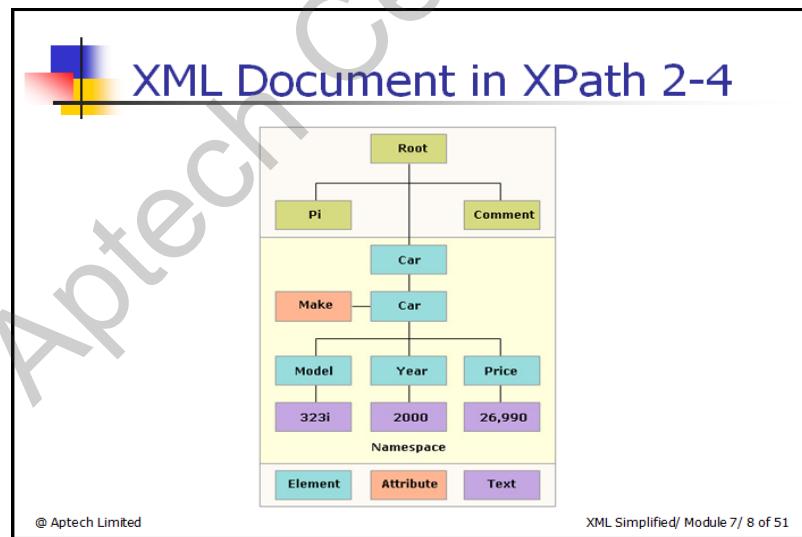
Slides 7 to 10

Let us understand the viewing of XML document in XPath.



XML Document in XPath 1-4

- An XML document is viewed as a tree.
- Each part of the document is represented as a node.
- Nodes
 - Root
 - A single `root` node that contains all other nodes in the tree.
 - Element
 - Every element in a document has a corresponding `element` node that appears in the tree under the `root` node.
 - Within an `element` node appear all of the other types of nodes that correspond to the element's content.
 - Element nodes may have a unique identifier associated with them that is used to reference the node with XPath.



XML Document in XPath 3-4

- Attribute
 - Each `element` node has an associated set of attribute nodes.
 - Element is the parent of each of these attribute nodes.
 - An attribute node is not a child of its parent element.
- Text
 - Character data is grouped into text nodes.
 - Characters inside comments, processing instructions and attribute values do not produce text nodes.
 - The `text` node has a parent node and it may be the `child` node.
- Comment
 - There is a `comment` node for every comment, except within the document type declaration (DTD).
 - The `comment` node has a parent node and it may be the child node too.

@ Aptech Limited

XML Simplified/ Module 7/ 9 of 51

XML Document in XPath 4-4

- Processing instruction
 - Processing instruction node exists for every processing instruction except for any processing instruction within the document type declaration.
 - The processing instruction node has a parent node and it may be the `child` node too.
- Namespace
 - Each element has an associated set of namespace nodes.
 - Provides descriptive information about their parent node.

@ Aptech Limited

XML Simplified/ Module 7/ 10 of 51

Using slides 7 to 10, explain the viewing of XML document in XPath.

In XPath, an XML document is viewed conceptually as a tree in which each part of the document is represented as a node as shown in figure on slide 8.

XPath have seven types of nodes. Explain each of them in detail using slides 7, 9, and 10.

Following table summarizes the various nodes in XPath tree:

Node Type	String Value	Expanded Name
root	Determined by concatenating the string-values of all text-node descendants in document order.	None.
element	Determined by concatenating the string-values of all text-node descendants in document order.	The element tag, including the namespace prefix (if applicable).
attribute	The normalized value of the attribute.	The name of the attribute, including the namespace prefix (if applicable).

Node Type	String Value	Expanded Name
text	The character data contained in the text node.	None.
comment	The content of the comment (not including <!-- and -->).	None.
processing instruction	The part of the processing instruction that follows the target and any whitespace.	The target of the processing instruction.
namespace	The URI of the namespace.	The namespace prefix.

Tips:

XML documents are treated as trees of nodes. The topmost element of the tree is called the root element.

Relationship in Nodes:

XML documents are treated as trees of nodes. The topmost element of the tree is called the root element.

Look at the following XML document:

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

<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

Examples of nodes in the XML document are:

```
<bookstore> (root element node)
<author>J K. Rowling</author> (element node)
lang="en" (attribute node)
```

Parent - Each element and attribute has one parent.

In the bookstore XML document, the book element is the parent of the title, author, year, and price.

Children - Element nodes may have zero, one, or more children.

In the bookstore XML document, the title, author, year, and price elements are all children of the book element.

Siblings - Nodes that have the same parent.

In the bookstore XML document, the title, author, year, and price elements are all siblings.

Ancestors - A node's parent, parent's parent, and so on.

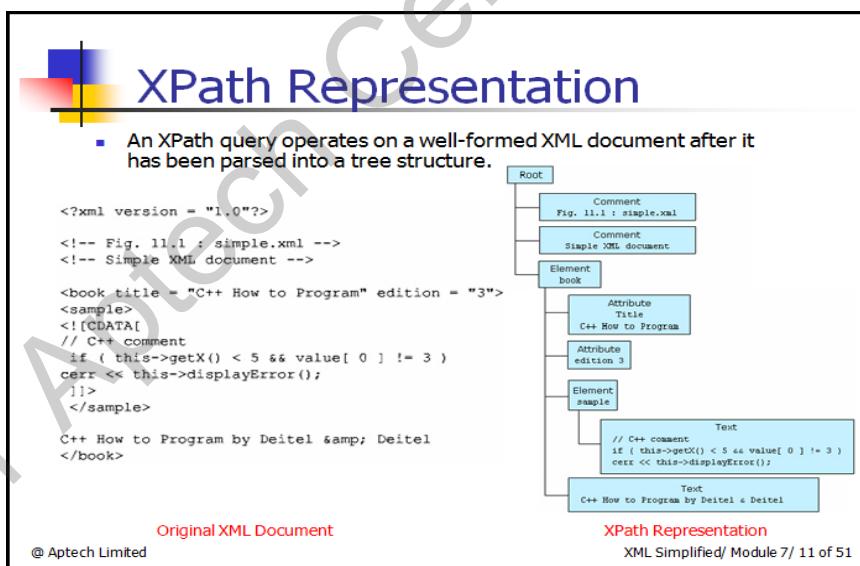
In the bookstore XML document, the ancestors of the title element are the book element and the bookstore element.

Descendants - A node's children, children's children, and so on.

In the bookstore XML document, descendants of the bookstore element are the book, title, author, year, and price elements.

Slide 11

Let us understand the XPath representation.



Using slide 11, explain the XPath representation.

An XPath query operates on a well-formed XML document after it has been parsed into a tree structure. Figure on slide 11 depicts a simple XML document along with its generated XPath tree.

Explain the figure showing the corresponding XPath tree that is generated after it has been parsed.

Slide 12

Let us understand the operators in XPath.



Operators in XPath

- An XPath expression returns a node set, a boolean, a string, or a number.
- XPath provides basic floating point arithmetic operators and some comparison and boolean operators.

Operator	Description
/	Child operator; selects immediate children of the left-side collection.
//	Recursive descent; searches for the specified element at any depth
.	Indicates the current context
..	The parent of the current context node
*	Wildcard; selects all elements regardless of the element name
@	Attribute; prefix for an attribute name
:	Namespace separator; separates the namespace prefix from the element or attribute name

@ Aptech Limited XML Simplified/ Module 7/ 12 of 51

Using slide 12, explain the operators in XPath.

An XPath expression returns a node set, a boolean, a string, or a number. XPath provides basic floating point arithmetic operators and some comparison and boolean operators.

The XPath expressions are constructed using the operators and special characters as shown in the table shown on slide 12.

Slide 13

Let us understand the XPath operators with examples.



Examples of XPath Operators

Expression	Refers to
Author/FirstName	All <FirstName> elements within an <Author> element of the current context node.
BookStore//Title	All <Title> elements one or more levels deep in the <BookStore> element.
BookStore/*>Title	All <Title> elements that are grandchildren of <BookStore> elements.
BookStore//Book/Excerpt//Work	All <Work> elements anywhere inside <Excerpt> children of <Book> elements, anywhere inside the <BookStore> element.
.//Title	All <Title> elements one or more levels deep in the current context.

@ Aptech Limited XML Simplified/ Module 7/ 13 of 51

Using slide 13, explain the XPath operators with examples.

Table on slide 13 shows few examples along with their description that use XPath operators in an expression.

Slides 14 to 16

Let us understand the types of matching.

Types of Matching 1-3

- XPath can create of the patterns.
- The match attribute of the `xsl:template` element supports a complex syntax.
- Allows to express exactly which nodes to be matched.
- The select attribute of `xsl:apply-templates`, `xsl:value-of`, `xsl:for-each`, `xsl:copy-of`, and `xsl:sort` supports a superset of the syntax.
- Allows to express exactly which nodes to selected and which nodes not to be selected.

© Aptech Limited XML Simplified/ Module 7/ 14 of 51

Types of Matching 2-3

- Matching by name
 - The source element is identified by its name, using the `match` attribute.
 - The value given to the `match` attribute is called the pattern.
- Code Snippet


```
<xsl:template match = "Greeting">
```

Matches all greeting elements in the source document.
- Matching by ancestry
 - As in CSS, a match can be made using the element's ancestry.
- Code Snippet


```
<xsl:template match = "P//EM">
```

Matches any 'EM' element that has 'P' as an ancestor.
- Matching the element names
 - The most basic pattern contains a single element name which matches all elements with that name.
- Code Snippet


```
<xsl:template match="Product">
<xsl:value-of select="Product_ID"/>
</xsl:template>
```

Matches Product elements and marks their Product_ID children bold.

© Aptech Limited XML Simplified/ Module 7/ 15 of 51

Types of Matching 3-3

- Matching the root
 - Enables all descendant nodes to inherit the properties on the root of document.
 - Uses a single forward slash to represent the root.
- Code Snippet


```
<xsl:template match = "/">
```

Selects the root pattern.
- Matching by attribute
 - As in CSS, a match can be made using the element's ancestry.
- Syntax


```
<xsl:template match = " element name['attribute'
(attribute-name)=attribute-value] ">
```

Uses square brackets to hold the attribute name and value.
- Code Snippet


```
<xsl:template match="Product">
<xsl:apply-templates select="@Units"/>
</xsl:template>
```

Applies the templates to the non-existent Units attributes of Product elements.

© Aptech Limited XML Simplified/ Module 7/ 16 of 51

Using slides 14 to 16, explain types of matching.

XPath is used in the creation of the patterns. The match attribute of the `xsl:template` element supports a complex syntax that allows to express exactly which nodes to be matched. The `select` attribute of `xsl:apply-templates`, `xsl:value-of`, `xsl:for-each`, `xsl:copy-of`, and `xsl:sort` supports an even more powerful superset of this syntax that allows to express exactly which nodes to select and which nodes not to be selected.

Explain the types of matching in detail using slides 15 and 16.

Slide 17

Let us understand the objectives of lesson 2.



Lesson 2 – XPath Expressions and Functions

In this second lesson, **XPath Expressions and Functions**, you will learn to:

- State and explain the various XPath expressions and functions.
- List the node set functions.
- List the boolean functions.
- State the numeric functions.
- Describe the string functions.
- Explain what result tree fragments are.

@ Aptech Limited XML Simplified/ Module 7/ 17 of 51

Using slide 17, explain the topics covered in the lesson 2.

Slides 18 and 19

Let us understand the XPath expressions.



XPath Expressions 1-2

- Statements that can extract useful information from the XPath tree.
- Instead of just finding nodes, one can count them, add up numeric values, compare strings, and more.
- Are like statements in a functional programming language.
- Evaluates to a single value.

@ Aptech Limited XML Simplified/ Module 7/ 18 of 51

XPath Expressions 2-2

- Node-set
 - An unordered group of nodes from the input document that match an expression's criteria.
- Boolean
 - XSLT allows any kind of data to be transformed into a boolean.
 - Often done implicitly when a string or a number or a node-set is used where a boolean is expected.
- Number
 - Numeric values useful for counting nodes and performing simple arithmetic.
 - Numbers like 43 or -7000 that look like integers are stored as doubles.
 - Non-number values, such as strings and booleans, are converted to numbers automatically as necessary.
- String
 - A sequence of zero or more Unicode characters.
 - Other data types can be converted to strings using the string() function.

© Aptech Limited

XML Simplified/ Module 7/ 19 of 51

Using slides 18 and 19, explain the XPath expression.

XPath expressions are statements that can extract useful information from the XPath tree. Instead of just finding nodes, one can count them, add up numeric values, compare strings, and more. They are much like statements in a functional programming language. Every XPath expression evaluates to a single value.

Explain the four types of expressions in XPath.

The XPath expression syntax includes literal forms for strings and numbers as well as operators and functions for manipulating all four XPath data types.

Slide 20

Let us understand the XPath functions.

XPath Functions

- XPath defines various functions required for XPath 2.0, XQuery 1.0 and XSLT 2.0.
- The different functions are Accessor, AnyURI, Node, Error and Trace, Sequence, Context, Boolean, Duration/Date/Time, String, QName and Numeric.
- Can be used to refine XPath queries and enhance the programming power and flexibility of XPath.
- Each function in the function library is specified using a function prototype that provides the return type, function name, and argument type.
- If an argument type is followed by a question mark, the argument is optional; otherwise, the argument is required.
- Function names are case-sensitive.
- The default prefix for the function namespace is fn.

© Aptech Limited

XML Simplified/ Module 7/ 20 of 51

Using slide 20, explain the XPath functions.

XPath defines various functions required for XPath 2.0, XQuery 1.0, and XSLT 2.0. The different functions are Accessor, AnyURI, Node, Error and Trace, Sequence, Context, Boolean, Duration/Date/ Time, String, QName, and Numeric.

XML Path Language (XPath) functions can be used to refine XPath queries and enhance the programming power and flexibility of XPath. Each function in the function library is specified using a function prototype that provides the return type, function name, and argument type. If an argument type is followed by a question mark, the argument is optional; otherwise, the argument is required. Function names are case-sensitive.

The default prefix for the function namespace is `fn`.

Slides 21 to 24

Let us understand the node-set functions.

Node-Set Functions 1-4

- Take a node-set argument.
- Return a node-set or information about a particular node within a node-set.
- Are `name()`, `local-name()`, `namespace-uri()` and `root()`.

Syntax

```

name()
fn:name()
fn:name(nodeset)

local-name()
fn:local-name()
fn:local-name(nodeset)

namespace-uri()
fn:namespace-uri()
fn:namespace-uri(nodeset)

root()
fn:root()
fn:root(node)

```

where,

`name()` : Returns name of current node or the first node in specified node set.

`local-name()` : Returns name of current node or the first node in specified node set – without namespace prefix.

`namespace-uri()` : Returns namespace URI of current node or first node in specified node set.

`root()` : Returns root of tree to which the current node or specified node belongs. This will usually be a document node.

© Aptech Limited

XML Simplified/ Module 7 / 21 of 51

Node-Set Functions 2-4

Code and Schema

```

<!-- Book.xsd -->
< ?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.BookCatalog.com"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <!-- definition for simple elements-->
  <xsd:element name="Author" type="xsd:string"/>
  <xsd:element name="Genre" type="xsd:string"/>
  <xsd:element name="Price" type="xsd:decimal"/>
  <xsd:element name="Title" type="xsd:string"/>
  <xsd:element name="Description" type="xsd:string"/>

  <!-- definition for complex elements-->
  <xsd:complexType name="Book">
    <xsd:sequence>
      <xsd:element ref="Author"/>
      <xsd:element ref="Genre"/>
      <xsd:element ref="Price"/>
      <xsd:element ref="Title"/>
      <xsd:element ref="Description"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="Catalog">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

© Aptech Limited

XML Simplified/ Module 7 / 22 of 51



Node-Set Functions 3-4

Stylesheet

```

<!--Sample.xsl -->
1 <xslstylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
2   <xsl:output method="html"/>
3   <xsl:template match="/">
4     <html>
5       <body>
6         <xsl:node-set>Functions</xsl:node-set>
7       </body>
8     </html>
9   </xsltemplate>
10  <xsl:template match="*=>">
11    <table width="100%" border="1">
12      <tr>
13        <td width="25%><xsl:value-of select="namespace-uri()"/></td>
14        <td width="25%><xsl:value-of select="name()"/></td>
15        <td width="25%><xsl:value-of select="local-name()"/></td>
16        <td width="25%><xsl:value-of select="text()"/></td>
17      </tr>
18    </table>
19  </xsltemplate>
20  <xsl:template match="*=>">
21    <td>
22      <xsl:value-of select="namespace-uri()"/>
23    </td>
24    <td>
25      <xsl:value-of select="name()"/>
26    </td>
27    <td>
28      <xsl:value-of select="local-name()"/>
29    </td>
30    <td>
31      <xsl:value-of select="text()"/>
32    </td>
33  </xsl:template>
34  <xsl:apply-templates select="*=>"/>
35 </xsltemplate>
36 <xsl:apply-templates select="*=>"/>
37 </xslstylesheet>

```

© Aptech Limited XML Simplified/ Module 7/ 23 of 51



Node-Set Functions 4-4

Formatted Output

Node-set Function

namespace-uri()	name()	local-name	text()
http://www.BookCatalog.com	Catalog	Catalog	
http://www.BookCatalog.com	Book	Book	
http://www.BookCatalog.com	Author	Author	Gambardella, Matthew
http://www.BookCatalog.com	Title	Title	XML Developer's Guide
http://www.BookCatalog.com	Genre	Genre	Computer
http://www.BookCatalog.com	Price	Price	44.95
http://www.BookCatalog.com	Publish	Publish	2000-10-01
http://www.BookCatalog.com	Description	Description	An in-depth look at creating applications with XML.

© Aptech Limited XML Simplified/ Module 7/ 24 of 51

Using slides 21 to 24, explain node-set functions.

Node-set functions take a node-set argument. These return a node-set, or information about a particular node within a node-set. The different node-set functions are `name()`, `local-name()`, `namespace-uri()`, and `root()`.

Figure on slide 21 shows the syntax for node-set functions. Explain the syntax. Figure on slide 22 shows the code and schema. Explain the code snippet.

Explain the code snippet on slide 23 that depicts the style sheet.

Slides 25 to 29

Let us understand the boolean functions.

Boolean Functions 1-5

- **boolean(arg)**
 - Returns a boolean value for a number, string, or node-set.

Syntax

```
fn:boolean(arg)
```

Code Snippet

```
<xsl>
<li><b>boolean(0)</b> = <xsl:value-of select="boolean(0)"/></li>
<li><b>boolean(1)</b> = <xsl:value-of select="boolean(1)"/></li>
<li><b>boolean(-100)</b> = <xsl:value-of select="boolean(-100)"/></li>
<li><b>boolean('hello')</b> = <xsl:value-of select="boolean('hello')"/></li>
<li><b>boolean('')</b> = <xsl:value-of select="boolean('')"/></li>
<li><b>boolean(/book)</b> = <xsl:value-of select="boolean(/book)"/></li>
</ul>
```

↓

```
boolean(0) = false
boolean(1) = true
boolean(-100) = true
boolean('hello') = true
boolean('') = false
boolean(/book) = false
```

© Aptech Limited XML Simplified/ Module 7/ 25 of 51

Boolean Functions 2-5

- **not(arg)**
 - The sense of an operation can be reversed by using the `not()` function.

Syntax

```
fn:not(arg)
```

© Aptech Limited XML Simplified/ Module 7/ 26 of 51

Boolean Functions 3-5

- **not(arg)**
 - The sense of an operation can be reversed by using the `not()` function.

Code Snippet

```
<xsl:template match="PRODUCT[not(position()=1)]">
<xsl:value-of select=".//>
</xsl:template>
```

↓

The template rule selects all product elements that are not the first child of their parents.

Code Snippet

```
<xsl:template match="PRODUCT[position() != 1]">
<xsl:value-of select=".//>
</xsl:template>
```

↓

The same template rule could be written using the not equal operator != instead.

© Aptech Limited XML Simplified/ Module 7/ 27 of 51

Boolean Functions 4-5

- `true()`
 - Returns the boolean value `true`.

Syntax

```
fn:true()
```

Code Snippet

```
<xsl:value-of select="true()"/>
```

© Aptech Limited XML Simplified/ Module 7/ 28 of 51

Boolean Functions 5-5

- `false()`
 - Returns the boolean value `true`.

Syntax

```
fn:false()
```

Code Snippet

```
<xsl:value-of select="false() or false()"/>
<xsl:value-of select="true() and false()"/>
<xsl:value-of select="false() and false()"/>
```

© Aptech Limited XML Simplified/ Module 7/ 29 of 51

Using slides 25 to 29, explain the boolean functions.

The XPath syntax supports boolean functions that return true or false, and can be used with comparison operators in filter patterns.

Explain the different boolean functions on the slides.

Slides 30 to 34

Let us understand the numeric functions.

Numeric Functions 1-5

- Return strings or numbers.
- Can be used with comparison operators in filter patterns such as:
 - number(arg)
 - ceiling(num)
 - floor(num)
 - round(num)

Syntax

```

number(arg)
fn:number(arg)

ceiling(num)
fn:ceiling(num)

floor(num)
fn:floor(num)

round(num)
fn:round(num)

```

where,

number(arg) : Returns numeric value of argument. Argument could be a boolean, a string, or a node-set.

ceiling(num) : Returns smallest integer greater than the number argument.

floor(num) : Returns largest integer that is not greater than the number argument.

round(num) : Rounds the number argument to the nearest integer.

@ Aptech Limited XML Simplified/ Module 7/ 30 of 51

Numeric Functions 2-5

Stylesheet(1-3)

```

<?xml version="1.0"?>
<xslstylesheet type="text/xsl" href="Number.xsl"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <body>
        <h3>Numeric Functions</h3>
        <ul>
          <li>
            <b><xsl:value-of select="number('1548')"/> = <xsl:value-of select="number('1548')"/></b></li>
          <li>
            <b><xsl:value-of select="number('-1548')"/> = <xsl:value-of select="number('-1548')"/></b></li>
          <li>
            <b><xsl:value-of select="number('text')"/> = <xsl:value-of select="number('text')"/></b></li>
          <li>
            <b><xsl:value-of select="number('226.38' div '1')"/> = <xsl:value-of select="number('226.38' div '1')"/></b></li>
          </ul>
        </body>
      </html>
    </xsl:template>
  </xsl:stylesheet>

```

@ Aptech Limited XML Simplified/ Module 7/ 31 of 51

Numeric Functions 3-5

Stylesheet(2-3)

```

<ul>
  <li>
    <b><xsl:value-of select="ceiling(2.5)"/> = <xsl:value-of select="ceiling(2.5)"/></b>
  </li>
  <li>
    <b><xsl:value-of select="ceiling(-2.3)"/> = <xsl:value-of select="ceiling(-2.3)"/></b>
  </li>
  <li>
    <b><xsl:value-of select="ceiling(4)"/> = <xsl:value-of select="ceiling(4)"/></b>
  </li>
</ul>
<ul>
  <li>
    <b><xsl:value-of select="floor(2.5)"/> = <xsl:value-of select="floor(2.5)"/></b>
  </li>
  <li>
    <b><xsl:value-of select="floor(-2.3)"/> = <xsl:value-of select="floor(-2.3)"/></b>
  </li>
  <li>
    <b><xsl:value-of select="floor(4)"/> = <xsl:value-of select="floor(4)"/></b>
  </li>
</ul>

```

@ Aptech Limited XML Simplified/ Module 7/ 32 of 51

Numeric Functions 4-5

Stylesheet(3-3)

```
<ul>
  <li>
    <b>round(3.6)</b> = <xsl:value-of select="round(3.6)" />
  </li>
  <li>
    <b>round(3.4)</b> = <xsl:value-of select="round(3.4)" />
  </li>
  <li>
    <b>round(3.5)</b> = <xsl:value-of select="round(3.5)" />
  </li>
  <li>
    <b>round(-0.6)</b> = <xsl:value-of select="round(-0.6)" />
  </li>
  <li>
    <b>round(-2.5)</b> = <xsl:value-of select="round(-2.5)" />
  </li>
</ul>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

@ Aptech Limited XML Simplified/ Module 7/ 33 of 51

Numeric Functions 5-5

Output

Numeric Functions

- `number('1548')` = 1548
- `number('-1548')` = -1548
- `number('text')` = NaN
- `number('226.38' div '1')` = 226.38
- `ceiling(2.5)` = 3
- `ceiling(-2.3)` = -2
- `ceiling(4)` = 4
- `floor(2.5)` = 2
- `floor(-2.3)` = -3
- `floor(4)` = 4
- `round(3.6)` = 4
- `round(3.4)` = 3
- `round(3.5)` = 4
- `round(-0.6)` = -1

@ Aptech Limited XML Simplified/ Module 7/ 34 of 51

Using slides 30 to 34, explain the numeric functions.

XPath syntax supports number functions that return strings or numbers and can be used with comparison operators in filter patterns. The different numeric functions are `number(arg)`, `ceiling(num)`, `floor(num)`, and `round(num)`.

Explain the figure on slide 30 shows the syntax for numeric functions.

Explain the code snippet on slides 31 to 33 depicts the style sheet for numeric functions. Explain the code snippet.

Slides 35 to 39

Let us understand the string functions.

String Functions 1-5

- String functions are used to:
 - Evaluate
 - Format and manipulate string arguments
 - Convert an object to a string
- Different String functions are:
 - `string(arg)`
 - `compare()`
 - `concat()`
 - `substring()`

© Aptech Limited XML Simplified/ Module 7/ 35 of 51

String Functions 2-5

<p>Syntax</p> <pre>string(arg) fn:string(arg) translate() fn:translate(string, string, string) concat() fn:concat(string, string, ...) substring() fn:substring(string, start, len) fn:substring(string, start)</pre>	<div style="border: 1px solid black; padding: 5px; background-color: #ffffcc;"> <p>where,</p> <p><code>string(arg)</code>: Returns string value of the argument. The argument could be a number, boolean, or node-set.</p> <p><code>translate()</code>: Returns first argument string with occurrences of characters in second argument string replaced by character at the corresponding position in third argument string.</p> <p><code>concat()</code>: Returns concatenation of the strings.</p> <p><code>substring()</code>: Returns substring from the start position to specified length.</p> </div>
---	---

© Aptech Limited XML Simplified/ Module 7/ 36 of 51

String Functions 3-5

<p>Code Snippet</p> <pre><?xml version="1.0"?> <?xml-stylesheet type="text/xsl" href="BookDetail.xsl"?> <BookStore> <Book> <Title>The Weather Pattern</Title> <Author>Weather Man</Author> <Price>100.00</Price> </Book> <Book> <Title>Weaving Patterns</Title> <Author>Weaver</Author> <Price>150.00</Price> </Book> <Book> <Title>Speech Pattern</Title> <Author>Speaker</Author> <Price>15.00</Price> </Book> <Book> <Title>Writing Style</Title> <Author>Writer</Author> <Price>1500.00</Price> </Book> </BookStore></pre>	
---	--

© Aptech Limited XML Simplified/ Module 7/ 37 of 51

String Functions 4-5

Stylesheet

```

1 <?xml version="1.0"?>
2 <xsl:stylesheet type="text/xsl" href="BookDetail.xsl">
3 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4 <xsl:template match="">
5   <html>
6     <head>
7       <title>example</title>
8     </head>
9     <body>
10      <xsl:value-of select='translate("-----Books-----","Bok","ook")'>
11      <br/>
12      <xsl:value-of select="concat('AWARD WINNING', 'BOOK DETAILS')"/>
13      <xsl:apply-templates select="//Book"/>
14    </body>
15  </html>
16 </xsl:template>
17 <xsl:template match="Book">
18   <xsl:if test="contains>Title, 'Pattern')">
19     <DIV>
20       <D>
21         <xsl:value-of select="Title"/>
22       </D>
23     <B>
24     <D>
25       <xsl:value-of select="Author"/>
26     </D>
27   costs
28   <xsl:value-of select="Price"/>
29 </DIV>
30 </xsl:if>
31 </DIV>
32 </xsl:template>
33 </xsl:stylesheet>
34

```

@ Aptech Limited

XML Simplified/ Module 7/ 38 of 51

String Functions 5-5

Output

-----Books-----
 AWARD WINNINGBOOK DETAILS
The Weather Pattern by *Weather Man* costs 100.00 .
Weaving Patterns by *Weaver* costs 150.00 .
Speech Pattern by *Speaker* costs 15.00 .

@ Aptech Limited

XML Simplified/ Module 7/ 39 of 51

Using slides 35 to 39, explain the string functions.

String functions are used to evaluate, format, and manipulate string arguments, or to convert an object to a string.

Explain the different string functions using slide 35.

Explain the figure on slide 36 shows the syntax for string functions. Then, explain the code snippet provided on slides 37 and 38.

Slide 40

Let us understand the result tree fragments.

Result Tree Fragments

- A portion of an XML document that is not a complete node or set of nodes.
- The only allowed operation in a result tree fragment is on a string.
- The operation on the string may involve first converting the string to a number or a boolean.
- Result tree fragment is an additional data type other than four basic XPath data types, such as, string, number, boolean, node-set.
- A result tree fragment represents a fragment of the result tree.
- It is not permitted to use the /, //, and [] XPath operators on Result tree fragments.

© Aptech Limited

XML Simplified/ Module 7/ 40 of 51

Using slide 40, explain the result tree fragments.

A result tree fragment is a portion of an XML document that is not a complete node or set of nodes.

The only allowed operation in a result tree fragment is on a string. The operation on the string may involve first converting the string to a number or a boolean. Result tree fragment is an additional data type other than four basic XPath data types (such as string, number, boolean, node-set).

A result tree fragment represents a fragment of the result tree. In particular, it is not permitted to use the /, //, and [] XPath operators on result tree fragments.

Slide 41

Let us understand the objectives of lesson 3.

Lesson 3 – Working with different Styles

In this last lesson, **Working with different Styles**, you will learn to:

- Explain how to switch between styles.
- Describe how to transform XML documents into HTML using XSLT.

© Aptech Limited

XML Simplified/ Module 7/ 41 of 51

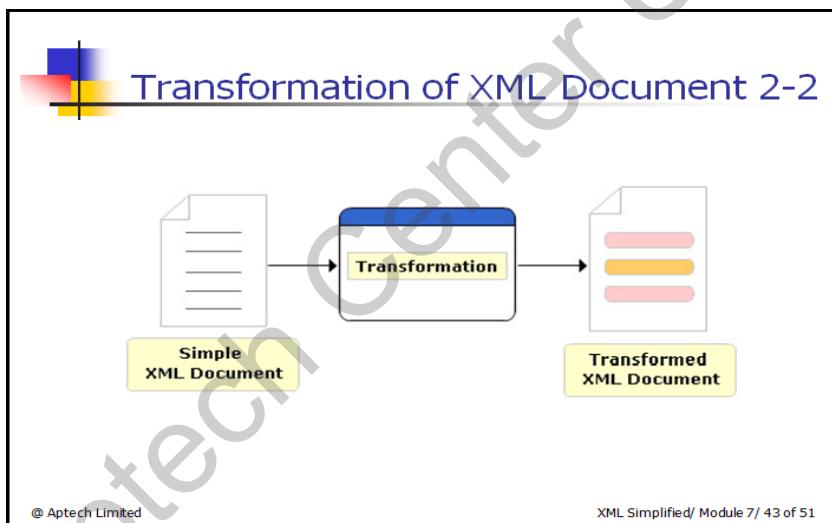
Using slide 41, explain the topics covered in lesson 3.

Slides 42 and 43

Let us understand how to transform a XML document.



Transformation of XML Document 1-2



Using slides 42 and 43, explain how to transform a XML document.

Transformation is one of the most important and useful techniques for working with XML. XML can be transformed by changing its structure, its markup, and perhaps its content into another form. The most common reason to transform XML is to extend the reach of a document into new areas by converting it into a presentational format.

Explain the figure on slide 43 demonstrating the transformation of XML document.

Slides 44 and 45

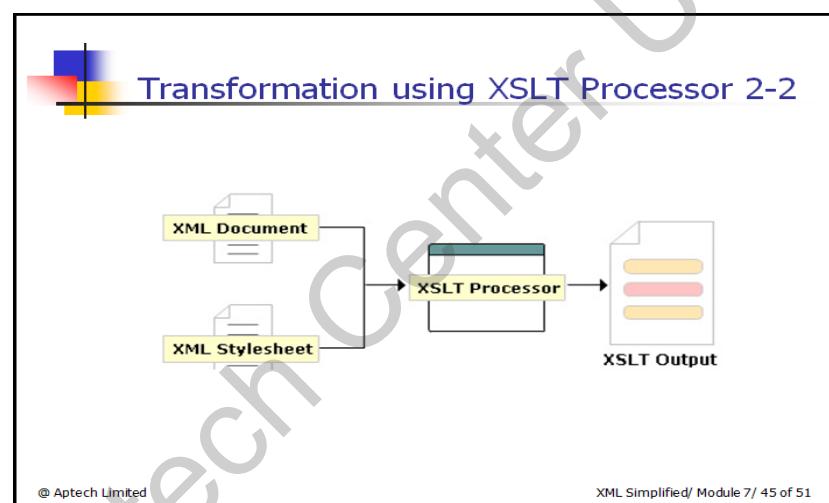
Let us understand transforming using XSLT processor.

Transformation using XSLT Processor 1-2

- Takes two inputs:
 - An XSLT stylesheet to govern the transformation process.
 - An input document called the source tree.
- The output is called the result tree.
- The XSLT engine begins by reading in the XSLT stylesheet and caching it as a look-up table.
- XPath locates the parts of XML document such as Element nodes, Attribute nodes and Text nodes.
- For each node the XSLT processes, it will look in the table for the best matching rule to apply.
- Starting from the root node, the XSLT engine finds rules, executes them, and continues until there are no more nodes in its context node set to work with.
- At that point, processing is complete and the XSLT engine outputs the result document.

@ Aptech Limited

XML Simplified/ Module 7/ 44 of 51



@ Aptech Limited

XML Simplified/ Module 7/ 45 of 51

Using slides 44 and 45, explain transformation using XSLT processor.

An XSLT processor takes two things as input: an XSLT style sheet to govern the transformation process and an input document called the source tree. The output is called the result tree.

The XSLT engine begins by reading in the XSLT style sheet and caching it as a look-up table. XPath locates the parts of XML document such as Element nodes, Attribute nodes, and Text nodes. Thus, for each node the XSLT processes, it will look in the table for the best matching rule to apply. Starting from the root node, the XSLT engine finds rules, executes them, and continues until there are no more nodes in its context node set to work with. At that point, processing is complete and the XSLT engine outputs the result document.

Explain the figure on slide 45 depicting the XSLT transformation.

Slides 46 and 47

Let us understand transforming XML using XSLT.

Transforming XML using XSLT 1-2

- Step 1
 - Creates a normal XML document.

Code Snippet

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

- Step 2
 - Add the following lines.

Code Snippet

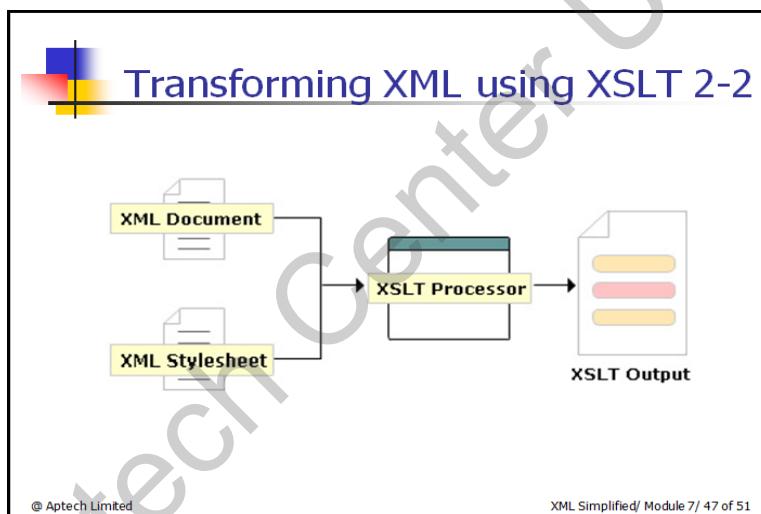
```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0" >
  ...
</xsl:stylesheet>
```

- Step 3
 - Set it up to produce HTML-compatible output.
 - The `<xsl:output>` tag should be used with either "text" or "html" to output anything besides well-formed XML.
 - The default value is "xml".

Code Snippet

```
<xsl:stylesheet>
  <xsl:output method="html"/>
  ...
</xsl:stylesheet>
```

© Aptech Limited XML Simplified/ Module 7/ 46 of 51



Using slides 46 and 47, explain transforming XML using XSLT. The transformation of XML document can be done in various steps that are:

Step 1: Create a normal XML document.

Step 2: Create an XSL style sheet.

Step 3: Set it up to produce HTML-compatible output.

Slides 48 and 49

Let us understand transforming XML using XSLT example.

Transforming XML using XSLT Example 1-2

- Example code of transforming XML documents into HTML using XSLT processor

Code Snippet

```

1  <?xml version="1.0"?>
2  <?xml-stylesheet type="text/xsl" href="ProductInfo.xsl"?>
3
4  <Company>
5
6    <Product>
7      <Product_ID>S002</Product_ID>
8      <Name>Steel</Name>
9      <Price>1000/tonne</Price>
10     <Boiling_Point Units="Kelvin">20.28</Boiling_Point>
11     <Melting_Point Units="Kelvin">13.81</Melting_Point>
12   </Product>
13
14   <Product>
15     <Product_ID>S004</Product_ID>
16     <Name>Iron</Name>
17     <Price>5000/tonne</Price>
18     <Boiling_Point Units="Kelvin">34.216</Boiling_Point>
19     <Melting_Point Units="Kelvin">23.81</Melting_Point>
20   </Product>
21
22 </Company>

```

where,
Company: The root Company element contains Product child elements.
Units: Attribute specifies the units for products melting point and boiling point.

© Aptech Limited XML Simplified/ Module 7/ 48 of 51

Transforming XML using XSLT Example 2-2

Stylesheet

```

1  <?xml version="1.0"?>
2
3  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4
5    <xsl:template match="Company">
6      <html>
7        <xsl:apply-templates/>
8      </html>
9    </xsl:template>
10
11   <xsl:template match="Product">
12     <P>
13       <xsl:apply-templates/>
14     </P>
15   </xsl:template>
16
17 </xsl:stylesheet>

```

Output

```

S002Steel1000/tonne20.2813.81
S004Iron5000/tonne34.21623.81

```

© Aptech Limited XML Simplified/ Module 7/ 49 of 51

Using slides 48 and 49, explain transforming XML using XSLT example.

An example code of transforming XML documents into HTML using XSLT processor has been provided to explain the process.

Then, explain the code on slides 48 and 49 that depicts the style sheet for ProductionInfo.xsl file.

Slides 50 and 51

Let us summarize the session.



Summary 1-2

- **XPath**
 - Notation for retrieving information from a document.
 - Provides a common syntax for features shared by Extensible Stylesheet Language Transformations (XSLT) and XQuery.
 - Have seven types of node as Root, Element, Attribute, Text, Comment, Processing instruction and Namespace.
 - Used in the creation of the patterns.
- **XPath Expressions and Functions**
 - The four types of expressions in XPath are
 - Node-sets.
 - Booleans.
 - Numbers.
 - Strings.

@ Aptech Limited XML Simplified/ Module 7/ 50 of 51



Summary 2-2

- **XPath Expressions and Functions (contd...)**
 - Functions defined for XPath are Accessor, AnyURI, Node, Error and Trace, Sequence, Context, Boolean, Duration/Date/Time, String, QName and Numeric.
 - A Result tree fragment is a portion of an XML document that is not a complete node or set of nodes.
- **Working with different styles**
 - Transformation is one of the most important and useful techniques for working with XML.
 - To transform XML is to change its structure, its markup, and perhaps its content into another form.
 - Transformation can be carried out using an XSLT processor also.

@ Aptech Limited XML Simplified/ Module 7/ 51 of 51

Using slides 50 and 51, summarize the session. End the session, with a brief summary of what has been taught in the session.

7.3 Post Class Activities for Faculty

You can brief the objectives of what has been taught in the course. You can solve the queries related to other sessions taught in the course.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 8: Introduction to JSON

8.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. Prepare a question or two that will be a key point to relate the current session objectives.

8.1.1 Objectives

By the end of this session, learners will be able to:

- Define and describe JavaScript Object Notation (JSON)
- Explain the uses of JSON
- List and describe the features of JSON
- Compare JSON with XML
- Describe the advantages of JSON
- Explain syntax and rules
- Explain literal notation in JavaScript and conversion to JSON
- Explain how to create a simple JSON file

8.1.2 Teaching Skills

To teach this session, you should be well versed with JSON along with its history, features, uses, and benefits. You should be familiar with the differences between JSON and XML, JSON syntax and rules, and JavaScript literal notation. You should also be aware of how JSON files are created.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students an overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

8.2 In-Class Explanations

Slide 3

Let us understand the need of JSON.

Introduction: The Need of JSON

- o XML parses using Document Object Model (DOM) with complex processes and no cross-browser compatibility.
- o JavaScript engine incorporates a markup language into an HTML Web page.
- o The need of more natural format native to this engine and overcome DOM limitations was felt.

```

<?xml>
{JSON}
  
```

© Aptech Ltd. Modern Markup for Data Interchange/Session 8

Using slide 3, explain the need of JSON to students. Tell them that XML parses text by using Document Object Model (DOM). This model calls several methods and that DOM works differently in each browser. As a result, parsing becomes complex and time consuming. Moreover, the risk of cross-browser compatibility failure also exists. Due to these facts, it became essential to have a new component to incorporate a markup language into an HTML Web page easily in all mainstream browsers. This component was the JavaScript engine that required a more discerning and natural format. Thus, JSON was introduced.

Slide 4

Let us understand JSON.

What is JSON

- A lenient text format derived from JavaScript language and ECMAScript.
- Platform-as well as language-independent.
- Implements conventions of the C-based languages.
- More concise than XML.

JSON
JAVASCRIPT OBJECT NOTATION

JSON is a lightweight, text-based, and open standard data-interchange format.

© Aptech Ltd. Modern Markup for Data Interchange/Session 8

Using slide 4, briefly explain JSON to students. Focus on the definition. Tell them that JSON implements conventions that are familiar to the programmers of C languages, such as C#, Java, Perl, and JavaScript.

Slide 5

Let us explore the uses of JSON.

Uses of JSON

- For serializing data to be sent over a network
- For facilitating communication between incompatible platforms
- For developing JavaScript based applications
- For sending public data through Web services and Application Programming Interfaces (APIs)
- For fetching data without hard coding in HTML

© Aptech Ltd. Modern Markup for Data Interchange/Session 8

Using slide 5, briefly explain the uses of JSON. Tell them that serialization converts data structures to a format that is suitable for transferring over a network or for storing in a memory or file. Give them an example of incompatible platforms. For instance, JSON allows communicating between a Java application on UNIX and a Visual Basic application on Windows. Give them a real life scenario for the last use, such as the Web site eBay.com wherein a template defines how the site appears to visitors, rather than hard coding each product. The desired data is then retrieved from the JSON file on server.

Additional Information:

For more information, visit the following links:

<http://tutorials.pluralsight.com/software-engineering-best-practices/everything-you-need-to-know-about-json?branch=vCillusion>
<http://www.copterlabs.com/json-what-it-is-how-it-works-how-to-use-it/>

In-Class Question:

After you finish explaining the uses of JSON, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

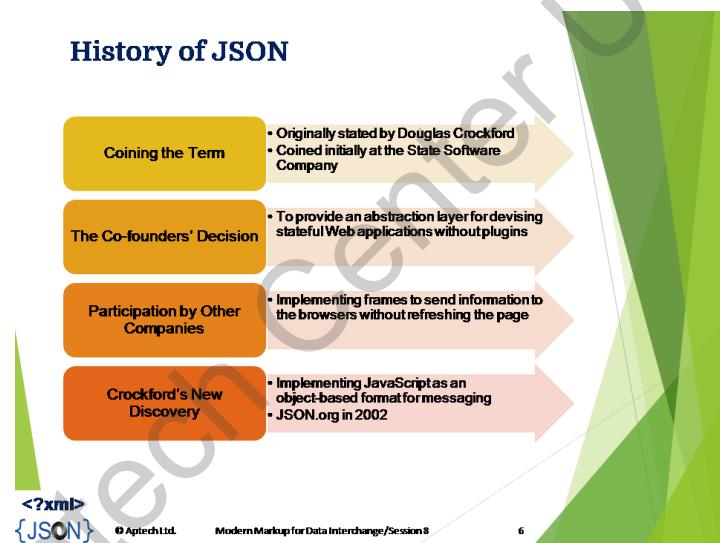


Why JSON is used to facilitate communication between incompatible platforms?

Answer: JSON as a data-interchange format is used to facilitate communication between incompatible platforms because it works on almost any platform and with most programming languages.

Slide 6

Let us see the history of JSON.



Using slide 6, explain to students how JSON evolved. Tell that the co-founders wanted a system that can take the advantage of the browser potential to give a layer of abstraction for designing stateful Web applications. These applications need to retain a lasting connection with a server by having two Hypertext Transfer Protocol (HTTP) connections open. For such a system, Crockford found that JavaScript can be implemented as an object-based format for messaging, which paved the way for JSON.

Slides 7 and 8

Let us now understand the features of JSON.

Features of JSON 1-2

- Standard structure
- Simplicity
- Open-source
- Self-describing
- Lightweight
- Scalable/Reusable

© Aptech Ltd. Modern Markup for Data Interchange/Session 8

Features of JSON 2-2

- Clean data
- Efficiency
- Interoperability
- Extensibility
- Internationalization

© Aptech Ltd. Modern Markup for Data Interchange/Session 8

Using slides 7 and 8, explain each feature of JSON to students.

- **Standard Structure:** Makes it easier to code in JSON.
- **Simplicity:** Makes it easier to understand the code as well as parse.
- **Open:** Is publicly available for adding other elements when required.
- **Self-describing:** Allows describing one's own field names and their values.
- **Lightweight:** Takes less storage space than XML format as well as is easier to obtain and load such data without reloading the page.
- **Scalable/Reusable:** Makes it easier to change the programming language on the server due to same data structure across all such languages.
- **Clean Data:** Makes JSON less susceptible to errors due to standard format as well as enables offering robust cross-browser solutions.
- **Efficiency:** Facilitates transferring only the changed details to client on every request, as the previous output is cloned on client side.

- **Interoperability:** Allows two or more applications coded in a different programming language to communicate with each other smoothly.
- **Extensibility:** Involves storing only standard data such as numbers and text, unlike XML storing any data type. While this is a pro of XML, it is not required for transferring JSON data, which simply denotes simplicity.
- **Internationalization:** Implements the Unicode norm due to which generic tools can easily manipulate JSON structures.

Additional Information:

For more information, visit the following links:

<http://www.json.org/xml.html>

<http://www.revillweb.com/articles/why-use-json/>

In-Class Question:

After you finish explaining the features of JSON, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Why JSON is an open-source format?

Answer: JSON is an open-source format as a developer can easily add other elements when needed to customize it as per specific requirements.

Slides 9 to 11

Let us distinguish between JSON and XML.

Characteristic	JSON	XML
Data Types	Offers scalar data types.	Does not offer any idea of data types.
Array Support	Provides native support.	Expresses arrays by conventions.
Object Support	Provides native support.	Expresses objects by conventions.
Null Support	Recognizes the value natively.	Mandates the use of xsi:nil on elements.
Comments	Does not support.	Provides native support (via APIs).

<?xml>
 {JSON}

© Aptech Ltd. Modern Markup for Data Interchange / Session 8

JSON versus XML 2-3

| Characteristic | JSON | XML |
|-----------------------|--|--|
| Namespaces | Does not support namespaces. | Accepts namespaces to prevent name collisions. |
| Formatting | Is simple and offers more direct data mapping. | Is complex and needs more effort for mapping application types to elements. |
| Size | Has very short syntax. | Has lengthy documents. |
| Parsing in JavaScript | Needs no additional application for parsing. | Implements XML DOM and requires extra code for mapping text to JavaScript objects. |
| Parsing | Has JSONPath. | Has XPath specification. |
| Learning Curve | Is not steep at all. | Is steep. |

<?xml>
 {JSON} © Aptech Ltd. Modern Markup for Data Interchange/Session 8

10

JSON versus XML 3-3

| Characteristic | JSON | XML |
|-------------------|---|---|
| Complexity | Is complex. | Is more complex. |
| Schema (Metadata) | Has a schema but is not widely used. | Uses many specifications for defining a schema. |
| Styling | Has no special specification. | Has XSLT specification for styling an XML document. |
| Querying | Has specifications such as JSON Query Language (JQQL) and JSONiq but are not widely used. | Has XQuery specification that is widely used. |
| Security | Is less secure, as the browser has no JSON parser. | Is more secure. |

<?xml>
 {JSON} © Aptech Ltd. Modern Markup for Data Interchange/Session 8

11

Using slides 9 to 11, explain to students the differences between JSON and XML in terms of different characteristics.

Additional Information:

For more information, visit the following links:

<http://codebucket.co.in/which-one-is-better-xml-or-json/>

http://www.cs.tufts.edu/comp/150IDS/final_papers/tstras01.1/FinalReport/FinalReport.html

Slide 12

Let us view the difference between JSON code and XML code.

JSON versus XML Coding

| JSON | XML |
|---|---|
| <pre>{ "students": [{ "name": "Steve", "age": "20", "city": "Denver" }, { "name": "Bob", "age": "21", "city": "Sacramento" }] }</pre> | <pre><students> <student> <name>Steve</name> <age>20</age> <city>Denver</city> </student> <student> <name>Bob</name> <age>21</age> <city>Sacramento</city> </student> </students></pre> |

12

Using slide 12, focus on both the snippets and show students how JSON is lighter and more compact than XML.

Slide 13

Let us explore the advantages of JSON.

Advantages of JSON

- Quick serialization
- Efficient encoding
- Faster parsing than XML and Yaml Ain't Markup Language (YAML)
- Simpler to work with other languages
- Easy differentiation between data types

13

Using slide 13, explain the advantages of using JSON. Tell them that JSON is used for serializing and de-serializing data without involving lots of padding that is prevalent in XML. Further, the JSON structure is quite compact. As a result, the size of JSON documents is smaller than XML documents.

The encoding mechanism is efficient enough to add minimum characters needed to represent the structure and values. Even the parser for YAML that serializes complex data sets without a DTD consumes more time than JSON to give an output of larger streams of serialized data. Further, JSON differentiates between the string '2' and the number '2'.

Additional Information:

For more information, visit the following link:

<http://www.json2yaml.com/yaml-vs-json>

Slide 14

Let us now understand the limitations of JSON.

Limitations of JSON

- o Difficult in reading the format and precision of keeping brackets in its place
- o Following snippet illustrates complexity:

```
<?xml>
{<JSON>
  "problems": [
    "Diabetes": [
      "medications": [
        "medicationsClasses": [
          "className": [
            "associatedDrug": [
              "name": "Asprin", "dose": "", "strength": "500
mg"
            ]
          ]
        ]
      ]
    ]
  ]
}
```

Aptech Ltd.

Modern Markup for Data Interchange / Session 8

14

Using slide 14, explain limitations of JSON to students. Focus on the code and say that the most significant limitation is difficulty in reading the format when there are so many brackets and commas involved.

Slide 15

Let us explore the types of data structures supported by JSON.

JSON Data Structures

- Support for two universally-supported data structures:
 - Ordered list of values, which can be an array, vector, list, or sequence.
 - Unordered set of name/value pairs, which can be a keyed list, object, record, struct, or a hash table.

15

Using slide 15, explain the two data structures supported by JSON to students. Tell that both of them are supported by almost all programming languages.

Slide 16

Let us now understand JSON arrays and objects.

JSON Arrays and Objects

- Following figure represents an array (ordered set) in JSON:

```

graph LR
    A["[ ]"] --- B["value"]
    B --- C[","]
    C --- D["[ ]"]
  
```

- Following figure represents an object (unordered set) in JSON:

```

graph LR
    A["[ ]"] --- B["string"]
    B --- C[":"]
    C --- D["value"]
    D --- E["[ ]"]
  
```

16

Using slide 16, explain to students that an ordered set is also called an array, while an unordered set is called an object. Focus on the diagrams to tell them how they are written.

In-Class Question:

After you finish explaining JSON arrays and objects, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Give an example of an array and object data structures.

Answer: A sequenced set holding colors of rainbow is an array, while a set holding details of a book such as name, ISBN no, and price is an object.

Slide 17

Let us see how to define a JSON object.

JSON Objects

- Following snippet represents a JSON object:

```
{  
  "Name": "Janet George",  
  "Street": "123 Ashley Street",  
  "City": "Chicago",  
  "Code": "60604"  
}
```

<?xml>
{JSON}

© Aptech Ltd.

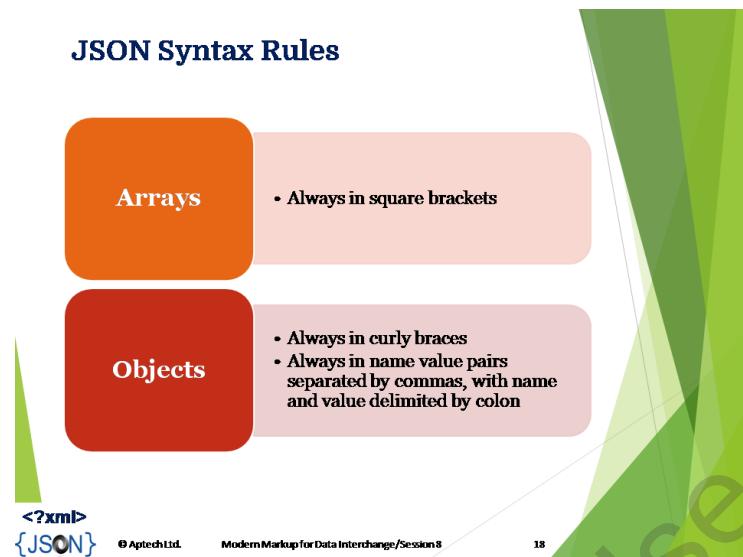
Modern Markup for Data Interchange/Session 8

17

Using slide 17, explain how to write a JSON object by focusing on the snippet. Tell them that in an object, a name/value pair is a member. The member name should always be a string in double quotes.

Slide 18

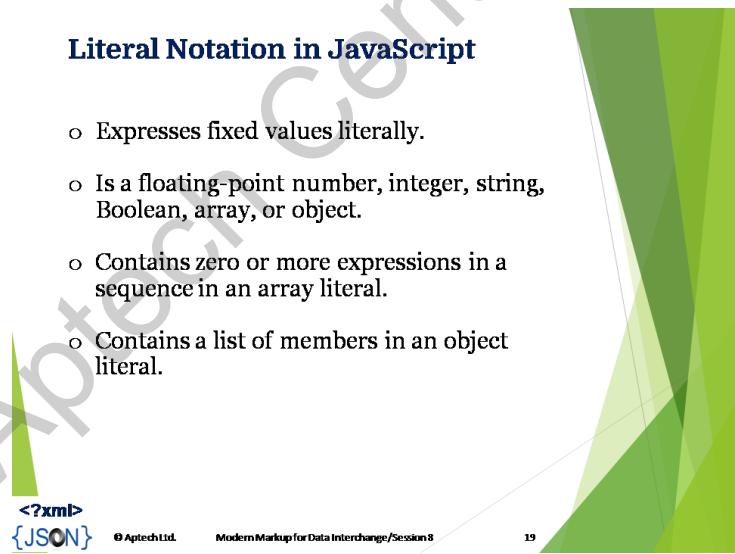
Let us now understand the syntax rules for arrays and objects.



Using slide 18, explain the syntax rules for arrays and objects to students.

Slide 19

Let us now understand literal notation in JavaScript.



Using slide 19, explain JavaScript literal notation to students. Tell them that in a programming language, literals are fixed values such as a string value “passed” and a constant integer value of 7. The literals are used in most languages for stating an expression. This expression can be an input parameter to a function, a value assigned to a variable, or a part of a condition in a control statement.

Slides 20 and 21

Let us now explore literal notation for arrays and objects in JavaScript.




**JavaScript Array Literal Notation
1-2**

- Following snippet defines an array using literal notation:

```
var fruits = ["Pineapple", "Orange", "Melon",
"Kiwi", "Guava"];

alert (fruits[4] + " is one of the " +
fruits.length + " fruits.");
```

**JavaScript Array Literal Notation
2-2**

- Following snippet defines an object using literal notation:

```
var Address = {
    "Street": "123 New Mansion
Street.",
    "City": "Denver",
    "PostalCode": 80123 };

alert ("The product will be sent to postal
code " + Address.PostalCode);
```

© Aptech Ltd. Modern Markup for Data Interchange/Session 8

20

© Aptech Ltd. Modern Markup for Data Interchange/Session 8

21

Using slides 20 and 21, explain how literal notation is used to define an array and object in JavaScript.

Additional Information:

For more information, visit the following link:
<http://www.crockford.com/javascript/survey.html>

Slide 22

Let us now explore literals in JSON.

From JavaScript Literals to JSON

- JSON has stricter rules:
 - Object member name to be a valid JSON string in quotation marks
 - A member value or an array element in JSON to be confined to a restricted set
- JSON does not support Date/time literals.



22

Using slide 22, explain about the stricter rules that JSON follows for literals in arrays and objects.

Slide 23

Let us now see how to create a simple JSON file.

Creating a JSON File

- The `JSON.stringify()` function converts a JavaScript value to a serialized JSON string.
- Following is the syntax:

```
JSON.stringify(value [, replacer] [, space])
where,
value identifies a JavaScript value.
replacer represents a function or array.
space adds line break and white space.
```

23

Using slide 23, explain the `JSON.stringify()` function to students. Focus on the syntax.

Additional Information:

For more information, visit the following link:

[https://msdn.microsoft.com/library/cc836459\(v=vs.94\).aspx](https://msdn.microsoft.com/library/cc836459(v=vs.94).aspx)

Slides 24 and 25

Let us summarize the session.

Summary 1-2

- JSON is a text-based and open standard format derived from JavaScript and ECMAScript.
- JSON is lighter, more scalable, less complex, quicker to learn, and faster to process than XML.
- JSON works independently of any language or platform.
- JSON is preferred for serializing and de-serializing data to be exchanged on the Web.
- Data in JSON format is represented as name/value pairs.

<?xml>
{JSON} © Aptech Ltd. Modern Markup for Data Interchange/Session 8

24

Summary 2-2

- JSON supports two data structures namely, array as ordered list and objects as unordered set of name/value pairs.
- While syntax for JavaScript's literal values is flexible, JSON follows stricter rules for arrays and objects.
- A JSON file is saved with .json extension.
- The `JSON.stringify()` function is used to transform a JavaScript object to a JSON string.

<?xml>
{JSON} © Aptech Ltd. Modern Markup for Data Interchange/Session 8

25

Using slides 24 and 25, summarize the session. Explain each of the following points in brief. Tell them that:

- JSON is a text-based and open standard format derived from JavaScript and ECMAScript.
- JSON is lighter, more scalable, less complex, quicker to learn, and faster to process than XML.
- JSON works independently of any language or platform.
- JSON is preferred for serializing and de-serializing data to be exchanged on the Web.

- Data in JSON format is represented as name/value pairs.
- JSON supports two data structures namely, array as ordered list and objects as unordered set of name/value pairs.
- While syntax for JavaScript's literal values is flexible, JSON follows stricter rules for arrays and objects.
- A JSON file is saved with .json extension.
- The `JSON.stringify()` function is used to transform a JavaScript object to a JSON string.

8.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 9: Work with JSON Data

9.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. Prepare a question or two that will be a key point to relate the current session objectives.

9.1.1 Objectives

By the end of this session, learners will be able to:

- Identify data types supported by JSON
- Explain how to represent complex data with JSON
- Explain how to execute serialization and de-serialization of JSON with JavaScript
- Describe tools and editors that can be used to work with JSON
- Describe the syntax and schema of a JSON document

9.1.2 Teaching Skills

To teach this session, you should be well versed with the different data types used in JSON and their syntax. You should be familiar with various data structures supported by JSON, JSON schema and its creation, and various JSON tools and editors.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students an overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

9.2 In-Class Explanations

Slides 3 to 7

Let us understand the different JSON data types.

JSON Data Types 1-4

The two primary data types supported by JSON are primitive and structure.

```

graph LR
    DT[Data Type] --> P[Primitive]
    DT --> S[Structure]
    P --> String[String]
    P --> Number[Number]
    P --> Boolean[Boolean]
    P --> Null[Null]
    S --> Array[Array]
    S --> Object[Object]
  
```

In JSON, the type of a variable is recognized automatically during the parsing phase.

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 9

3

JSON Data Types 1-4

Number

- It is a floating-point format (double precision).
- Does not accept Octal, hexadecimal, NaN, and Infinity values.
- The types allowed are integer, fraction, and exponent.
- **Syntax** {"string": number_value,}

String

- A series of zero or other Unicode characters within double quotes and backslash escapes.
- Delimited with double-quotation marks.
- A character denotes a single character string.
- **Syntax** {"string": "string value",}

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 9

4

JSON Data Types 2-4

Boolean

- Has only two values: true and false.
- Using quotes for Boolean values treats them as String values.
- **Syntax** {string: true/false,}

Null

- Is an empty type.

White Space

- A white space can appear between the characters in string values to make code more comprehensible.
- **Syntax** {string: " ",}

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 9

5

JSON Data Types 3-4

Arrays allow storing various values of the same type in one variable.

Syntax `[value,]`

The characteristics of an array in JSON are:

- It is a sequential collection of values, not necessarily of the same type.
- Indexing begins at 0 or 1.
- It is enclosed in square brackets [].
- Each value is set apart by comma (,).

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 9

6

JSON Data Types 4-4

An object is an independent data type, having its own attributes.

Syntax `{string : value,}`

The characteristics of an object in JSON are:

- It is a non-sequential (having no order) set of key/value pairs.
- It is enclosed in curly braces {}.
- Each key/value pairs are set apart by comma (,) and every key is proceeded by colon (:).
- The keys are only strings, each differing from the other.
- It is used when the key names are random strings.

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 9

7

Using slides 3 to 7, explain the classification of JSON data types. For each data type, list the characteristics and explain the syntax.

Additional Information:

For more information, visit the following links:

<https://www.postgresql.org/docs/9.4/static/datatype-json.html>

http://www.tutorialspoint.com/json/json_data_types.htm

Slide 8

Let us understand about JSON value.

JSON Value

- In JSON, value can be of any primitive and structure data type.
- A JSON value can be a number, string, Boolean such as true or false, null, object, or an array.
- The following Code Snippet shows some examples of JSON values:

```
var emp-no = 1234;
var name = "Sherill";
var experience = null;
```



<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 9 8

Using slide 8, explain how the data types are used. Explain some examples given on the slide.

Slide 9

Let us now understand how to store different values in arrays.

Storing Different Values in Arrays

JSON arrays can store elements of different types. Following is a sample code:

```
[ 456, "Dog", 123, "Frog", true ]
```

- Elements 0 and 2 in the array are of the type Number.
- Elements 1 and 3 are of String type.
- Element 4 is a Boolean type.

In JSON, arrays can also contain other arrays. This is called nesting of arrays.

<?xml>
{JSON} © Aptech Ltd. Modern Markup for Data Interchange/Session 9 9

Using slide 9, explain to the students that JSON not only allows storing different values in arrays and objects but also enables you to nest them. To make it easily understandable for the students, you can explain the concept using the example provided on the slide.

Using the example given in Figure 9.2 in the Learner Guide, explain that in JSON, arrays can also contain other arrays and this is called nesting of arrays.

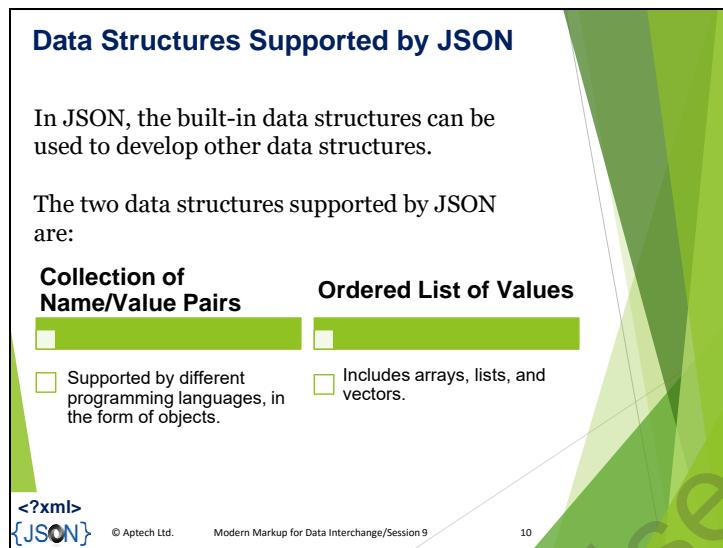
Using the following example, explain the concept of nesting array in a JSON Object. Explain that the code shows a JSON blog object storing user details along with an array of tags. The array contains the tag details of blog contents in an array. These tag details help in searching the content on a blog.

```
{
  "blogID":12345,
  "firstName":"James",
  "lastName":"Gunn",

  "blogContent":"This is my first JSON object and it looks simple",
  "tags":[
    "first",
    "JSON",
    "looks",
    "simple"
  ]
}
```

Slide 10

Let us understand about the data structures supported by JSON.



Using slide 10, tell the students that data structures are an integral part of all the programming languages. Explain the two data structures supported by JSON.

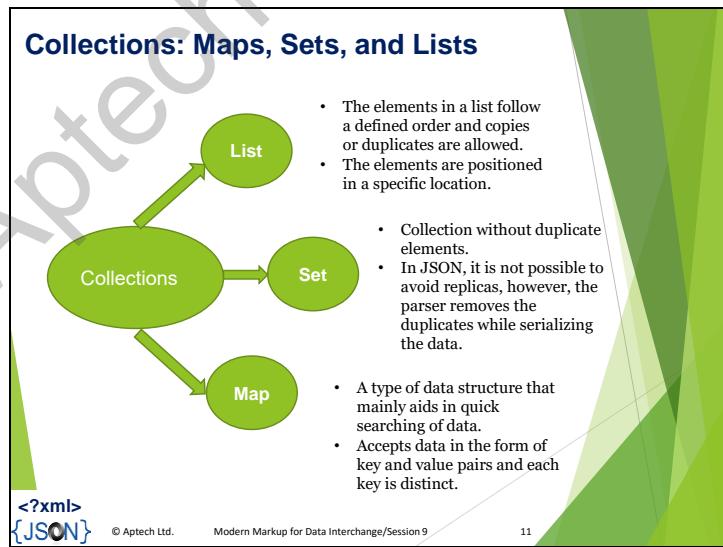
Additional Information:

For more information, visit the following link:

https://adobe.github.io/Spry/samples/data_region/JSONDataSetSample.html

Slide 11

Let us understand about collections.

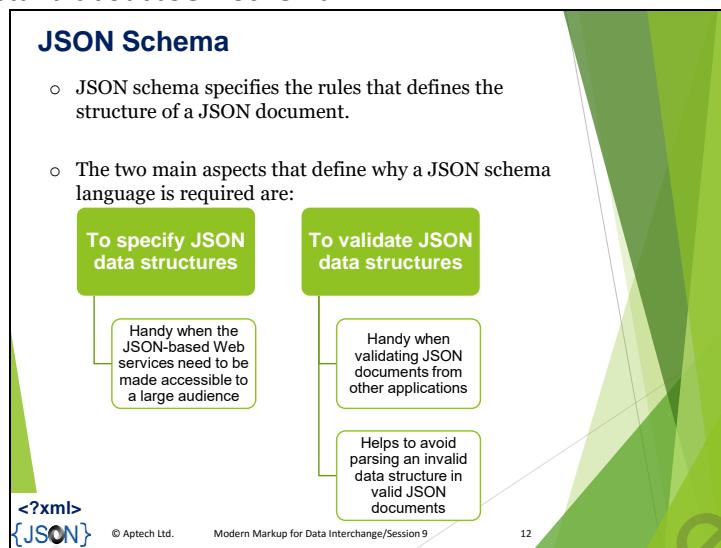


Using slide 11, explain that sets and lists correspond to a set of objects, whereas maps relate a value to an object. Provide a description of each of these data structures and explain their features.

Inform the students that it is the parser that takes up the main role of serializing and de-serializing JSON data. The JSON parser maps the core data structure with the internal JSON structure.

Slide 12

Let us now understand about JSON Schema.



Using slide 12, explain JSON Schema to the students and the two main aspects that define why a JSON schema language is required.

As additional knowledge, you can tell the students that JSON schema is based on the XML Schema Definition (XSD) and is just an Internet draft; the latest version 4 expired on August 4th, 2013. Though it has expired, application developers still use the concept for generating JSON document, which is a contract between different applications or producers and consumers of Web services.

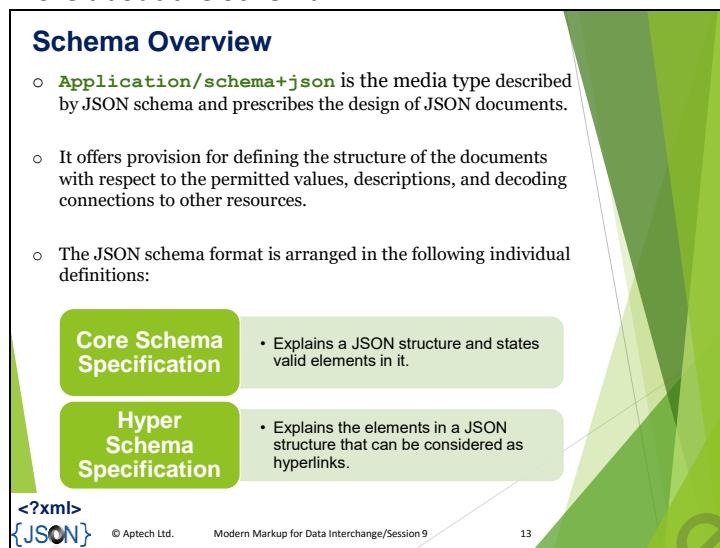
Additional information:

For more information, visit the following link:

<http://json-schema.org/documentation.html>

Slide 13

Let us understand more about the Schema.



Schema Overview

- **Application/schema+json** is the media type described by JSON schema and prescribes the design of JSON documents.
- It offers provision for defining the structure of the documents with respect to the permitted values, descriptions, and decoding connections to other resources.
- The JSON schema format is arranged in the following individual definitions:
 - Core Schema Specification
 - Hyper Schema Specification

Core Schema Specification

- Explains a JSON structure and states valid elements in it.

Hyper Schema Specification

- Explains the elements in a JSON structure that can be considered as hyperlinks.

<?xml>
{JSON}

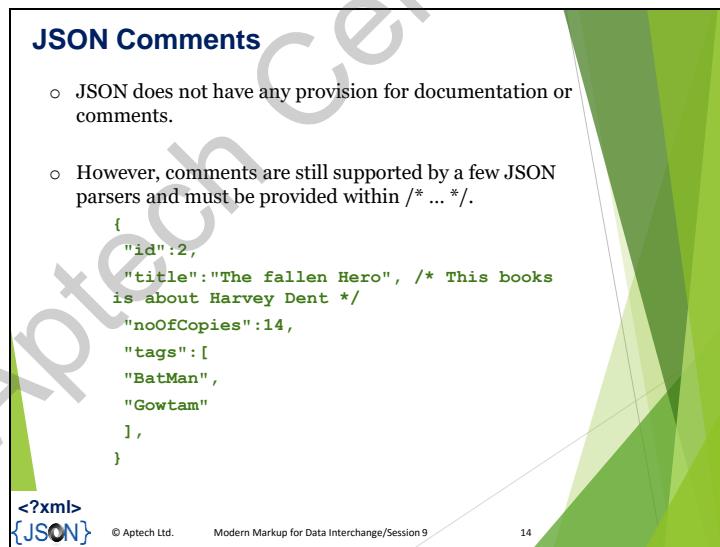
© Aptech Ltd. Modern Markup for Data Interchange/Session 9 13

Using slide 13, provide details about the schema and how the JSON schema format is arranged.

At this stage, you can convey the concept of a schema using a Library Management System (LMS) as an example.

Slide 14

Let us understand about comments.



JSON Comments

- JSON does not have any provision for documentation or comments.
- However, comments are still supported by a few JSON parsers and must be provided within /* ... */.

```
{
  "id":2,
  "title":"The fallen Hero", /* This books
is about Harvey Dent */
  "noOfCopies":14,
  "tags":[
    "BatMan",
    "Gowtam"
  ],
}
```

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 9 14

Using slide 14, introduce comments to the students. Explain that comments are not supported, as its developer Douglas Crockford found users utilizing them to store parsing directives.

However, you can still use a workaround to give comments within a JSON document, although you need to delete all before a parser processes it.

Slide 15

Let us now understand how to create and parse JSON messages with JavaScript.

Creating and Parsing JSON Messages with JavaScript

- The `JSON.stringify()` method allows converting a JavaScript object into a JSON String.
- The `JSON.parse()` method allows parsing a JSON object using JavaScript.
- It converts a JSON String to an object in JavaScript.
- Start by defining a string in JSON format, using the method for conversion, and then looping through the attributes for printing its values.

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 9 15

Using slide 15, explain how to create and parse JSON messages with JavaScript. Use relevant code snippets to demonstrate how a JavaScript object is converted to a JSON string using the `JSON.stringify()` method and how a JSON object is converted to a JavaScript object in an HTML file using the `JSON.parse()` method.

Slide 16

Let us understand about developer tools on browser.

JSON with Developer Tools on Browser

- There are many plugins or extensions that help in validating and formatting a JSON document or a JSON HTTP response.
- One such extension in Firefox is `JSONView`, which allows viewing a JSON document.
- With `JSONView` the JSON document is displayed in the browser.
- `JSONView` displays the raw text even though the JSON document has errors.
- Chrome also offers `JSONView` for validating a JSON document.
- For modifying the JSON values during runtime, it is recommended converting the JSON document to a JavaScript object before using the built-in browser developer tools.

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 9 16

Using slide 16, give details about the various plugins or extensions that help in validating and formatting a JSON document or a JSON HTTP response. You can show how `JSONView` in Firefox and Chrome allow to view a JSON document.

Slide 17

Let us understand about online tools and editors.

The screenshot shows the JSONLint interface. At the top, it says "Online Tools and Editors". Below that is a bulleted list of points about JSONLint. One point includes a screenshot of an error message from JSONLint, which reads:

```
Error: Parse error on line 4:
...": {
    "firstname": "Janes",
    "lastname"
    .....
}
Expecting 'STRING', 'NUMBER', 'NULL', 'TRUE', 'FALSE', '(', ')', got: 'undefined'
```

Another point shows a successful validation result:

```
Valid JSON
```

At the bottom left, there are buttons for XML and JSON. At the bottom right, it says "© Aptech Ltd. Modern Markup for Data Interchange/Session 9" and has a page number "17".

Using slide 17, explain that there are many online tools and editors for validating JSON data and the most popular online validator is JSONLint. You can demonstrate how this works by copying the JSON data to any online editor or asking the students to test this for themselves whenever they have access to a browser.

Additional Information:

For more information, visit the following link:

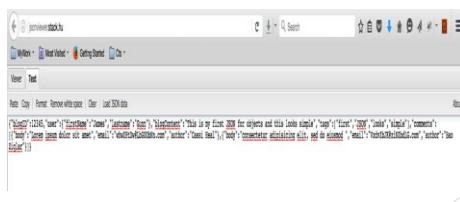
<http://jsonlint.com/>

Slides 18 and 19

Let us understand about online JSON viewers.

Online JSON Viewers 1-2

- While working with JSON, it is often required to use a JSON viewer to see how the document will look in the browser.
- One of the most widely used is jsonviewer.stack.hu and JSON text can be directly copied to this online viewer.
- The following screenshot shows the **jsonviewer.stack.hu**:



jsonviewer.stack.hu

Viewer Test

File Copy Format Remove whitespace Clear last JSON data

```
[{"id":12345,"name":"Facebook","category":"Social"}, {"id":56789,"name":"Twitter","category":"Social"}, {"id":34567,"name":"Instagram","category":"Social"}, {"id":98765,"name":"LinkedIn","category":"Social"}, {"id":112233445566778899,"name":"YouTube","category":"Video"}, {"id":223344556677889900,"name":"TikTok","category":"Video"}, {"id":778899001122334455,"name":"Snapchat","category":"Image"}, {"id":44556677889900112233,"name":"Pinterest","category":"Image"}, {"id":667788990011223344,"name":"Tumblr","category":"Image"}, {"id":889900112233445566,"name":"Reddit","category":"News"}, {"id":5566778899001122334455,"name":"Medium","category":"News"}, {"id":33445566778899001122334455,"name":"Hacker News","category":"News"}, {"id":112233445566778899001122334455,"name":"GitHub","category":"Code"}]
```

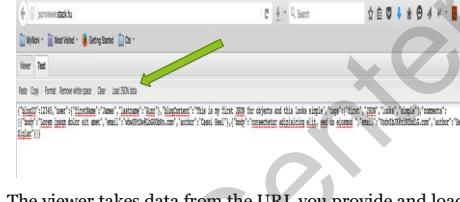
<?xml>

{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 9 18

Online JSON Viewers 2-2

- Another way of supplying data to the viewer is by clicking **Load JSON data**.
- The viewer takes data from the URL you provide and loads data from that source.
- Once the JSON code is visible in the viewer, clicking **Format** formats the code.



jsonviewer.stack.hu

Viewer Test

File Copy Format Remove whitespace Clear last JSON data

Load JSON data

```
[{"id":12345,"name":"Facebook","category":"Social"}, {"id":56789,"name":"Twitter","category":"Social"}, {"id":34567,"name":"Instagram","category":"Social"}, {"id":98765,"name":"LinkedIn","category":"Social"}, {"id":112233445566778899,"name":"YouTube","category":"Video"}, {"id":223344556677889900,"name":"TikTok","category":"Video"}, {"id":778899001122334455,"name":"Snapchat","category":"Image"}, {"id":44556677889900112233,"name":"Pinterest","category":"Image"}, {"id":667788990011223344,"name":"Tumblr","category":"Image"}, {"id":889900112233445566,"name":"Reddit","category":"News"}, {"id":5566778899001122334455,"name":"Medium","category":"News"}, {"id":33445566778899001122334455,"name":"Hacker News","category":"News"}, {"id":112233445566778899001122334455,"name":"GitHub","category":"Code"}]
```

<?xml>

{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 9 19

Using slides 18 and 19, explain about the jsonviewer.stack.hu tool which helps to view how the document looks in the browser. As it is an online tool, you can practically show the students how it works.

Slide 20

Let us summarize the session.

Summary

- The two primary data types supported by JSON are primitive and structure.
- The primitive types are String, Number, Null, and Boolean. The structure types are Array and Object.
- Arrays allow storing various values of the same type in one variable.
- An object is an independent data type, having its own attributes.
- In JSON, arrays can also contain other arrays. This is called nesting of arrays.
- The two data structures supported by JSON are Collection of Name/Value Pairs and Ordered List of Values.
- JSON schema specifies the rules that defines the structure of a JSON document.
- JSON does not have any provision for documentation or comments.
- **JSONLint** is an open source project that helps in validating JSON data.

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 9 20

Using slide 20, summarize the session. Explain each of the following points in brief. Tell them that:

- The two primary data types supported by JSON are primitive and structure.
- The primitive types are String, Number, Null, and Boolean. The structure types are Array and Object.
- Arrays allow storing various values of the same type in one variable.
- An object is an independent data type, having its own attributes.
- In JSON, arrays can also contain other arrays. This is called nesting of arrays.
- The two data structures supported by JSON are Collection of Name/Value Pairs and Ordered List of Values.
- JSON schema specifies the rules that defines the structure of a JSON document.
- JSON does not have any provision for documentation or comments.
- **JSONLint** is an open source project that helps in validating JSON data.

9.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session.

Tips: You can also check the **Articles/Blogs/Expert Videos** uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 10: JSON in Real World

10.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. Prepare a question or two that will be a key point to relate the current session objectives.

10.1.1 Objectives

By the end of this session, learners will be able to:

- Describe the support of different browser and programming languages for JSON
- Describe JSON content types
- Explain how to use JSON for Web and Data Storage
- Compare JSON with relational databases
- Identify security and data portability issues with respect to JSON

10.1.2 Teaching Skills

To teach this session, you should be well versed with support for JSON by different browsers and programming languages, especially C# and Java. You should be familiar with the JSON content types, using JSON for Web and data storage, and differences between JSON and relational as well as NoSQL databases. You should also be aware of security and portability issues in JSON.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities

Follow the order given here during In-Class activities.

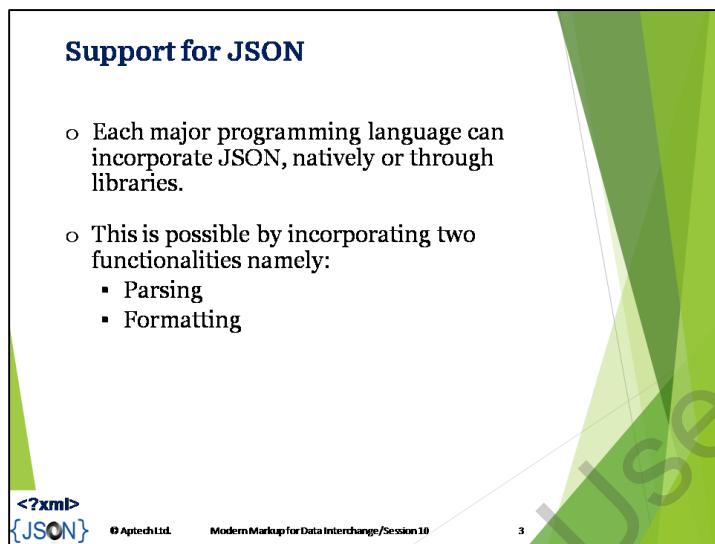
Overview of the Session:

Give the students an overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

10.2 In-Class Explanations

Slide 3

Let us understand the support for JSON in different programming languages.



Using slide 3, explain how JSON is supported in different programming languages through parsing and formatting functionalities. Tell students that any major programming language can incorporate JSON either through libraries or native support. However, none of them can bypass JavaScript for parsing JSON, which converts it into an object literal.

Each major language has powerful libraries for parsing and formatting data in JSON format. Parsing converts JSON data into the data structure supported by the corresponding language. This structure can be an array, hash, or dictionary. Formatting converts array, hash, or dictionary to JSON text.

Additional Information:

For more information, visit the following link:

<https://www.quora.com/With-which-programming-language-does-JSON-pair-best>

Slide 4

Let us understand the use of JSON with C# and Java.

JSON with C# and Java

- o These languages support statically typed classes and not objects of HashMap or Dictionary type.
- o The solution is to use a library along with a custom code for converting these structures into static type instances.
- o The Gson library from Google is one such library that resolves the issue.

<?xml>
{JSON}

Aptech Ltd. Modern Markup for Data Interchange / Session 10 4

Using slide 4, tell students that a few developers do not feel it feasible to use JSON in C# or Java. This is because the languages support statically typed classes instead of Dictionary or HashMap objects. Thus, developers would need a library for converting these structures to static instances. One such library is Gson from Google, which transforms JSON data to Java objects.

Slide 5

Let us now understand Gson.

What is Gson

- Is an open-source Java library for transforming a Java object to JSON data and vice-versa.
- Offers easy mechanisms such as constructor (factory method) and `toString()`.
- Functions well with arbitrary Java objects, involving the pre-existing ones.
- Serializes and deserializes huge data without any issues.
- Is convenient to learn and use by only using `toJson()` and `fromJson()`.

<?xml>
{JSON}

Aptech Ltd. Modern Markup for Data Interchange / Session 10 5

Using slide 5, briefly explain Gson to students. Finally, tell them that the goals of Gson are to convert already existing objects to and from JSON, allow custom representations for objects, and giving readable and compact JSON data.

Additional Information:

For more information, visit the following link:

<https://sites.google.com/site/gson/gson-user-guide>

Slide 6

Let us see how JSON is used in PHP.

JSON with PHP

- From 5.2.0 version, the extension for JSON is packaged into PHP.
- Following table shows the functions for encoding and decoding JSON structures:

| Function | Description |
|-----------------|---|
| json_encode | Serializes the stated array or object and returns it in the JSON format if successful or FALSE. |
| json_decode | Deserializes JSON data and returns the suitable PHP type. |
| json_last_error | Returns the error that happened last. |

<?xml>
[JSON]

Aptech Ltd. Modern Markup for Data Interchange / Session 10 6

Using slide 6, explain the functions used in PHP for encoding and decoding JSON structures.

Additional Information:

For more information, visit the following links:

<http://php.net/manual/en/book.json.php>

<http://www.phpbuilder.com/articles/application-architecture/object-oriented/processing-json-in-php.html>

Slide 7

Let us now explore the MIME or content type of JSON.

MIME Type of JSON

- Is also called media type or content type.
- Is a two-part identifier composed of a type and subtype isolated by a slash.
- Aids in identifying the type of formatted content being sent over the Web.
- Is 'application/json' for JSON.
- Is unofficially 'text/json' or 'text/Javascript'.

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 10

7

Using slide 7, explain MIME type to students. Focus on its definition and its official as well as unofficial form for JSON.

Slide 8

Let us now explore the applications of JSON.

Applications of JSON

| | |
|---|---|
| APIs | • For exchanging data to and from APIs
• Popular in social networking sites implementing API |
| NoSQL | • For storing data easily in NoSQL databases, as JSON is easily convertible into JavaScript |
| Asynchronous JavaScript and JSON (AJAJ) | • For replacing XML when new data is fetched by a loaded Web page in a browser |
| JSON-RPC (Remote Procedure Call) | • For replacing Simple Object Access Protocol (SOAP) or XML-RPC and for sending several calls and notifications to a server |
| Package Management | • For storing metadata |

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 10

8

Using slide 8, explain various applications of JSON to students. Tell them about NoSQL databases in brief. Then, state that AJAJ is similar to AJAX but uses JSON and not XML. AJAJ, as a dynamic Web development methodology, allows a Web page to ask and display new data. Next, the SOAP protocol defines some commands and data types for allowing a system to send multiple calls and notifications to the server. Packages make it simpler to deploy the applications and integrate any changes, later. Many package management tools

implement the package .json file having metadata such as version, name, description, and license details.

Additional Information:

For more information, visit the following link:

<http://www.w3resource.com/JSON/introduction.php>

In-Class Question:

After you finish explaining the applications of JSON, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

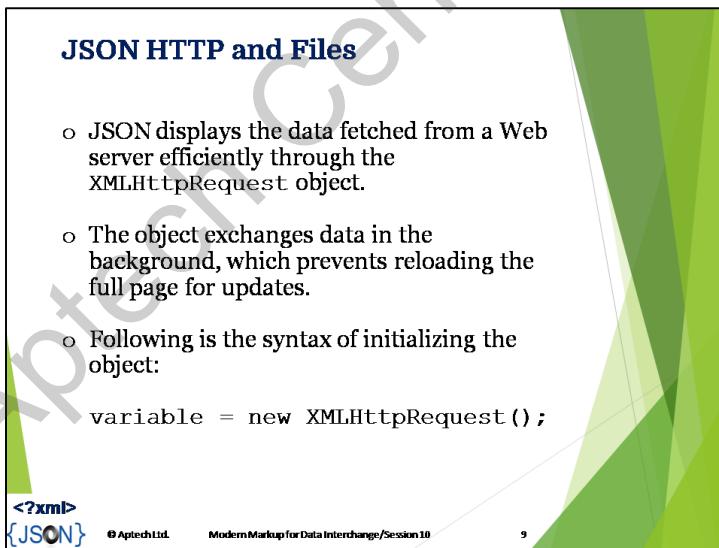


Why it is easy to work with JSON for storing data into a database?

Answer: It is easy to work with JSON for storing data into a database because a developer can convert a JSON structure to a compatible object structure in the browser. This converted structure is also easy to integrate in Server-side script for storing the data in the server's database.

Slide 9

Let us now understand how JSON is useful in displaying data fetched from a Web server on a Web page.



JSON HTTP and Files

- JSON displays the data fetched from a Web server efficiently through the XMLHttpRequest object.
- The object exchanges data in the background, which prevents reloading the full page for updates.
- Following is the syntax of initializing the object:
`variable = new XMLHttpRequest();`

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange / Session 10 9

Using slide 9, explain how JSON uses the XMLHttpRequest object to display the data from a Web server. Focus on the syntax.

Additional Information:

For more information, visit the following links:

http://enterprisewebbook.com/ch2_ajax_json.html

http://www.tutorialspoint.com/ajax/what_is_xmlhttprequest.htm

In-Class Question:

After you finish explaining how JSON is useful in displaying data fetched from a Web server, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Why the XMLHttpRequest object is efficient?

Answer: The XMLHttpRequest object is efficient, as it exchanges data with the server in the background such that there is no need to reload the full page for updating just a few sections.

Slide 10

Let us understand why JSON is used to store data.

JSON for Data Storage

JSON helps in storing data fetched from services and applications.

- **Reasons:** Simplicity and ‘just adequate structure’

RDBMS and Structured Query Language (SQL) are now replaced by NoSQL databases for developers who prefer JSON.

- **Trends:** More implementation of JSON-centric document databases, JSON in traditional RDBMS
- **Reasons:** Developer-friendly and agility

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange / Session 10 10

Using slide 10, explain why JSON is useful even in storing data. The long control of RDBMS and SQL has come to an end with NoSQL databases, which are catering to the needs of developers who prefer JSON. Database vendors have also started offering JSON-centric document databases. Many NoSQL database vendors prefer JSON over conventional relational schemes in Oracle, MySQL, MS SQL Server, and PostgreSQL. This is ideal for those who want JSON data interchange format for their applications. JSON records are easily extensible and legibly structured, making the format agile. Further, the absence of predefined schema enables easier upgrades.

Additional Information:

For more information, visit the following link:

<http://www.infoworld.com/article/2608293/nosql/how-json-sparked-nosql---and-will-return-to-the-rdbms-fold.html>

Slide 11

Let us now compare document and relational databases.

| Document versus Relational Databases | | |
|--|---|--|
| | Document-oriented Databases | Relational Databases |
| Coupling | Tighter due to the need to navigate the document while querying | Loose |
| Portability and Standardization | No, due to new query language setup for each document store | Yes, due to well-defined SQL query core |
| Optimization | Restricted, due to no abstraction between the logical data structure and its physical storage | Fully optimized, due to abstraction ensured by highly queryable stored data definition |
| Consistency and Normalization | No, due to no support for constraints | Yes |

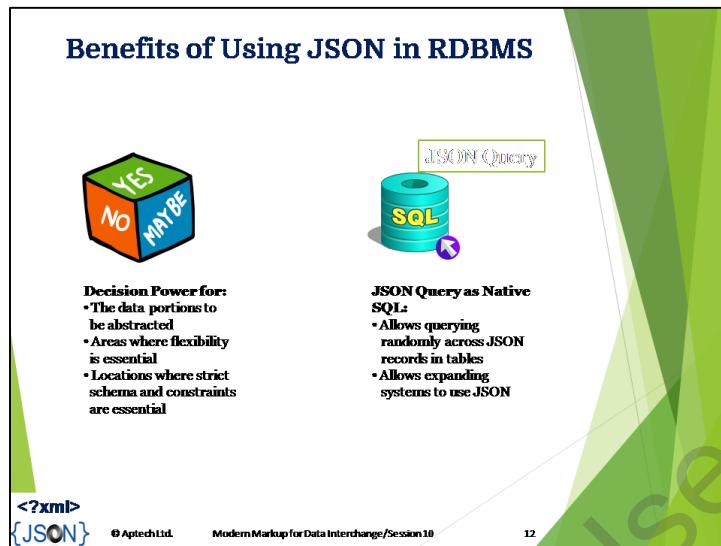
<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 10 11

Using slide 11, explain the differences between document and relational databases.

Slide 12

Let us explore the benefits of using JSON in an RDBMS.



Using slide 12, explain the benefits of using JSON in an RDBMS.

Slide 13

Let us now distinguish between JSON and RDBMS.

JSON versus RDBMS Data Models

| | JSON | RDBMS |
|--------------------------|--|--|
| Storage Structure | Is array or object. | Is a table. |
| Metadata | Can be in a schema, but is not pre-created. | Is stored in a schema generated while creating a table. |
| Data Retrieval | Uses evolving languages namely, JSON Query Language (JQL) and JSONiq | Uses SQL. |
| Sorting | Is only for arrays. | Is for tables. |
| Learning Curve | Is smoother. | Is time consuming. |
| Application | Is used in several programming languages. | Is in the form of many commercial and open-source databases. |

<?xml> {JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 10

13

Using slide 13, explain the differences between JSON and RDBMS to students. Tell them that JSON has a much smooth learning curve, as the supported structures and data types are used in several programming languages. This makes it quick for a developer with basic programming background to understand JSON concepts and coding syntax.

In-Class Question:

After you finish explaining the distinction between JSON and RDBMS, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Why is the learning curve for JSON easier than an RDBMS?

Answer: The learning curve for JSON is easier than an RDBMS because the data types and structure of JSON are supported and used in several programming languages. This makes it quicker for a developer to grasp JSON concepts. However, RDBMS is a distinct and vast field to understand.

Slide 14

Let us now explore security issues with JSON.

The slide has a green decorative border. At the top, the title 'Security Issues in JSON' is displayed. Below the title, there are two bullet points:

- JSON, as a flexible subset of JavaScript, has some security issues.
- In the following figure the root cause increasing the risk of malicious script, a major issue:

Below the second bullet point, there is a code snippet:

```
var data = eval('(' + JSONresponse + ')');
```

An arrow points from the text 'increasing the risk of malicious script, a major issue:' to the `eval` function call in the code snippet. Below the code, there is a black rectangular box containing a warning message:

```
<?xml>
{JSON}

    © Aptech Ltd. Modern Markup for Data Interchange / Session 10 14

```

The message in the alert box reads:

!alert('Alert: malware has been detected on your computer. Visit cleanmyinfectedcomputer.com to clean your computer.');

Using slide 14, tell students that JSON code or data has some security issues. A major issue is the increasing risk of malicious script being injected inside the code. Focus on the figure given on the slide.

Slide 15

Let us understand the issue with the eval () function.

Issues with Eval()

- o Most JSON text is syntactically JavaScript.
- o JavaScript interpreter converts JSON into a JavaScript object without validation.
- o This increases the risk of authentication forgery, identity and data theft, and misuse of resources.
- o **Solution:** JSON.parse() and JSON.stringify() functions processing text only in JSON format

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 10 15

Using slide 15, explain the major security issue of malicious script is triggered by using the eval () function of JavaScript. Most JSON text is JavaScript in terms of syntax due to which the function parses JSON data. This means there is no JSON-based parser used and that directly the JavaScript interpreter executes the JSON code for generating JavaScript objects. This is actually risky because there is no validation happening before parsing. Such data is thus vulnerable to security issues. For instance, it can be exposed to malicious code injection attacks for authentication forgery and data theft. Finally, focus on the solutions.

JSON.parse () processes text only in JSON format.

Additional Information:

For more information, visit the following links:

https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/JSON/pa
rse

<http://stackoverflow.com/questions/17785592/difference-between-json-stringify-and-json-parse>

Slide 16

Let us now explore the security breaches to which JSON code or format can be easily exposed.

XSS and CSRF Issues

Cross Site Scripting (XSS)

Request: http://vulnerablesite.local/index?name=<script>alert('xss')</script>

Response:

```
<html>
  <body>
    <div>
      Hello <script>alert('xss')</script>
    </div>
  </body>
</html>
```

Cross Site Request Forgery (CSRF)

Request: http://vulnerablesite.local/changepassword?newpwd=MyS3cr3tPa\$\$word

Attack:

The slide also features decorative green triangles on the left and right sides, and a footer with '<?xml>' and '{JSON}' icons.

Using slide 16, explain students about two major security breaches namely, Cross-site Scripting (XSS) and Cross-site Request Forgery (CSRF). XSS involves inserting random HTML (along with JavaScript) into Web pages for usually bypassing access control mechanisms. In CSRF, unauthorized commands are sent by a user that seems to be authorized to the site.

Additional Information:

For more information, visit the following link:

<http://flask.pocoo.org/docs/0.11/security/>

In-Class Question:

After you finish explaining the security breaches, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

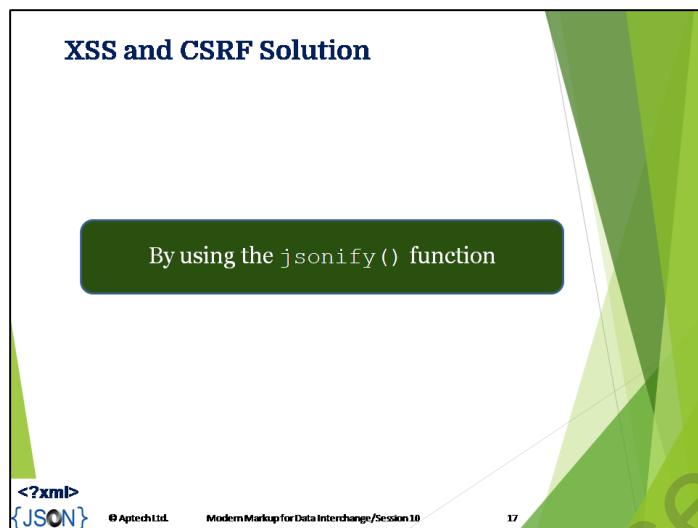


What is the difference between XSRF and XSS?

Answer: In XSS, the user's trust for a specific site is exploited. However, in CSRF, the site's trust for a user's browser is exploited.

Slide 17

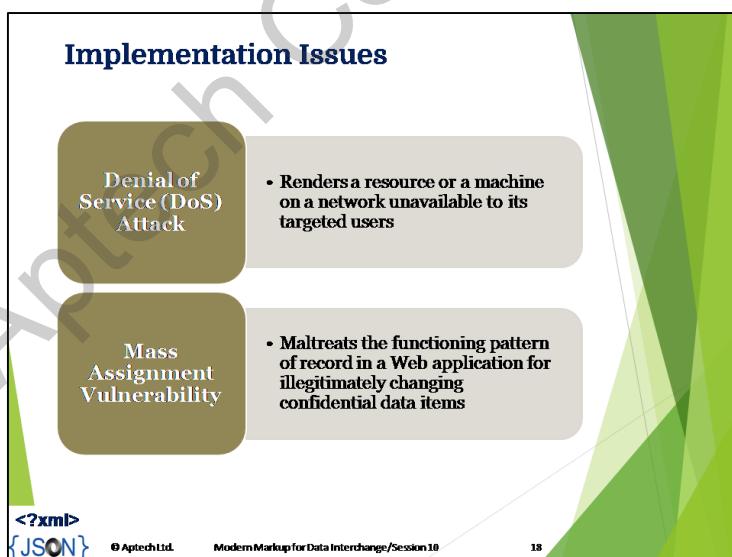
Let us now see the solution for the two security breaches.



Using slide 17, tell students that a strange section of JavaScript specification can keep these breaches away but only if the `jsonify()` function is used for generating JSON structures. Well, this means that the risk still exists if other methods are used for generating JSON structures.

Slide 18

Let us now see some implementation issues related to JSON.



Using slide 18, explain the two implementation issues with JSON to students.

Additional Information:

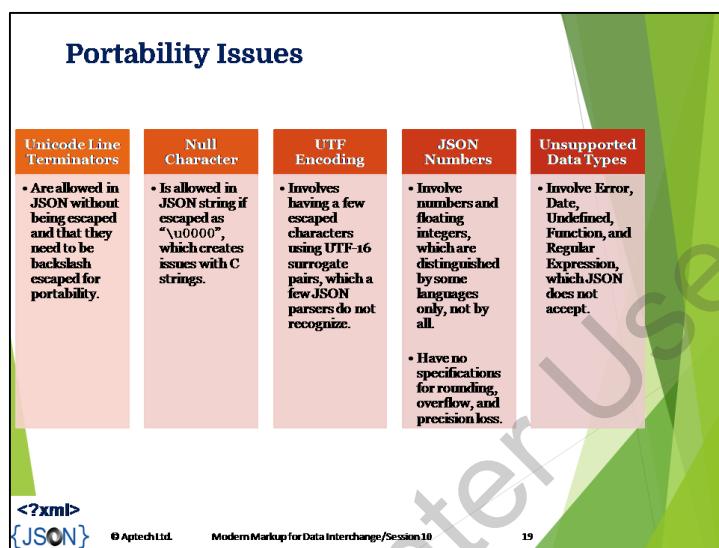
For more information, visit the following links:

https://en.wikipedia.org/wiki/Denial-of-service_attack

https://en.wikipedia.org/wiki/Mass_assignment_vulnerability

Slide 19

Let us now see some portability issues related to JSON.



Using slide 19, explain the five portability issues with JSON to students.

Additional Information:

For more information, visit the following link:

https://www.reddit.com/r/Unity3D/comments/30cles/managing_a_large_number_of_small_levels_in_using/

Slide 20

Let us now see how to handle JSON securely.

Handling JSON Securely

- o Avoiding eval() by using a JavaScript library available at www.json.org, such as JSON sans eval()
- o Ensuring data integrity via XMLHttpRequests

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 10 20

Using slide 20, explain the students how to handle JSON securely by avoiding the eval() function and using XMLHttpRequests.

Additional Information:

For more information, visit the following link:

<http://yuiblog.com/blog/2007/04/10/json-and-browser-security/>

Slides 21 and 22

Let us summarize the session.

Summary 1-2

- o Modern programming languages incorporate JSON via libraries or native parsing support.
- o Gson is an open-source Java library for converting a Java object into JSON and vice-versa.
- o Formal MIME type for JSON text is 'application/json'.
- o JSON is used in several applications, such as in APIs, NoSQL databases, RDBMS', Web development, and application packages.

<?xml>
{JSON}

© Aptech Ltd. Modern Markup for Data Interchange/Session 10 21

Summary 2-2

- Unlike relational databases, JSON does not have tables, pre-created metadata, and support for SQL.
- Using `eval()` for parsing JSON data can lead to security attacks such as XSS and CSRF.
- JSON is prone to DoS attack and vulnerability of mass assignment.
- Developers can secure JSON structures by replacing `eval()` with a JavaScript library and/or using XMLHttpRequests.



© Aptech Ltd. Modern Markup for Data Interchange/Session 10 22

Using slides 21 and 22, summarize the session. Explain each of the following points in brief. Tell them that:

- Modern programming languages incorporate JSON via libraries or native parsing support.
- Gson is an open-source Java library for converting a Java object into JSON and vice-versa.
- Formal MIME type for JSON text is ‘application/json’.
- JSON is used in several applications, such as in APIs, NoSQL databases, RDBMS’, Web development, and application packages.
- Unlike relational databases, JSON does not have tables, pre-created metadata, and support for SQL.
- Using `eval()` for parsing JSON data can lead to security attacks such as XSS and CSRF.
- JSON is prone to DoS attack and vulnerability of mass assignment.
- Developers can secure JSON structures by replacing `eval()` with a JavaScript library and/or using XMLHttpRequests.

10.3 Post Class Activities for Faculty

Tips: You can check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the session.