KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act 1956)
Coimbatore-641 021
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMPUTER SCIENCE, CA & IT

SUBJECT CODE : 17CSU404B **SUBJECT : XML PROGRAMMING**
SEMESTER : IV **CLASS : II B.Sc. CS** **L T P C = 3 0 0 3**

Course Outcome

This course relates to the interface between web servers and their clients. The course provides information includes markup languages, programming interfaces and languages, and standards for document identification and display. The use of Web technology makes to enhance active student learning and improves their creativity in making web pages.

Learning outcome

- To Create a new webpage
- To understand the fundamental features of web applications.
- To understand the objects and components needed for a web designing.

UNIT-I

Introduction: Understanding Mark-up Languages, Introduction to XML and its Goals.

UNIT-II

XML Basics: XML Structure and Syntax, Document classes and Rules.

UNIT-III

Other XML Concepts: Scripting XML

UNIT-IV

Other XML Concepts: XML as Data, Linking with XML

UNIT-V

XML with Style: XSL –Style Sheet Basics, XSL basics, XSL style sheets.

Suggested Readings

1. William, J. Pardi. XML in action web technology.
2. Michael, J. Young. Step by Step XML.

Websites:

1. https://www.w3schools.com/xml/xml_examples.asp

3. https://www.xml.com/pub/a/2002/10/30/qa.html

4. https://study.com/.../web-page-design-and-programming-languages-html-xhtml-xml-co

ESE MARKS ALLOCATION

1.	**Section A** 20 x 1 = 20	20
2.	**Section B** 5 x 2 = 10	10
3.	**Section C** **5 x 6= 30** Either 'A' or 'B' choice	30
	Total	60

KAHE

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore - 641021.

(For the candidates admitted from 2017 onwards)

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

SUBJECT : **XML PROGRAMMING**

SEMESTER :**IV** **L T P C**

SUBJECT CODE:**17CSU404B** **CLASS** :**II B.Sc CS** **3 0 0 3**

LECTURE PLAN

STAFF NAME: D.MANJULA

S.No	Lecture Duration (Hr)	Topics	Support Materials
		UNIT-I	
1.	1	• **Introduction:** Understanding Mark-up Languages • A Brief History of Markup Languages	T1:1-4 W1
2.	1	• How Markup Works • Specific and Generalized Markup Languages	T1:5, W1, T1:10, W1, R1:7-8
3.	1	• The Big Markup Picture	T1:14, W3, R2:8-9
4.	1	• **Enter XML:** What Is XML? • Where Does XML Fit In?,The Goals of XML	T1:15-16 T2:3-12 ,T1:16-23, T2:12-14
5.	1	• XML, Recommendations, and Standards	T1:26-28
6.	1	• **Recapitulation and discussion of Important questions**	
		Total No of Hours Planned For Unit – I	**6**

UNIT-II			
1.	1	• **XML Basics:** XML Structure and Syntax	T1:29-40, R1:35-36,W3,W5
2.	1	• **Playing by the Rules**- The DTD	T1:45,W6
3.	1	• Document Classes	T1:46-47, W7
4.	1	• The Document Type Definition	T1:48-72
5.	1	• **Recapitulation and discussion of Important questions**	
		Total No of Hours Planned For Unit – II	**5**
UNIT-III			
1.	1	• **Other XML Concepts:**Putting XML to Work, Scripting XML, A Scripting Refresher	T1: 73-76
2.	1	• The XML Processor • XML: The Parent/Child Relationship	T1:77-81,W2
3.	1	• From XML to HTML	T1: 84-94
4.	1	• **Beyond the Basics**-More Scripting Techniques	T1:95-100,W8
5.	1	• **Recapitulation and discussion of Important questions**	
		Total No of Hours Planned For Unit – III	**5**
UNIT-IV			
1.	1	• **Other XML Concepts:** XML as Data	T1:101,W7
2.	1	• Data Typing in XML , XML Namespaces	T1:101-115, W7
3.	1	• Using the XML Data Source Object, Putting It All Together	T1:116-136
4.	1	• **Linking with XML:** Simple Links the HTML Way	T1: 137-138, R2: 18-20
5.	1	• XLink: The XML Linking Mechanism, XPointer-Looking Inward	T1:143-160, R2: 20-23
6.	1	• **Recapitulation and discussion of Important**	

		questions	
		Total No of Hours Planned For Unit – IV	**6**
		UNIT-V	
1.	1	• **XSL:XML with Style-** Style Sheet Basics, XSL Basics ,XSL Style Sheets	T1:161-166
2.	1	• **Addressing Data with XSL Patterns:** What Is XSL Patterns?,XSL Patterns Language Syntax	T1:195-203, W2
3.	1	• The XSL Patterns Object Model , More on XSL Patterns , XML-Data, Schemas at Work , Advanced Topics, Only the Beginning	T1:222-228, T1:236-249, W3
4.	1	• XML Today and Tomorrow, The XML Object Model, Data Types in XML	T1:251-274, T1:293-303
5.	1	• **Recapitulation and discussion of Important questions**	
6.	1	• **Discussion of Previous year Question Papers**	
7.	1	• **Discussion of Previous year Question Papers**	
8.	1	• **Discussion of Previous year Question Papers**	
		Total No of Hours Planned For Unit – V	**8**
		Total No. of Hours Planned: 30	

Text Book:

T1→William, J. Pardi. XML in action web technology.
T2→ Michael, J. Young. Step by Step XML.

Reference Book:

R1→ Robert Standefer, Enterprise XML, Clearly explained

R2→Heather Williamson, XML: The Complete Reference, Tata McGraw-Hill Publishing Company Limited, New Delhi.

WEBSITES

W1→https://en.wikipedia.org/wiki/Markup_language

W2→ https://www.w3schools.com/xml/xml_whatis.asp

W3→https://www.ibm.com/developerworks/xml/tutorials/xmlintro/xmlintro.html

W4→http://www.edankert.com/overview/introduction.design.html

W5→http://www.xmlfiles.com/xml/xml_syntax.asp

W6→https://msdn.microsoft.com/en-us/library/system.xml.xmldocument(v=vs.110).aspx

W7→https://www.cs.uct.ac.za/mit_notes/web_programming/html/ch09s07.html

W8→http://help.madcapsoftware.com/flare2017r2/Content/Scripts/Flare/Editing-Scripts.htm

KAHE

UNIT – 1

SYLLABUS

Introduction: Understanding Mark-up Languages, Introduction to XML and its Goals.

UNDERSTANDING MARKUP LANGUAGES

What are Markup Languages?

There are many different markup languages out there in the world. For web design and development, there are three specific markup languages that you will likely run across. These are HTML, XML, and XHTML.

WHAT IS A MARKUP LANGUAGE?

To properly define this term - a markup language is a language that annotates text so that the computer can manipulate that text. Most markup languages are human readable because the annotations are written in a way to distinguish them from the text itself. For example, with HTML, XML, and XHTML, the markup tags are < and >. Any text that appears within one of those characters is considered part of the markup language and not part of the annotated text.

For example:

* *
This is a paragraph of text written in HTML
* *

This example is an HTML paragraph. It is made up of an opening tag ( ), a closing tag ( ), and the actually text that would be displayed on screen (this is the text contained between the two tags). Each tag includes a "less than" and "great than" symbol to designate it as part of the markup.

When you format text to be displayed on a computer or other device screen, you need to distinguish between the texts itself and the instructions for the text. The "markup" is the instructions for displaying or printing the text.

Markup doesn't have to be computer readable. Annotations done in print or in a book are also considered markup. For example, many students in school will highlight certain phrases in their

text books. This indicates that the highlighted text is more important than the surrounding text. The highlight color is considered markup.

HTML—HYPERTEXT MARKUP LANGUAGE

HTML or Hyper Text Markup Language is the primary language of the Web and the most common one you will work with as a web designer/developer.

In fact, it may be the only markup language you use in your work.

All web pages are written in a flavor of HTML. HTML defines the way that images, multimedia, and text are displayed in web browsers. This language includes elements to connect your documents (hypertext) and make your web documents interactive (such as with forms). Many people call HTML "website code", but in truth it is really just a markup language. Neither term is strictly wrong nor will you hear people, including web professionals, use these two terms interchangeably.

HTML is a defined standard markup language. It is based upon SGML (Standard Generalized Markup Language). It is a language that uses tags to define the structure of your text. Elements and tags are defined by the < and > characters.

While HTML is by far the most popular markup language used on the Web today, it is not the only choice for web development.

As HTML was developed, it got more and more complicated and the style and content tags combined into one language. Eventually, the W3C decided that there was a need for a separation between the style of a web page and the content. A tag that defines the content alone would remain in HTML while tags that define style were deprecated in favor of CSS (Cascading Style Sheets).

The newest numbered version of HTML is HTML5. This version added more features into HTML and removed some of the strictness that was imposed by XHTML (more on that language shortly).

XML—EXTENSIBLE MARKUP LANGUAGE

The eXtensible Markup Language is the language that another version of HTML is based on. Like HTML, XML is also based off of SGML. It is less strict than SGML and stricter than plain HTML. XML provides the extensibility to create various different languages.

XML is a language for writing markup languages. For example, if you are working on genealogy, you might create tags using XML to define the father, mother, daughter, and son in your XML like this: <father><mother><daughter><son>. There are also several standardized

languages already created with XML: MathML for defining mathematics, SMIL for working with multimedia, XHTML, and many others.

XHTML—EXTENDED HYPERTEXT MARKUP LANGUAGE

XHTML 1.0 is HTML 4.0 redefined to meet the XML standard.

XHTML has been replaced in modern web design with HTML5 and the changes that have come since. You are unlikely to find any newer sites using XHTML, but if you are working on a much older site, you may still encounter XHTML out there in the wild.

There aren't a lot of major differences between HTML and XHTML, but here is what you will notice:

- XHTML is written in lower case. While HTML tags can be written in UPPER case, MiXeD case, or lower case, to be correct, XHTML tags must be all lower case. (Note - many web professionals write HTML in all lowercase, even though it is not technical required).
- All XHTML elements must have an end tag. Elements with only one tag, such as and need a closing slash (/) at the end of the tag:

<hr />
<img />

- All attributes must be quoted in XHTML. Some people remove the quotes around attributes to save space, but they are required for correct XHTML.
- XHTML requires that tags are nested correctly. If you open a bold () element and then an italics () element, you must close the italics element () before you close the bold (). (Note that both of these elements have been deprecated because they are visual elements. HTML now uses and in place of these two)
- HTML Attributes must have a name and a value. Attributes that are stand-alone in HTML must be declared with values as well, for example, the HR attribute would be written noshade="noshade".

Introduction to XML:

XML is a software- and hardware-independent tool for storing and transporting data.

What is XML?

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data

- XML was designed to be self-descriptive
- XML is a W3C Recommendation

Example:

XML Does Not DO Anything, Maybe it is a little hard to understand, but XML does not DO anything.

This note is a note to raja from ram, stored as XML:

```
<note>
  <to>Raja</to>
  <from>Ram</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The XML above is quite self-descriptive:

- It has sender information.
- It has receiver information
- It has a heading
- It has a message body.

But still, the XML above does not DO anything. XML is just information wrapped in tags.

Someone must write a piece of software to send, receive, store, or display it:

Note

To: Raja

From: Ram

Reminder

Don't forget me this weekend!

The Difference between XML and HTML:

XML and HTML were designed with different goals:

No.	HTML	XML
1)	HTML is used **to display data** and focuses on how data looks.	XML is a software and hardware independent tool used**to transport and store data**. It focuses on what data is.
2)	HTML is a **markup language**itself.	XML provides a **framework to define markup languages**.
3)	HTML is **not case sensitive**.	XML is **case sensitive**.
4)	HTML is a presentation language.	XML is neither a presentation language nor a programming language.
5)	HTML **has its own predefined tags**.	You **can define tags according to your need**.
6)	In HTML, it is **not necessary to use a closing tag**.	XML **makes it mandatory to use a closing tag**.
7)	HTML is **static** because it is used to display data.	XML is **dynamic** because it is used to transport data.
8)	HTML **does not preserve whitespaces**.	XML **preserve whitespaces**.

XML Does Not Use Predefined Tags

The XML language has no predefined tags.

The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.

HTML works with predefined tags like <p>, <h1>, <table>, etc.

With XML, the author must define both the tags and the document structure.

XML is Extensible

Most XML applications will work as expected even if new data is added (or removed).

Imagine an application designed to display the original version of note.xml (<to><from><heading><data>).

Then imagine a newer version of note.xml with added <date> and <hour> elements, and a removed <heading>.

The way XML is constructed; older version of the application can still work:

```
<note>
  <date>2015-09-01</date>
  <hour>08:30</hour>
  <to>Raja</to>
  <from>Ram</from>
  <body>Don't forget me this weekend!</body>
</note>
```

Old Version

Note

To: Raja

From: Ram

Head: (none)

Don't forget me this weekend!

New Version

Note

To: Raja

From: Ram

Date: 2015-09-01 08:30

Don't forget me this weekend!

Design goals for XML

- XML shall be straightforwardly usable over the Internet.
- XML shall support a wide variety of applications.
- XML shall be compatible with SGML.
- It shall be easy to write programs which process XML documents.
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- XML documents should be human-legible and reasonably clear.
- The XML design should be prepared quickly.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.
- Terseness in XML markup is of minimal importance.

A Family of Standards and Recommendations:

XML is not a single standard. The core XML 1.0 syntactic definition is enhanced by the following standards that are either complete or under development:

Recommendations

- XML 1.0: February 1998 (Revised October 2000)
- XML Namespaces: January 1999
- XSLT: November 1999 (part of XSL)
- XPath: November 1999 (used by XSLT and XPointer)
- XHTML (HTML redefined as XML): January 2000
- XHTML Basic: December 2000 (Base subset)
- Canonical XML: March 2001
- XML Schemas Parts 1 and 2: May 2001
- XLink and XBase: June 2001
- XSL: October 2001
- XML Information Set: October 2001
- XML-Signature: February 2002
-

Proposed Recommendations

- Exclusive XML Canonicalization: May 2002
-

Candidate Recommendations

- XML Fragment Interchange: February 2001
- XPointer: September 2001
- XInclude: February 2002
- XML Signature Decryption: March 2002
- XML Encryption: March 2002

Working Drafts

- XML Events: October 2001
- XML Protocol: December 2001
- XForms 1.0: January 2002
- XML Key Management: March 2002
- XKMS: March 2002
- XPath 2.0: April 2002
- XQuery: Arpil 2002
- XML 1.1: April 2002

Important XML Standards

This tutorial will also dig deep into the following important XML standards:

- XML AJAX
- XML DOM
- XML XPath
- XML XSLT
- XML XQuery
- XML DTD
- XML Schema
- XML Services

Following is a partial list of related XML-related areas being worked on at W3C:

- Associating Style Sheets with XML documents Version 1.0

Recommendation -- W3C [June 29, 1999]

Standardized syntax for using an xml- stylesheet processing instruction to associate an XML document with an XSL or CSS stylesheet.

- Authoring Tool Accessibility Guidelines Version 1.0

Recommendation -- W3C [Feb. 3, 2000] Guidelines for web authoring tool developers to assist in the design of authoring tools that produce accessible Web content and in the creation of an accessible authoring interface.

- CSS1 Recommendation (Revised) -- W3C [Jan. 11, 1999] CSS1 is a simple style sheet mechanism that allows authors and readers to attach style (e.g. fonts, colors and spacing) to HTML documents.
- CSS2 Recommendation -- W3C [May 12, 1998] Level 2 of the Cascading Style Sheet mechanism
- DOM Level 1 Recommendation -- W3C [Oct. 1, 1998] Document Object Model Level 1
- Namespaces in XML Recommendation -- W3C [Jan. 14, 1999] Provide a simple method for qualifying element and attribute names used in XML documents. This is accomplished by associating a particular tag set by associating a prefix with a URI reference.
- PICS Recommendation -- W3C [May 27, 1998] Standard format for making digitally-signed, machine-readable assertions about a particular information resource
- PICS Rules Recommendation -- W3C [Dec. 29, 1997] · RDF Model and Syntax Recommendation -- W3C [Feb. 22, 1999] A foundation for processing metadata; it provides interoperability between applications that exchange machine-understandable information on the Web.
- XHTML 1.0 Recommendation -- W3C [Jan. 26, 2000] A reformulation of HTML 4 as an XML 1.0 application and three DTDs corresponding to the ones defined by HTML 4.
- XML Path Language (XPath) Version 1.0 Recommendation -- W3C [Nov. 16, 1999] Provide a common syntax and semantics for querying and addressing the contents of XML documents that could **be** used by XSLT (XSL Transformation Language), XLink, and XPointer.

· XML v. 1.0 DTD Revised XML Recommendation DTD -- W3C [Sept. 10, 1998] A revised version of the XML Recommendation's DTD.

· XML XPointer Requirements W3C [Feb. 24, 1999] · XSL Transformations (XSLT) Specification Version 1.0 Recommendation -- W3C [Nov. 16, 1999] A language used to "transform" (or reconstruct the structure of) the data structures contained within XML documents.

· XInclude Inclusion Proposal (proposed for XML version 2.0) Fragments of content from external resources are included in a documents content along with the content residing at the actual URL being accessed (sort of like a server-side include).

POSSIBLE QUESTIONS

2 MARKS

1. What is XML?
2. Differentiate XML & HTML?
3. Write a note on SGML & HTML.
4. What are the Recommendations are in XML? Mention its Name.
5. List important XML Standards.
6. Does XML use Predefined tags?

8 MARKS

1. Explain in detail about history and Relationships between XML, HTML, and SGML.
2. Write about Design Goals of XML.
3. Write detailed note on: A Family of Standards and Recommendations
4. Enlighten XML-related areas being worked on at W3C.
5. Write about the History of SGML and HTML.
6. Write the history of World Wide Web.
7. Explain the features and components of XML with Illustration.
8. Write an account on WWW and W3C

KARPAGAM ACADEMY OF HIGHER EDUCATION

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

II B.Sc CS (Batch 2017-2020)

XML PROGRAMMING

PART - A OBJECTIVE TYPE/MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS

UNIT-I

Sno	Questions	Opt1	Opt2	Opt3	Opt4	Answer
1	XML stands for	Extended Markup language	Extensible Markup Language	Extended Meaningful language	X-Mark Language	Extensible Markup Language
2	SGML stands for	Standard Geographic Marking Language	Simulated General Marking Language	Standard Generalized Markup Language	Simple& General Markup Language	Standard Generalized Markup Language
3	CSS stands for	Cascading Style Sheet	Common Standard style	Common Style Sheet	Cascade Structured Stylesheet	Cascading Style Sheet
4	DTD is an abbreviation of _____	Document Typing Device	Digital Typing Device	Document Type Definition	Direct Type Definition	Document Type Definition
5	DOM is an abbreviation of _____	Document Object Model	Document Of markup	Defining Object Model	Digital Object Model	Document Object Model
6	Document Style Semantics and Specification Language is abbreviated as	DSL	DSSL	DSSSL	DSSSSL	DSSSL
7	Hypermedia/ Time-based Structuring Language is abbreciated as	HTSL	HTL	HyTime	HMTSL	HyTime

8	_____ is a standards Organization	XML	Deitel Associations Inc.	ARPA	World Wide Web Consortium	World Wide Web Consortium
9	Which of these is not a W3C recommendation	XML	HTML	CSS	WWW	WWW
10	The technologies standardized by W3C are called _____	Standards	Recommendations	Languages	Markups	Recommendations
11	_____ are not actual software products, but documents that specifythe role, syntax, rules etc.,of a technology.	Standards	Recommendations	Languages	Markups	Recommendations
12	W3C Recommendations pass through _____ phases	2	4	5	3	3
13	_____ specifies an evolving phase of W3C Recommendatio	Working Draft	Candidate Recommendation	Proposed Recommendation	Languages	Working Draft
14	A stable version of the document - implemented by the industry is called _____	Working Draft	Candidate Recommendation	Proposed Recommendation	Languages	Candidate Recommendation
15	A Candidate Recommendation that is considered Mature is called _____	Working Draft	Candidate Recommendation	Proposed Recommendation	Languages	Proposed Recommendation
16	A Blueprint of networking a dozen of computer systems was proposed in a conference sponsered by	ARPA	W3C	MIT	INRIA	ARPA
17	The first network of computers is called	World wide web	Internet	ARPAnet	WebWorld	ARPAnet
18	The ARPAnet communication line operated at a rate of _____	100 bps	110 kbps	56 kbps	80 bps	56 kbps
19	Connecting of telephone lines to computers operated at rate of	100 bps	110 bps	56 kbps	80 bps	110 bps
20	_____ is the grandparent of today's internet	World wide web	Telephone network	ARPAnet	WebWorld	ARPAnet
21	The _____ on the header of each packet is used to find the destination to which the packet is sent	packet information	address information	sequence information	data information	address information

22	The ARPA network operated with a technique called	broadcasting	telephoning	data exchanging	packet switching	packet switching
23	Small package of digital data transmitted through the net is called _____	packets	data	nodes	files	packets
24	_____ is used to reassembling the received packets	packet information	address information	sequence information	data informatio n	sequence information
25	The Protocols for communicating over the ARPAnet was known as _____	Transmission protocol	Transaction control Program	Transmission Connection Protocol	Transmissi on Control Protocol	Transmissio n Control Protocol
26	TCP stands for _____	Transmission protocol	Transaction control Program	Transmission Connection Protocol	Transmissi on Control Protocol	Transmissio n Control Protocol
27	The informaion carrying capacity of communication lines is called _____	bandwidth	communicatio n protocol	information rate	net rate	bandwidth
28	_____ is the founder of W3C	Tim Berners Lee	Charles GoldFarb	Edward Mosher	Raymond Lorie	Tim Berners Lee
29	The Technology of sharing information using hyperlinked text document was developed by _____	Tim Berners Lee	Charles GoldFarb	Edward Mosher	Raymond Lorie	Tim Berners Lee
30	The formatting information kept in separate files called _____	XML Documents	Format Files	Style Sheets	Structuring Sheets	Style Sheets
31	The documents with missing or extra information are considered	incomplete documents	non-structured documents	invalid documents	unformed documents	invalid documents
32	The structure of each document is strictly defined in a file called	DOM	DTD	CSS	XSL	DTD
33	The team of researchers Charles GoldFarb, Edward Mosher and Raymond Lorie developed a language called _____	XML	SGML	GML	XSL	GML
34	The software capable of analyzing the structure and syntax of a document is called _____	compiler	translator	analyser	Parser	Parser

35	In 1974 Goldfarb proved that a _____ can validate a document without actually processing it.	compiler	translator	analyser	Parser	Parser
36	The first parser was developed by	Tim Berners Lee	Charles GoldFarb	Edward Mosher	Raymond Lorie	Charles GoldFarb
37	_____ is a subset of SGML	XML	HTML	XSL	none of the above	XML
38	_____ is an application of SGML	XML	HTML	XSL	none of the above	HTML
39	XML is a _____	Programming languge	Formatting language	Meta language	Processing Language	Meta language
40	TEI stands for	Text Enhancing Interface	Text Encoding Initiative	Transmission Enhancing Information	none of the above	Text Encoding Initiative
41	_____ incorporates the elements of both CSS and DSSSL	XSL	XML	DSL	Xlink	XSL
42	XSL stands for	Extensible standard Languae	Extensible Stylesheet Language	Extended Simple Langage	Extented Standard Language	Extensible Stylesheet Language
43	____ combines the ideas from Hytime and TEI	XSL	XML	DSL	XLink	XLink
44	XLink in an acronym of	Extensible standard Languae	Extensible Stylesheet Language	Extended Library Langage	Extensible Linking Language	Extensible Linking Language
45	Who is known as the father of Internet	Tim Berners Lee	Charles GoldFarb	Edward Mosher	Raymond Lorie	Tim Berners Lee

UNIT – II

SYLLABUS

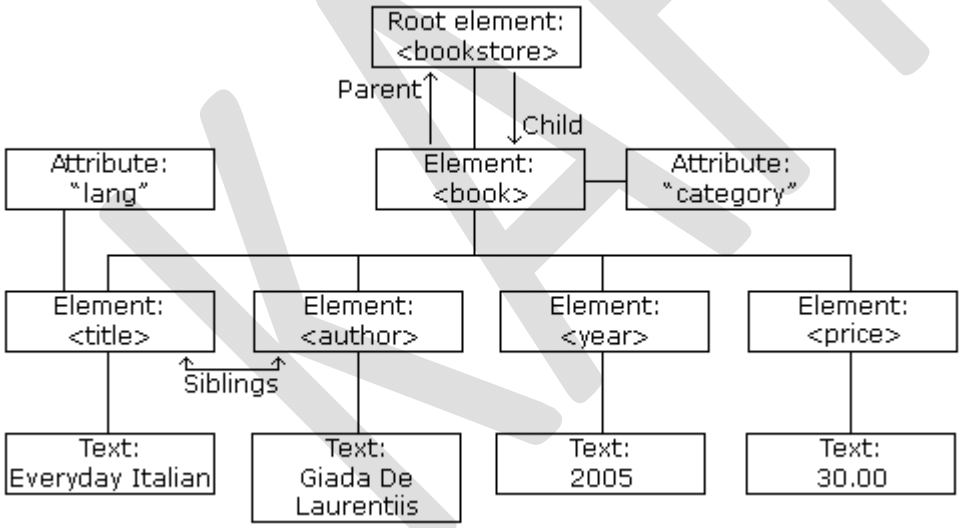
XML Basics: XML Structure and Syntax, Document classes and Rules.

XML Basics:

XML Tree

XML documents form a tree structure that starts at "the root" and branches to "the leaves".

XML Tree Structure



An Example XML Document

The image above represents books in this XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
```

```
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
 </book>
 <book category="children">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
 </book>
 <book category="web">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
 </book>
</bookstore>
```

XML Tree Structure

XML documents are formed as element trees.

An XML tree starts at a root element and branches from the root to child elements.

All elements can have sub elements (child elements):

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

The terms parent, child, and sibling are used to describe the relationships between elements.

Parents have children. Children have parents. Siblings are children on the same level (brothers and sisters).

All elements can have text content (Harry Potter) and attributes (category="cooking").

Self-Describing Syntax

XML uses a much self-describing syntax.

A prolog defines the XML version and the character encoding:

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

The next line is the root element of the document:

<bookstore>

The next line starts a <book> element:

<book category="cooking">

The <book> elements have 4 child elements: <title>, <author>, <year>, <price>.

<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>

The next line ends the book element:

</book>

You can assume, from this example, that the XML document contains information about books in a bookstore.

XML Syntax Rules

The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use.

XML Documents Must Have a Root Element

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

<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>

In this example **<note>** is the root element:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The XML Prolog

This line is called the XML **prolog**:

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

The XML prolog is optional. If it exists, it must come first in the document.

XML documents can contain international characters, like Norwegian øæå or French êèé.

To avoid errors, you should specify the encoding used, or save your XML files as UTF-8.

UTF-8 is the default character encoding for XML documents.

UTF-8 is also the default encoding for HTML5, CSS, JavaScript, PHP, and SQL.

All XML Elements Must Have a Closing Tag

In HTML, some elements might work well, even with a missing closing tag:

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

In XML, it is illegal to omit the closing tag. All elements **must** have a closing tag:

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

The XML prolog does not have a closing tag.
This is not an error. The prolog is not a part of the XML document.

XML Tags are Case Sensitive

XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.

Opening and closing tags must be written with the same case:

<Message>This is incorrect</message>
<message>This is correct</message>

"Opening and closing tags" are often referred to as "Start and end tags". Use whatever you prefer. It is exactly the same thing.

XML Elements Must be Properly Nested

In HTML, you might see improperly nested elements:

This text is bold and italic

In XML, all elements **must** be properly nested within each other:

This text is bold and italic

In the example above, "Properly nested" simply means that since the element is opened inside the element, it must be closed inside the 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.

INCORRECT:

```
<note date=12/11/2007>
  <to>Tove</to>
  <from>Jani</from>
</note>
```

CORRECT:

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

The error in the first document is that the date attribute in the note element is not quoted.

Entity References

Some characters have a special meaning in XML.

If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

This will generate an XML error:

<message>salary < 1000</message>

To avoid this error, replace the "<" character with an **entity reference**:

<message>salary < 1000</message>

There are 5 pre-defined entity references in XML:

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

Only < and & are strictly illegal in XML, but it is a good habit to replace > with > as well.

Comments in XML

The syntax for writing comments in XML is similar to that of HTML.

<!-- This is a comment -->

Two dashes in the middle of a comment are not allowed.

Not allowed:

<!-- This is a -- comment -->

Strange, but allowed:

<!-- This is a - - comment -->

White-space is preserved in XML

XML does not truncate multiple white-spaces (HTML truncates multiple white-spaces to one single white-space):

XML:	Hello Tove
HTML:	Hello Tove

XML Stores New Line as LF

Windows applications store a new line as: carriage return and line feed (CR+LF).

Unix and Mac OSX uses LF.

Old Mac systems uses CR.

XML stores a new line as LF.

XML Elements

An XML document contains XML Elements.

What is an XML Element?

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

<price>29.99</price>

An element can contain:

- text
- attributes
- other elements
- or a mix of the above

```
<bookstore>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

In the example above:

<title>, <author>, <year>, and <price> have **text content** because they contain text (like 29.99).

<bookstore> and <book> have **element contents**, because they contain elements.

<book> has an **attribute** (category="children").

Empty XML Elements

An element with no content is said to be empty.

In XML, you can indicate an empty element like this:

<element></element>

You can also use a so called self-closing tag:

<element />

The two forms produce identical results in XML software (Readers, Parsers, Browsers).

Empty elements can have attributes.

XML Naming Rules

XML elements must follow these naming rules:

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces

Any name can be used; no words are reserved (except xml).

Best Naming Practices

Create descriptive names, like this: <person>, <firstname>, <lastname>.

Create short and simple names, like this: <book_title> not like this: <the_title_of_the_book>.

Avoid "-". If you name something "first-name", some software may think you want to subtract "name" from "first".

Avoid ".". If you name something "first.name", some software may think that "name" is a property of the object "first".

Avoid ":". Colons are reserved for namespaces (more later).

Non-English letters like éòá are perfectly legal in XML, but watch out for problems if your software doesn't support them.

Naming Styles

There are no naming styles defined for XML elements. But here are some commonly used:

Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Underscore	<first_name>	Underscore separates words
Pascal case	<FirstName>	Uppercase first letter in each word
Camel case	<firstName>	Uppercase first letter in each word except the first

If you choose a naming style, it is good to be consistent!

XML documents often have a corresponding database. A common practice is to use the naming rules of the database for the XML elements.

Camel case is a common naming rule in JavaScripts.

XML Elements are Extensible

XML elements can be extended to carry more information.

Look at the following XML example:

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <body>Don't forget me this weekend!</body>
</note>
```

Let's imagine that we created an application that extracted the <to>, <from>, and <body> elements from the XML document to produce this output:

MESSAGE

To: Tove
From: Jani

Don't forget me this weekend!

Imagine that the author of the XML document added some extra information to it:

```
<note>
  <date>2008-01-10</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Should the application break or crash?

No. The application should still be able to find the <to>, <from>, and <body> elements in the XML document and produce the same output.

This is one of the beauties of XML. It can be extended without breaking applications.

XML Attributes

XML elements can have attributes, just like HTML.

Attributes are designed to contain data related to a specific element.

XML Attributes must be quoted

Attribute values must always be quoted. Either single or double quotes can be used.

For a person's gender, the <person> element can be written like this:

<person gender="female">

or like this:

<person gender='female'>

If the attribute value itself contains double quotes you can use single quotes, like in this example:

<gangster name='George "Shotgun" Ziegler'>

or you can use character entities:

<gangster name="George "Shotgun" Ziegler">

XML Elements vs. Attributes

Take a look at these examples:

```
<person gender="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>

<person>
  <gender>female</gender>
```

```
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

In the first example gender is an attribute. In the last, gender is an element. Both examples provide the same information.

There are no rules about when to use attributes or when to use elements in XML.

My Favorite Way

The following three XML documents contain exactly the same information:

A date attribute is used in the first example:

```
<note date="2008-01-10">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

A <date> element is used in the second example:

```
<note>
  <date>2008-01-10</date>
  <to>Tove</to>
  <from>Jani</from>
</note>
```

An expanded <date> element is used in the third example: (THIS IS MY FAVORITE):

```
<note>
  <date>
    <year>2008</year>
    <month>01</month>
    <day>10</day>
  </date>
  <to>Tove</to>
  <from>Jani</from>
</note>
```

Avoid XML Attributes?

Some things to consider when using attributes are:

- attributes cannot contain multiple values (elements can)
- attributes cannot contain tree structures (elements can)
- attributes are not easily expandable (for future changes)

Don't end up like this:

```
<note day="10" month="01" year="2008"
to="Tove" from="Jani" heading="Reminder"
body="Don't forget me this weekend!">
</note>
```

XML Attributes for Metadata

Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements in much the same way as the id attribute in HTML. This example demonstrates this:

```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Reminder</heading>
    <body>I will not</body>
  </note>
</messages>
```

The id attributes above are for identifying the different notes. It is not a part of the note itself.

What I'm trying to say here is that metadata (data about data) should be stored as attributes, and the data itself should be stored as elements.

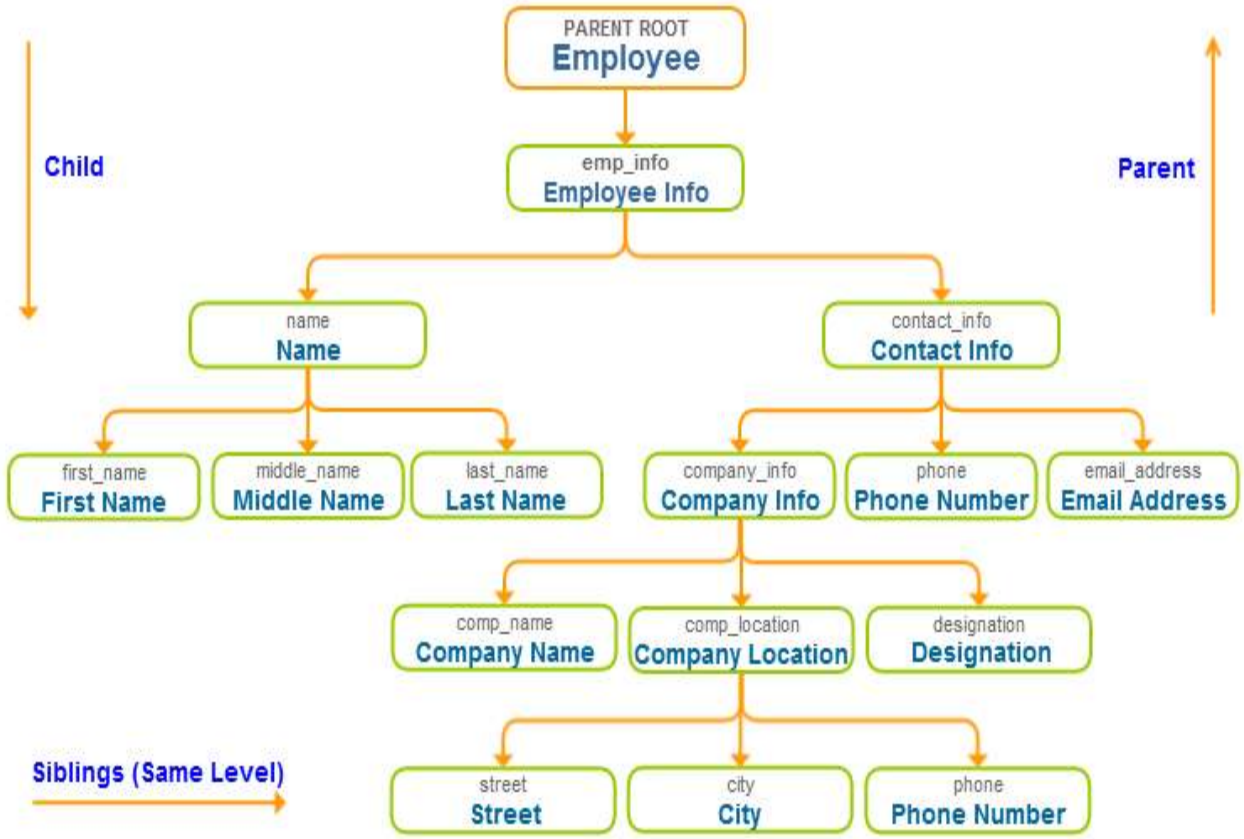
Describing Structures in XML

XML structure start to a **Parent root** from top of the side. Every XML documents have only **one root element**.

XML was describing a tree structure of data. And tree structures have one root, child elements, branches, attributes, values. Following are simple XML structures.

```
<root>
  <element>
    <subelement>...</subelement>
  </element>
</root>
```

So upto now you know XML have ability to specify new tags and also create a nested tags to make a strict valid tree structure for exchanging data.



Describing Structures in XML

Above visual tree structure assume our example base on this structure make one XML document including all that describe information.

```
<employee>
 <emp_info id="1">
  <name>
        <first_name>Opal</first_name>
        <middle_name>Venue</middle_name>
        <last_name>Kole</last_name>
      </name>
  <contact_info>
        <company_info>
            <comp_name>Odoo (formally OpenERP)</comp_name>
            <comp_location>
                <street>Tower-1, Infocity</street>
                <city>GH</city>
                <phone>000-478-1414</phone>
            </comp_location>
            <designation>Junior Engineer</designation>
        </company_info>
            <phone>000-987-4745</phone>
            <email>email@myemail.com</email>
```

```
        </contact_info>
  </emp_info>
</employee>
```

Structure Rules Explanation

Tag Name: label for a section of data. Label must be relative to section data.

Section Element: Section of data beginning with <tagname> and ending with </tagname>. Example. <name> ... </name>

Nested: Every element must be properly nested. Example: <name> <fist_name> ... </first_name> </name>

Attribute you can specify attribute *name=value* inside the starting tag of the element. You can specify several attributes.

Note: Attributes specify for identify the elements.

Well-Formed vs Valid

The terms **"well-formed"** and **"valid"** are important for markup languages, in particular for XML documents and applications of XML such as XHTML. There is an important difference between the two terms however, and this should be understood by developers.

Well-Formed Document

A **well-formed** XML document complies with the syntactic rules of XML markup. These rules are strict but basically quite simple. The XML syntax rules include

- documents must be self-describing (they start with an XML declaration)
- a document must contain one or more elements
- documents must contain a single root element
- start and end tags must be used to identify elements (must have closing tags)
- empty elements must be marked as such (with a self-closing />)
- attribute values must be quoted (single or double quotes are fine)
- Element names and attributes names are case sensitive.
- element tags must be correctly nested (must not overlap)
- Element names (attribute names) must start with a letter.

Other interesting things about XML documents

- Whitespace is **preserved** (unlike the way that whitespace is ignored by web browsers in HTML documents).
- Newlines are only stored as a single newline character (carriage-return characters are removed).
- HTML style comments <!-- ... --> (well, SGML actually) work fine as well.

A well-formed document is a good place to **start**, but it does not mean that the document will make any sense, or that its content makes any sense. It means that the formatting rules have been followed, and this ensures that other uses of the XML content can take place.

The concept of "well-formed" is a bit like an English sentence that is punctuated correctly; start with capital letters, a single space between works, maybe some commas, and a period at the end. However, it says nothing about the correct spelling of the words, or the grammatical sense or order of the words.

Valid Document

A **valid** XML document means that the document complies with the rules of a particular document specification. The specification is commonly done in an external DTD file, however it might also be a DTD section of the XML file, or an XML based schema file (which also allows other features).

Before a document can be valid to a standard, it must start by being well-formed. In fact, it's the first requirement for a valid document: it must be well-formed.

Perhaps the most common examples of document specifications are for XHTML documents. There are three separate document types for XHTML 1.0, Strict, Transitional and Frameset.

The XML specification requires that XML documents (such as XHTML 1.0 Strict) include an XML declaration at the beginning (self-identifying themselves as XML documents), it is common practice **not** to do this for XHTML documents - most newer web browsers understand this correctly, but older browsers such as **Internet Explorer 6** do not understand this and as a result have been known to render the HTML content incorrectly.

XML DTD

What is DTD

DTD stands for **Document Type Definition**. It defines the legal building blocks of an XML document. It is used to define document structure with a list of legal elements and attributes.

Purpose of DTD

Its main purpose is to define the structure of an XML document. It contains a list of legal elements and defines the structure with the help of them.

Checking Validation

Before proceeding with XML DTD, you must check the validation. An XML document is called "well-formed" if it contains the correct syntax.

A well-formed and valid XML document is one which has been validated against DTD.

Valid and well-formed XML document with DTD

Let's take an example of well-formed and valid XML document. It follows all the rules of DTD.

employee.xml

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

In the above example, the DOCTYPE declaration refers to an external DTD file. The content of the file is shown in below paragraph.

employee.dtd

```
1. <!ELEMENT employee (firstname,lastname,email)>
2. <!ELEMENT firstname (#PCDATA)>
3. <!ELEMENT lastname (#PCDATA)>
4. <!ELEMENT email (#PCDATA)>
```

Description of DTD

<!DOCTYPE employee : It defines that the root element of the document is employee.

<!ELEMENT employee: It defines that the employee element contains 3 elements "firstname, lastname and email".

<!ELEMENT firstname: It defines that the firstname element is #PCDATA typed. (parse-able data type).

<!ELEMENT lastname: It defines that the lastname element is #PCDATA typed. (parse-able data type).

<!ELEMENT email: It defines that the email element is #PCDATA typed. (parse-able data type).

XML DTD with entity declaration

A doctype declaration can also define special strings that can be used in the XML file.

An entity has three parts:

1. An ampersand (&)
2. An entity name
3. A semicolon (;)

Syntax to declare entity:

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

Let's see a code to define the ENTITY in doctype declaration.

author.xml

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE author [
  <!ELEMENT author (#PCDATA)>
  <!ENTITY sj "Sonoo Jaiswal">
]>
<author>&sj;</author>
```

In the above example, sj is an entity that is used inside the author element. In such case, it will print the value of sj entity that is "Sonoo Jaiswal".

Note: A single DTD can be used in many XML files.

CDATA vs PCDATA

CDATA

CDATA: (Unparsed Character data): CDATA contains the text which is not parsed further in an XML document. Tags inside the CDATA text are not treated as markup and entities will not be expanded.

Let's take an example for CDATA:

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

<!DOCTYPE employee SYSTEM "employee.dtd">

<employee>

<![CDATA[

  <firstname>vimal</firstname>

  <lastname>jaiswal</lastname>

  <email>vimal@javatpoint.com</email>

]]>

</employee>
```

In the above CDATA example, CDATA is used just after the element employee to make the data/text unparsed, so it will give the value of employee:

<firstname>vimal</firstname><lastname>jaiswal</lastname><email>vimal@javatpoint.com</email>

PCDATA

PCDATA: (Parsed Character Data): XML parsers are used to parse all the text in an XML document. PCDATA stands for Parsed Character data. PCDATA is the text that will be parsed by a parser. Tags inside the PCDATA will be treated as markup and entities will be expanded.

In other words you can say that a parsed character data means the XML parser examine the data and ensure that it doesn't content entity if it contains that will be replaced.

Let's take an example:

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

```
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

In the above example, the employee element contains 3 more elements 'firstname', 'lastname', and 'email', so it parses further to get the data/text of firstname, lastname and email to give the value of employee as:

vimal jaiswal vimal@javatpoint.com

KAHE

POSSIBLE QUESTIONS

2 MARKS

1. Write XML Syntax and structure with example?
2. Define Valid VS Well-Formed XML?
3. Write a short note on DTD with example.
4. What is CDATA?
5. What is PCDATA?
6. What is the purpose of DTD?
7. What is XML Attributes? Give Example.
8. What are all the XML naming rules?
9. What is an XML element? Give Example.
10. Define Entity Reference in XML?
11. How to define Comments in XML? Give Example.
12. What is an XML tree define its Structure?

8 MARKS

1. Explain in detail about XML tree with example.
2. Give explanation about DTD with example.
3. Elucidate XML syntax rules with example.
4. Enlighten CDATA Vs PCDATA.
5. Explain in detail about Valid VS Well-Formed documents XML.

KARPAGAM ACADEMY OF HIGHER EDUCATION

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

II B.Sc CS (Batch 2017-2020)

XML PROGRAMMING

PART - A OBJECTIVE TYPE/MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS

UNIT-II

Sno	Questions	Opt1	Opt2	Opt3	Opt4	Answer
1	XML files are stored in _____ format	image	text	binary	ASCII	text
2	XML files names commonly end with the extension _____	xsl	txt	bmp	xml	xml
3	Version of the XML used is mentioned in _____	comment statement	root element	xml declaration tag	any user defined tag	xml declaration tag
4	If the version of XML is not specified it is assumed to conform to _____ version.	latest	1.0	1.2	2.0	latest
5	All XML documents must contain _____ root elements	any number of	exactly one	optionally one	two	exactly one
6	Which of the following is an error in XML	nesting of elements	not declaring the version of XML	Xml files not ending with.xml extension	creating more than one root element	creating more than one root element
7	XML tags are _____	ignores case	case sensitive	reserved words	all uppercase only	case sensitive
8	XML document are _____ if it is syntactically correct	valid	invalid	well-formed	parsed	well-formed
9	_____ is required to parse the XML Document	XML-Parser	DTD file	Schema	XSL	XML-Parser
10	_____ parser is inbuild within IE5	sax	Xerces	JAPX	msxml	msxml
11	JAXP stands for	Java API For XML Parsing	Java and Xml Programming	Java Based XML Application Program	SAX	Java API For XML Parsing

12	SAX stands for	Stylesheet Application for XML	Simple API for XML	Schema API for XML	SGML And XML Programming	Simple API for XML
13	_____ consists of characters that may be represented in a document	ASCII	Markup text	Character Set	CDATA	Character Set
14	XML documents may contain _____characters	ASCII	Unicode	BCD	EBCDIC	Unicode
15	_____ consists fo characters of most worlds major languages.	ASCII	Unicode	BCD	EBCDIC	Unicode
16	Space, tabs, line feeds and carriage returns are commonly called	CDATA	PCDATA	White Space characters	Build-in Entities	White Space characters
17	Insignificant Whitespace characters are collapsed into a single whitespace character. This is called ___	trimming	collapsing	parsing	normalising	normalising
18	< > " these are called	Markup text	CDATA Section	Built -in entities	White Space Characrers	Built -in entities
19	All Element names are enclosed within _____	< >	( )	{ }	[ ]	< >
20	Each _____contains a starting tag and an ending tag	Element	Attribute	Entity	character	Element
21	If an element tag starts with < and ends with /> it is _____	root element	parent element	child element	empty element	empty element
22	Processing Instructions are delimited by	< >	<!-- -->	<? ?>	< />	<? ?>
23	___ is a name value pair	Element	Attribute	Entity	Parser	Attribute
24	The values of an attribute is enclosed within	< >	" "	/* */	( )	" "
25	A comment in XML lies within _____	< >	" "	/* */	<!-- -->	<!-- -->
26	_____ may contain text, reserved characters and even multiple whitespace characters	Markup text	CDATA Section	Built -in entities	Text contained in Element	CDATA Section
27	Naming Collisions can be resolved by using _____	Markup text	CDATA Section	Built -in entities	Namespaces	Namespaces
28	The Keyword _____ is used to create namespace prefixes	XML	msxml	xmlns	xns	xmlns
29	Each namespace prefix is tied toa a _____	URI	DTD	XML	DOM	URI
30	Child elements of a default namespace	are prefixed with xml	are prefixed with msxml	are prefixed with xmlns	do not need a prefix	do not need a prefix

31	The set of document type declarations inside an xml document is called the_____	public file	external subset	internal subset	system file	internal subset
32	_____ are the basic building blocks of an xml document	Elements	Attributes	Tags	Text	Elements
33	Elements are declared with the _____ type declaration in the DTD.	ELEMENT	PCDATA	ATTLIST	ENTITY	ELEMENT
34	Keyword _____ indicates that an element contains parsable character data.	CDATA	PCDATA	ATTLIST	ENTITY	PCDATA
35	In an element type decalaration, the pipe character (|) indicates that the element can contain ____ of the elements indicated.	all	any one	each	many	any one
36	Attributes are declared using the _____ type	ELEMENT	PCDATA	ATTLIST	ENTITY	ATTLIST
37	Keywod _____ indicates that the attribute can only take a specific value that has been defined in the DTD.	# IDREF	# REQUIRED	# FIXED	# IMPLIED	# FIXED
38	ID, IDREF, ENTITY,NMTOKEN are all types of _____	attribute defaults	attribute declarations	enumerated attributes	tokenized attributes	tokenized attributes
39	# REQUIRED, # IMPLIED, # FIXED are all	attribute defaults	attribute declarations	enumerated attributes	tokenized attributes	attribute defaults
40	Which of these is not an attribute default	# IDREF	# REQUIRED	# FIXED	# IMPLIED	# IDREF
41	Which of these is not an attribute type	CDATA	Enumerated	Tokenized	PCDATA	PCDATA
42	The % character is used to declare _____	Elements	parameter entity	attribute defaults	enumneratio n	parameter entity
43	Conditional sections of DTDs are often used with ___	elements	entities	attributes	parameters	entities
44	An XML document is said to be _____ if it does not reference an external DTD.	valid	well-formed	standalone	parsable	standalone
45	DTDs can be introduced into XML documents by using _____	DOCTYPE	ATTRTYPE	ELEMENT	PCDATA	DOCTYPE
46	If the DTD is defined outside the XML document it is called _____	public file	external subset	internal subset	system file	external subset
47	External DTDs are referenced using the _____ keyword	EXTERNAL	DOCTYPE	SYSTEM	SUBSET	SYSTEM
48	_____ are defined using Extended Backus-Naur Form (EBNF) grammar	DTD	DOM	Schema	XML	DTD
49	_____ use XML syntax instead of EBNF grammar	DTD	DOM	Schema	XML	Schema

50	The Document Type Declaration is placed in the ___ of the XML	epilogue	prolog	boby	xml tag	prolog
51	The External subset exists in a file that commonly have the extention _____	.xml	.xsl	.dom	.dtd	.dtd
52	The _____of the DTD is visible only with in the document it resides	internal subset	xml-declarations	external subset	CDATA Sections	internal subset
53	The element name that follows ELEMENT keyword in DTD is called the _____	root element	generic identifier	element type indicator	element attributes	generic identifier
54	The element's frequency is specified by using the _____	element type indicator	occurance indicator	generic identifier	attributes elements	occurance indicator
55	_____ specifies that an element must occur atleast once and may occur more that once also (one or more times)	+	*	?	-	+
56	_____ specifies that an element is optional, and if it occurs it may appear only once (0 ot 1 time)	+	*	?	-	?
57	_____ indicates that an element is optional, and if it occurs it may appear any number of times (0 or more)	+	*	?	-	*
58	Schema documents use _____ syntax	EBNF	XML	Java	DTD	XML
59	The collection of DTD's and Schemas for a variety of applications available on the Web are called _____	recommendations	repositories	packages	Application Interfaces	repositories
60	_____ are expected to replace DTD's for describing XML structure.	CSS	XSL	DOM	Schemas	Schemas
61	_____ is the root element of every MS-XML Schema documents	<Xml>	<Root>	<Schema>	ANY	<Schema>
62	In MS-XML Schema _____ defines an element	element	ElementType	ELEMENT	xml:Element	ElementType
63	In MS-XML Schema_____ is used to refer an element defined by ElementType	element	ElementType	ELEMENT	xml:Element	element
64	Element 'Schema' contains 3 elements, they are ____	ElementType, AttributeType and Comments	ElementType, ElementName and Description	ElementType, AttributeType and EntityType	ElementType ,AttributeType and description	ElementType, AttributeType and description

65	AttributeType defines an _____	attribute defaults	attribute	attribute data type	type of the attribute	attribute

UNIT III

SYLLABUS

Other XML Concepts: Scripting XML

SCRIPTING INTRODUCTION:

A **script** or **scripting** language is a **computer** language with a series of commands within a file that is capable of being executed without being compiled.

There are two main ways to customize Web pages and make them more interactive. The two are often used together because they do very different things.

Scripts

A script is a set of instructions. For Web pages they are instructions either to the Web browser (client-side scripting) or to the server (server-side scripting). These are explained more below.

Scripts provide change to a Web page. Think of some Web pages you have visited. Any page which changes each time you visit it (or during a visit) probably uses scripting.

All log on systems, some menus, almost all photograph slideshows and many other pages use scripts. Google uses scripts to fill in your search term for you, to place advertisements, to find the thing you are searching for and so on. Amazon uses scripting to list products and record what you have bought.

The **client-side** environment used to run **scripts** is usually a browser. The processing takes place on the end users computer. The source code is transferred from the web **server** to the user's computer over the internet and run directly in the browser. The **scripting** language needs to be enabled on the **client** computer.

Client-side

The client is the system on which the Web browser is running. JavaScript is the main client-side scripting language for the Web. Client-side scripts are interpreted by the browser. The process with client-side scripting is:

- the user requests a Web page from the server
- the server finds the page and sends it to the user
- the page is displayed on the browser with any scripts running during or after display

So client-side scripting is used to make Web pages change after they arrive at the browser. It is useful for making pages a bit more interesting and user-friendly. It can also provide useful gadgets such as calculators, clocks etc. but on the whole is used for appearance and interaction.

Client-side scripts rely on the user's computer. If that computer is slow they may run slowly. They may not run at all if the browser does not understand the scripting language. As they have to run on the user's system the code which makes up the script is there in the HTML for the user to look at (and copy or change).

Server-side

The server is where the Web page and other content lives. The server sends pages to the user/client on request. The process is:

- the user requests a Web page from the server
- the script in the page is interpreted by the server creating or changing the page content to suit the user and the occasion and/or passing data around
- the page in its final form is sent to the user and then cannot be changed using server-side scripting

The use of HTML forms or clever links allow data to be sent to the server and processed. The results may come back as a second Web page.

Server-side scripting tends to be used for allowing users to have individual accounts and providing data from databases. It allows a level of privacy, personalization and provision of information that is very powerful. E-commerce, MMORPGs and social networking sites all rely heavily on server-side scripting.

PHP and ASP.net are the two main technologies for server-side scripting.

The script is interpreted by the server meaning that it will always work the same way. Server-side scripts are never seen by the user (so they can't copy your code). They run on the server and generate results which are sent to the user. Running all these scripts puts a lot of load onto a server but none on the user's system.

The combination

A site such as Google, Amazon, Facebook or Hobowars will use both types of scripting:

Server-side handles logging in, personal information and preferences and provides the specific data which the user wants (and allows new data to be stored)

Client-side makes the page interactive, displaying or sorting data in different ways if the user asks for that by clicking on elements with event triggers

Good **examples** of server side **scripting** languages include Perl, PHP, and Python. The best example of a client side **scripting** language is JavaScript.

XML Parent-Child Relationship

To implement a parent-child relationship (that is not a subtype relationship) in XML.

Business Rules

1. **Child Entity**

   Each child entitymust be defined as a Core Entity.

   Therefore the rules relating to core entities apply to child entities (i.e. they must be defined as global complex types first.)

2.Parent Entity

Each parent entity must include the child entity by inserting a child entity element as a “reference” to the child entity’s declaration (see examples).

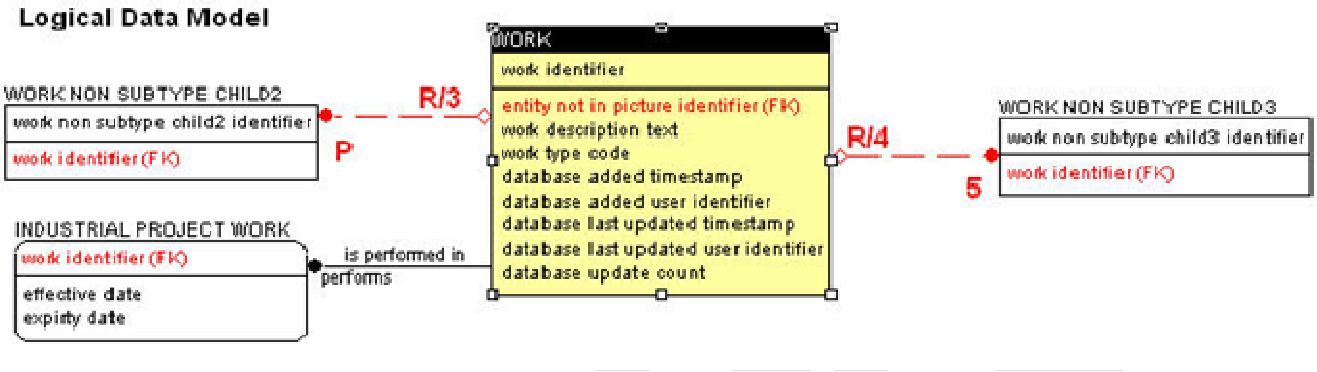
3. Cardinality and Optionality

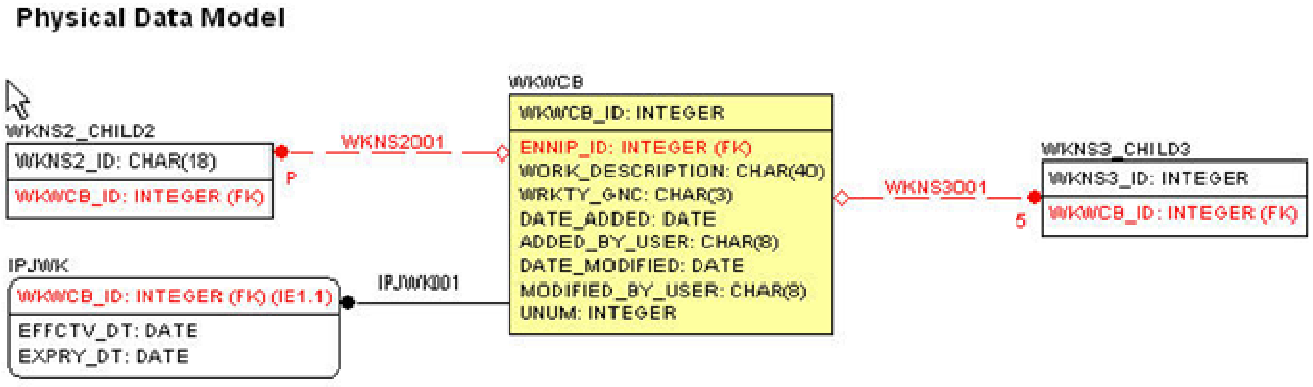
Must be defined using minOccurs and maxOccurs.

Examples

Consider the following Data model, showing the logical and physical perspectives.

The parent will forward reference all non-subtype child entities. The child does **not** backwards reference the parent in non-subtype relationships.





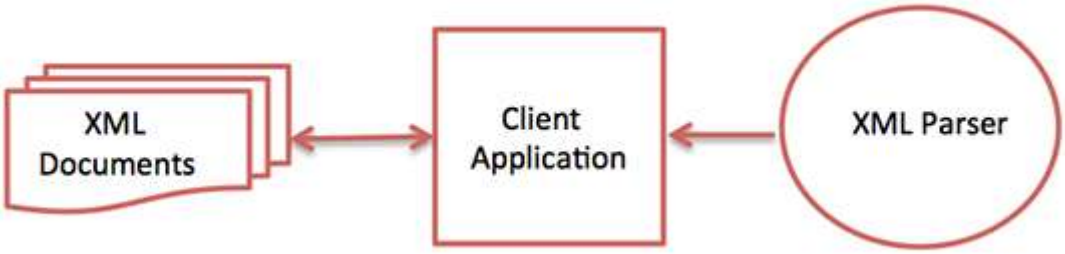

```
<?xml version="1.0" encoding="UTF-8"?>
<!--                ****  XML Schema Generated From ERwin ****    -->

<xsd:schema xmlns="http://www.worksafebc.com" xmlns:xsd="http://www.w3.org/2001/XMLSchema
" targetNamespace="http://www.worksafebc.com"elementFormDefault="unqualified" attributeFormDe
fault="unqualified">

  <!--  *** Standard Data Type Definitions reference *** -->
        <xsd:include schemaLocation="STANDARD DATA TYPES.xsd"/>

      <!--   *** List Non-Subtype Children of Entity: WORK NON SUBTYPE CHILD2 *** -->
```

```
<!--  *** List Parent Entity reference if Entity: WORK NON SUBTYPE CHILD2 is a subtype of another *** -->
        <!--  *** Entity: WORK NON SUBTYPE CHILD2 is not a subtype No parent to list ***  -->

        <xsd:annotation>
        <xsd:documentation> Entity:WORK NON SUBTYPE CHILD2 Table:WKNS2_CHILD2 </xsd:documentation>
        </xsd:annotation>

        <xsd:complexType name="WORK_NON_SUBTYPE_CHILD2_TYPE">
<--This is the name of the coreentity being defined, but is a childreference of entity “WORK” in Figure 2-->
            <xsd:sequence>
            <xsd:element name="WKNS2_ID" type="CHAR18_TYPE"/>
 Element definition: ERwin generated from column name
            <xsd:element name="WKWCB_ID" type="INTEGER_TYPE"/>

        <!--  ***  END ENTITY DEFINITION FOR : WORK NON SUBTYPE CHILD2  ***  -->

        <!-- *** SPECIFY ALL CHILD RELATIONSHIPS EXCLUDING SUBTYPES FOR ENTITY: WORK NON SUBTYPE CHILD2 *** -->
There are no children for this entity, hence none generated
        <!-- *** End  of relationships for WORK NON SUBTYPE CHILD2 *** -->

            </xsd:sequence>
        </xsd:complexType>
</xsd:schema>
```

XML - Parsers

XML parser is a software library or a package that provides interface for client applications to work with XML documents. It checks for proper format of the XML document and may also validate the XML documents. Modern day browsers have built-in XML parsers.

Following diagram shows how XML parser interacts with XML document −



The goal of a parser is to transform XML into a readable code.

To ease the process of parsing, some commercial products are available that facilitate the breakdown of XML document and yield more reliable results.

Some commonly used parsers are listed below −

- **MSXML (Microsoft Core XML Services)** − This is a standard set of XML tools from Microsoft that includes a parser.
- **System.Xml.XmlDocument** − This class is part of .NET library, which contains a number of different classes related to working with XML.
- **Java built-in parser** − The Java library has its own parser. The library is designed such that you can replace the built-in parser with an external implementation such as Xerces from Apache or Saxon.
- **Saxon** − Saxon offers tools for parsing, transforming, and querying XML.
- **Xerces** − Xerces is implemented in Java and is developed by the famous open source Apache Software Foundation.

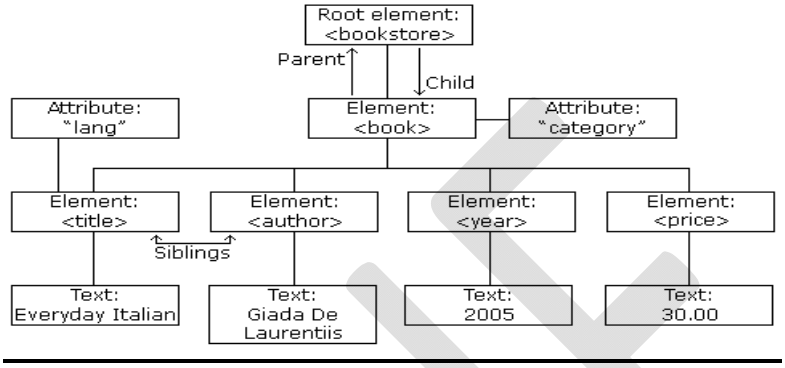
All major browsers have a built-in XML parser to access and manipulate XML.

The XML DOM (Document Object Model) defines the properties and methods for accessing and editing XML.

However, before an XML document can be accessed, it must be loaded into an XML DOM object.

All modern browsers have a built-in XML parser that can convert text into an XML DOM object.

XML DOM



What is the DOM?

The DOM defines a standard for accessing and manipulating documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The HTML DOM defines a standard way for accessing and manipulating HTML documents. It presents an HTML document as a tree-structure.

The XML DOM defines a standard way for accessing and manipulating XML documents. It presents an XML document as a tree-structure.

The HTML DOM

All HTML elements can be accessed through the HTML DOM.

This example changes the value of an HTML element with id="demo":

Example

```
<h1 id="demo">This is a Heading</h1>

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
```

This example changes the value of the first <h1> element in an HTML document:

Example

```
<h1>This is a Heading</h1>

<h1>This is a Heading</h1>

<script>
document.getElementsByTagName("h1")[0].innerHTML = "Hello World!";
</script>
```

Note: Even if the HTML document contains only ONE <h1> element you still have to specify the array index [0], because the getElementsByTagName() method always returns an array.

The XML DOM

All XML elements can be accessed through the XML DOM.

The XML DOM is:

- A standard object model for XML
- A standard programming interface for XML
- Platform- and language-independent
- A W3C standard

In other words: **The XML DOM is a standard for how to get, change, add, or delete XML elements.**

Get the Value of an XML Element

This code retrieves the text value of the first <title> element in an XML document:

Example

```
txt = xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
```

Loading an XML File

The XML file used in the examples below is books.xml.

This example reads "books.xml" into xmlDoc and retrieves the text value of the first <title> element in books.xml:

Example

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
   if (this.readyState == 4 && this.status == 200) {
   myFunction(this);
   }
};
xhttp.open("GET", "books.xml", true);
xhttp.send();

function myFunction(xml) {
   var xmlDoc = xml.responseXML;
   document.getElementById("demo").innerHTML =
   xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
}
```

```
</script>

</body>
</html>
```

Example Explained

- **xmlDoc** - the XML DOM object created by the parser.
- **getElementsByTagName("title")[0]** - get the first <title> element
- **childNodes[0]** - the first child of the <title> element (the text node)
- **nodeValue** - the value of the node (the text itself)

Loading an XML String

This example loads a text string into an XML DOM object, and extracts the info from it with JavaScript:

Example

```
<html>
<body>

<p id="demo"></p>

<script>
var text, parser, xmlDoc;

text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";

parser = new DOMParser();
xmlDoc = parser.parseFromString(text,"text/xml");
```

```
document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>

</body>
</html>
```

Example Explained

A text string is defined:

```
text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";
```

An XML DOM parser is created:

```
parser = new DOMParser();
```

The parser creates a new XML DOM object using the text string:

```
xmlDoc = parser.parseFromString(text,"text/xml");
```

Programming Interface

The DOM models XML as a set of node objects. The nodes can be accessed with JavaScript or other programming languages. In this tutorial we use JavaScript.

The programming interface to the DOM is defined by a set standard properties and methods.

Properties are often referred to as something that is (i.e. nodename is "book").

Methods are often referred to as something that is done (i.e. delete "book").

XML DOM Properties

These are some typical DOM properties:

- x.nodeName - the name of x
- x.nodeValue - the value of x
- x.parentNode - the parent node of x
- x.childNodes - the child nodes of x
- x.attributes - the attributes nodes of x

Note: In the list above, x is a node object.

XML DOM Methods

- x.getElementsByTagName(*name*) - get all elements with a specified tag name
- x.appendChild(*node*) - insert a child node to x
- x.removeChild(*node*) - remove a child node from x

Note: In the list above, x is a node object.

Old Versions of Internet Explorer

Old versions of Internet Explorer (IE5, IE6, IE7, IE8) do not support the DOMParser object.

To handle older versions of Internet Explorer, check if the browser supports the DOMParser object, or else create an ActiveXObject:

Example

```
if (window.DOMParser) {
  // code for modern browsers
  parser = new DOMParser();
  xmlDoc = parser.parseFromString(text,"text/xml");
} else {
  // code for old IE browsers
xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async = false;
  xmlDoc.loadXML(text);
}
```

The XMLHttpRequest Object

The XMLHttpRequest Object has a built in XML Parser - All modern browsers have a built-in XMLHttpRequest object to request data from a server. The XMLHttpRequest object can be used to request data from a web server.

The XMLHttpRequest object is **a developers dream**, because you can:

- Update a web page without reloading the page
- Request data from a server - after the page has loaded
- Receive data from a server - after the page has loaded
- Send data to a server - in the background

The **responseText** property returns the response as a string.

The **responseXML** property returns the response as an XML DOM object.

XML - Processors

When a software program reads an XML document and takes actions accordingly, this is called *processing* the XML. Any program that can read and process XML documents is known as an *XML processor*. An XML processor reads the XML file and turns it into in-memory structures that the rest of the program can access.

The most fundamental XML processor reads an XML document and converts it into an internal representation for other programs or subroutines to use. This is called a *parser*, and it is an important component of every XML processing program.

Processor involves processing the instructions, which can be studied in the chapter Processing Instruction.

Types

XML processors are classified as **validating** or **non-validating** types, depending on whether or not they check XML documents for validity. A processor that discovers a validity error must be able to report it, but may continue with normal processing.

A few validating parsers are − xml4c (IBM, in C++), xml4j (IBM, in Java), MSXML (Microsoft, in Java), TclXML (TCL), xmlproc (Python), XML::Parser (Perl), Java Project X (Sun, in Java).

A few non-validating parsers are − OpenXML (Java), Lark (Java), xp (Java), AElfred (Java), expat (C), XParse (JavaScript), xmllib (Python).

XML to HTML

Loads data from an XML file and displays it as an HTML table.

Display XML Data in HTML

In the last chapter, we explained how to parse XML and access the DOM with JavaScript.

In this example, we loop through an XML file (cd_catalog.xml), and display each CD element as an HTML table row:

```
<html>

<body>

<script type="text/javascript">

var xmlDoc=null;

if (window.ActiveXObject)

{// code for IE

xmlDoc=new ActiveXObject("Microsoft.XMLDOM");

}

else if (document.implementation.createDocument)

{// code for Mozilla, Firefox, Opera, etc.

xmlDoc=document.implementation.createDocument("","",null);

}

else

{

alert('Your browser cannot handle this script');
```

```
}
if (xmlDoc!=null)
{
xmlDoc.async=false;
xmlDoc.load("cd_catalog.xml");
document.write("<table border='1'>");
var x=xmlDoc.getElementsByTagName("CD");
for (i=0;i<x.length;i++)
{
document.write("<tr>");
document.write("<td>");
document.write(
x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue);
document.write("</td>");
document.write("<td>");
document.write(
x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue);
document.write("</td>");
document.write("</tr>");
}
document.write("</table>");
}
```

</script>

</body>

</html>

Example explained

- We check the browser, and load the XML using the correct parser
- We create an HTML table with <table border="1">
- We use getElementsByTagName() to get all XML CD nodes
- For each CD node, we display data from ARTIST and TITLE as table data.
- We end the table with </table>

Access across Domains

For security reasons, a modern browser does not allow access across domains.

This means, that both the web page and the XML file it tries to load, must be located on the same server.

The examples on W3Schools all open XML files located on the W3Schools domain.

If you want to use the example above on one of your web pages, the XML files you load must be located on your own server. Otherwise the xmlDoc.load() method, will generate the error "Access is denied".

XML Application

This chapter demonstrates a small XML application built with HTML and JavaScript

The XML Example Document

Look at the following XML document ("cd_catalog.xml"), that represents a CD catalog:

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

<CATALOG>

<CD>

<TITLE>Empire Burlesque</TITLE>

<ARTIST>Bob Dylan</ARTIST>

<COUNTRY>USA</COUNTRY>

<COMPANY>Columbia</COMPANY>

<PRICE>10.90</PRICE>

<YEAR>1985</YEAR>

</CD>

..

... more ...

..

Load the XML Document

To load the XML document (cd_catalog.xml), we use the same code as we used in the XML Parser chapter:

```
var xmlDoc;

if (window.ActiveXObject)
  {// code for IE
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  }
else if (document.implementation.createDocument)
  {// code for Firefox, Mozilla, Opera, etc.
  xmlDoc=document.implementation.createDocument("","",null);
  }
```

```
else
  {
  alert('Your browser cannot handle this script');
  }
xmlDoc.async=false;
xmlDoc.load("cd_catalog.xml");
```

After the execution of this code, xmlDoc is an XML DOM object, accessible by JavaScript.

Display XML Data as an HTML Table

The following code displays an HTML table filled with data from the XML DOM object:

```
document.write("<table border='1'>");
var x=xmlDoc.getElementsByTagName("CD");
for (var i=0;i<x.length;i++)
{
document.write("<tr>");
document.write("<td>");
document.write(
x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue);
document.write("</td>");
document.write("<td>");
document.write(
x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue);
document.write("</td>");
```

```
document.write("</tr>");

}

document.write("</table>");
```

For each CD element in the XML document, a table row is created. Each table row contains two table data cells with ARTIST and TITLE data from the current CD element.

Display XML Data in any HTML Element

XML data can be copied into any HTML element that can display text.

The code below is part of the <head> section of the HTML file. It gets the XML data from the first <CD> element and displays it in the HTML element with the id="show":

```
var x=xmlDoc.getElementsByTagName("CD");

i=0;

function display()

{

artist=

(x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue);

title=

(x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue);

year=

(x[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue);

txt="Artist: "+artist+"<br />Title: "+title+"<br />Year: "+year;


document.getElementById("show").innerHTML=txt;

}
```

The body of the HTML document contains an **onload** eventattribute that will call the display() function when the page has loaded. It also contains a ** ** element to receive the XML data.

```
<body onload="display()">

<div id='show'></div>

</body>
```

With the example above, you will only see data from the first CD element in the XML document. To navigate to the next line of data, you have to add some more code.

Add a Navigation Script

To add navigation to the example above, create two functions called next() and previous():

```
function next()

{

if (i<x.length-1)

  {

  i++;

  display();

  }

}

function previous()

{

if (i>0)

  {

  i--;
```

```
  display();
  }
}
```

The next() function makes sure that nothing is displayed if you already are at the last CD element, and the previous () function makes sure that nothing is displayed if you already are at the first CD element.

The next() and previous() functions are called by clicking next/previous buttons:

```
<input type="button" onclick="previous()" value="previous" />
<input type="button" onclick="next()" value="next" />
```

Displaying XML with CSS

With CSS (Cascading Style Sheets) you can add display information to an XML document.

Displaying your XML Files with CSS?

It is possible to use CSS to format an XML document.

Below is an example of how to use a CSS style sheet to format an XML document:

Take a look at this XML file:

```
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
```

```
</CD>
<CD>
<TITLE>Hide your heart</TITLE>
<ARTIST>Bonnie Tyler</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS Records</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1988</YEAR>
</CD>
<CD>
<TITLE>Greatest Hits</TITLE>
<ARTIST>Dolly Parton</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>RCA</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1982</YEAR>
</CD>
<CD>
<TITLE>Still got the blues</TITLE>
<ARTIST>Gary Moore</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Virgin records</COMPANY>
<PRICE>10.20</PRICE>
```

<YEAR>1990</YEAR>

</CD>

<CD>

<TITLE>Eros</TITLE>

<ARTIST>Eros Ramazzotti</ARTIST>

<COUNTRY>EU</COUNTRY>

<COMPANY>BMG</COMPANY>

<PRICE>9.90</PRICE>

<YEAR>1997</YEAR>

</CD>

<CD>

<TITLE>One night only</TITLE>

<ARTIST>Bee Gees</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>Polydor</COMPANY>

<PRICE>10.90</PRICE>

<YEAR>1998</YEAR>

</CD>

<CD>

<TITLE>Sylvias Mother</TITLE>

<ARTIST>Dr.Hook</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>CBS</COMPANY>

<PRICE>8.10</PRICE>

<YEAR>1973</YEAR>

</CD>

<CD>

<TITLE>Maggie May</TITLE>

<ARTIST>Rod Stewart</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>Pickwick</COMPANY>

<PRICE>8.50</PRICE>

<YEAR>1990</YEAR>

</CD>

<CD>

<TITLE>Romanza</TITLE>

<ARTIST>Andrea Bocelli</ARTIST>

<COUNTRY>EU</COUNTRY>

<COMPANY>Polydor</COMPANY>

<PRICE>10.80</PRICE>

<YEAR>1996</YEAR>

</CD>

<CD>

<TITLE>When a man loves a woman</TITLE>

<ARTIST>Percy Sledge</ARTIST>

<COUNTRY>USA</COUNTRY>

<COMPANY>Atlantic</COMPANY>

<PRICE>8.70</PRICE>

<YEAR>1987</YEAR>

</CD>

<CD>

<TITLE>Black angel</TITLE>

<ARTIST>Savage Rose</ARTIST>

<COUNTRY>EU</COUNTRY>

<COMPANY>Mega</COMPANY>

<PRICE>10.90</PRICE>

<YEAR>1995</YEAR>

</CD>

<CD>

<TITLE>1999 Grammy Nominees</TITLE>

<ARTIST>Many</ARTIST>

<COUNTRY>USA</COUNTRY>

<COMPANY>Grammy</COMPANY>

<PRICE>10.20</PRICE>

<YEAR>1999</YEAR>

</CD>

<CD>

<TITLE>For the good times</TITLE>

<ARTIST>Kenny Rogers</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>Mucik Master</COMPANY>

<PRICE>8.70</PRICE>

<YEAR>1995</YEAR>

</CD>

<CD>

<TITLE>Big Willie style</TITLE>

<ARTIST>Will Smith</ARTIST>

<COUNTRY>USA</COUNTRY>

<COMPANY>Columbia</COMPANY>

<PRICE>9.90</PRICE>

<YEAR>1997</YEAR>

</CD>

<CD>

<TITLE>Tupelo Honey</TITLE>

<ARTIST>Van Morrison</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>Polydor</COMPANY>

<PRICE>8.20</PRICE>

<YEAR>1971</YEAR>

</CD>

<CD>

<TITLE>Soulsville</TITLE>

<ARTIST>Jorn Hoel</ARTIST>

<COUNTRY>Norway</COUNTRY>

<COMPANY>WEA</COMPANY>

<PRICE>7.90</PRICE>

<YEAR>1996</YEAR>

</CD>

<CD>

<TITLE>The very best of</TITLE>

<ARTIST>Cat Stevens</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>Island</COMPANY>

<PRICE>8.90</PRICE>

<YEAR>1990</YEAR>

</CD>

<CD>

<TITLE>Stop</TITLE>

<ARTIST>Sam Brown</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>A and M</COMPANY>

<PRICE>8.90</PRICE>

<YEAR>1988</YEAR>

</CD>

<CD>

<TITLE>Bridge of Spies</TITLE>

<ARTIST>T'Pau</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>Siren</COMPANY>

<PRICE>7.90</PRICE>

<YEAR>1987</YEAR>

</CD>

<CD>

<TITLE>Private Dancer</TITLE>

<ARTIST>Tina Turner</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>Capitol</COMPANY>

<PRICE>8.90</PRICE>

<YEAR>1983</YEAR>

</CD>

<CD>

<TITLE>Midt om natten</TITLE>

<ARTIST>Kim Larsen</ARTIST>

<COUNTRY>EU</COUNTRY>

<COMPANY>Medley</COMPANY>

<PRICE>7.80</PRICE>

<YEAR>1983</YEAR>

</CD>

<CD>

<TITLE>Pavarotti Gala Concert</TITLE>

<ARTIST>Luciano Pavarotti</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>DECCA</COMPANY>

<PRICE>9.90</PRICE>

<YEAR>1991</YEAR>

</CD>

<CD>

<TITLE>The dock of the bay</TITLE>

<ARTIST>Otis Redding</ARTIST>

<COUNTRY>USA</COUNTRY>

<COMPANY>Atlantic</COMPANY>

<PRICE>7.90</PRICE>

<YEAR>1987</YEAR>

</CD>

<CD>

<TITLE>Picture book</TITLE>

<ARTIST>Simply Red</ARTIST>

<COUNTRY>EU</COUNTRY>

<COMPANY>Elektra</COMPANY>

<PRICE>7.20</PRICE>

<YEAR>1985</YEAR>

```
</CD>
<CD>
<TITLE>Red</TITLE>
<ARTIST>The Communards</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>London</COMPANY>
<PRICE>7.80</PRICE>
<YEAR>1987</YEAR>
</CD>
<CD>
<TITLE>Unchain my heart</TITLE>
<ARTIST>Joe Cocker</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>EMI</COMPANY>
<PRICE>8.20</PRICE>
<YEAR>1987</YEAR>
</CD>
</CATALOG>
```

Then look at this style sheet (CSS):

```
CATALOG
{
background-color: #ffffff;
width: 100%;
```

```
}
CD
{
display: block;
margin-bottom: 30pt;
margin-left: 0;
}
TITLE
{
color: #FF0000;
font-size: 20pt;
}
ARTIST
{
color: #0000FF;
font-size: 20pt;
}
COUNTRY,PRICE,YEAR,COMPANY
{
display: block;
color: #000000;
margin-left: 20pt;
}
```

KAHE

Below is a fraction of the XML file. The second line links the XML file to the CSS file:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
<CD>
<TITLE>Hide your heart</TITLE>
<ARTIST>Bonnie Tyler</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS Records</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1988</YEAR>
</CD>
....</CATALOG>
```

Formatting XML with CSS is not the most common method. Use JavaScript or XSLT instead.

Beyond the Basics-More Scripting Techniques:

XML is used in many aspects of web development.

XML is often used to separate data from presentation.

XML Separates Data from Presentation

XML does not carry any information about how to be displayed.

The same XML data can be used in many different presentation scenarios.

Because of this, with XML, there is a full separation between data and presentation.

XML is Often a Complement to HTML

In many HTML applications, XML is used to store or transport data, while HTML is used to format and display the same data.

XML Separates Data from HTML

When displaying data in HTML, you should not have to edit the HTML file when the data changes.

With XML, the data can be stored in separate XML files.

With a few lines of JavaScript code, you can read an XML file and update the data content of any HTML page.

Books.xml

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

 <book category="cooking">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
 </book>

 <book category="children">
```

```
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
 </book>

 <book category="web">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
 </book>

 <book category="web" cover="paperback">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
 </book>

</bookstore>
```

Transaction Data

Thousands of XML formats exist, in many different industries, to describe day-to-day data transactions:

1. Stocks and Shares
2. Financial transactions
3. Medical data
4. Mathematical data
5. Scientific measurements
6. News information
7. Weather services

Example: XML News

XML News is a specification for exchanging news and other information.

Using a standard makes it easier for both news producers and news consumers to produce, receive, and archive any kind of news information across different hardware, software, and programming languages.

An example XMLNews document:

```
<?xml version="1.0" encoding="UTF-8"?>
<nitf>
  <head>
    <title>Colombia Earthquake</title>
  </head>
  <body>
    <headline>
      <hl1>143 Dead in Colombia Earthquake</hl1>
    </headline>
    <byline>
      <bytag>By Jared Kotler, Associated Press Writer</bytag>
    </byline>
    <dateline>
      <location>Bogota, Colombia</location>
      <date>Monday January 25 1999 7:28 ET</date>
    </dateline>
  </body>
</nitf>
```

Example: XML Weather Service

An XML national weather service from NOAA (National Oceanic and Atmospheric Administration):

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

<credit>NOAA's National Weather Service</credit>
<credit_URL>http://weather.gov/</credit_URL>

<image>
  <url>http://weather.gov/images/xml_logo.gif</url>
  <title>NOAA's National Weather Service</title>
  <link>http://weather.gov</link>
</image>
```

<location>New York/John F. Kennedy Intl Airport, NY</location>
<station_id>KJFK</station_id>
<latitude>40.66</latitude>
<longitude>-73.78</longitude>
<observation_time_rfc822>Mon, 11 Feb 2008 06:51:00 -0500 EST
</observation_time_rfc822>

<weather>A Few Clouds</weather>
<temp_f>11</temp_f>
<temp_c>-12</temp_c>
<relative_humidity>36</relative_humidity>
<wind_dir>West</wind_dir>
<wind_degrees>280</wind_degrees>
<wind_mph>18.4</wind_mph>
<wind_gust_mph>29</wind_gust_mph>
<pressure_mb>1023.6</pressure_mb>
<pressure_in>30.23</pressure_in>
<dewpoint_f>-11</dewpoint_f>
<dewpoint_c>-24</dewpoint_c>
<windchill_f>-7</windchill_f>
<windchill_c>-22</windchill_c>
<visibility_mi>10.00</visibility_mi>

<icon_url_base>http://weather.gov/weather/images/fcicons/</icon_url_base>
<icon_url_name>nfew.jpg</icon_url_name>
<disclaimer_url>http://weather.gov/disclaimer.html</disclaimer_url>
<copyright_url>http://weather.gov/disclaimer.html</copyright_url>

</current_observation>

POSSIBLE QUESTIONS

2 MARKS

1. What is a Script Language?
2. Define Scripts ant it types.
3. Define XML parent child relationships.
4. What is an XML Parsers?
5. List the commonly used parsers in XML.
6. What is a DOM?
7. What is HTML DOM?
8. What is XML DOM?
9. List any two XML DOM Properties and Methods?
10. Define the XMLHttpRequest Object?
11. What is XML-Processors and its types?
12. What is XML CSS?

8 MARKS

1. Give detailed explanation about Scripts and its types?
2. Explain XML Parent-Child Relationship.
3. Elucidate XML DOM with example.
4. Write a programhow to parse XML and access the DOM with JavaScript.
5. Write a program how to use a CSS style sheet to format an XML document.
6. Give details: Beyond the Basics-More Scripting Techniques.

KARPAGAM ACADEMY OF HIGHER EDUCATION

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

II B.Sc CS (Batch 2017-2020)

XML PROGRAMMING

PART - A OBJECTIVE TYPE/MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS

UNIT-III

SNO	QUESTIONS	OPT1	OPT1	OPT3	OPT4	ANSWER
1	Which of these is to manipulate the contents of	XSL	DTD	CSS	DOM	DOM
2	W3C provides a standard recommendation to build the	XSL	DTD	CSS	DOM	DOM
3	DOM based parsers exposes a programatic library called	DOM-Application Programming Interface	Simple Application programming Interface for XML	Simple Application programming Interface for DOM	High Level Programming Interface	DOM-Application Programming Interface
4	DOM Interfaces are _____	Low Level Programmi	Platform independent	Platform dependen	Java Programs	Platform independent
5	_____ is a java based DOM Application Interface	JAPX	Jscript	JDOM	J#	JDOM
6	In the DOM tree the elements, attributes, contents etc., are	tags	nodes	branches	methods	nodes
7	The DOM tree is constructed in the _____	xml file	application program	computers memory	dtd file	computers memory
8	IBM's XML parser for Java is _____	JAPX	XML4J	JDOM	J#	XML4J
9	var xmldoc= new ActiveXObject("	Microsoft. XMLDOC	Microsoft.XML DOM	ActiveXObject.XMLD	XMLDOM	Microsoft.XML DOM
10	In Javascript_____ method is called to get the xml	get( )	post( )	load( )	parse( )	load( )
11	Property_____ corresponds to the	document Root	documentElement	document Firstchild	document Node	documentElement
12	Property _____ corresponds to the name of an	document Node	nodeList	document Name	nodeName	nodeName
13	_____property is used to get the number of child	length	childNodes	nodeList	getChildren	length
14	Which of these interface represents the XML	Node	Root	Head	Document	Document
15	Which of these do not derives from Node interface	Element	Attr	Text	Character Data	Text

16	CDATASection interface derives from _____	Comment	Text	Character Data	Node	Text
17	Which of these is not a node type	TEXT_NODE	COMMENT_NODE	ENTITY_NODE	ELEMENT_NODE	ENTITY_NODE
18	Individual child nodes of an element is accessed using the	child( )	individual( )	node( )	item( )	item( )
19	Nodes at the same level in a document is called _____	siblings	sisters	children	neighbour	siblings
20	The nodeValue method returns the value of the	Element node	Attribute node	Text Node	Comment Node	Text Node
21	The Element nodes have a node value _____	text	numbers	any data type	null	null
22	_____ returns the node in the previous level in the	parentNode()	firstChild	lastChild	previousChild	parentNode()
23	_____ interface represents an attribute node.	Attribute	Attr	AttList	Attrib	Attr
24	Which method of the Node interface is used to duplicate a	addChild	duplicateNode	cloneNode	appendChild	cloneNode
25	Which method of the Node interface is used to add a child	addChild	duplicateNode	cloneNode	appendChild	appendChild
26	Which method of the Node interface is used to delete a	replaceChild	removeChild	deleteNode	deleteChild	removeChild
27	_____method is used to create a new Document	createDocument	createDOM	newDocument	Document.newInstan	newDocument
28	_____ method is used to create a comment	<!--comment -->	createComment	newComment	None of the above	createComment
29	_____ method of the XmlDocument is used to	write	output	display	print	write
30	_____ uses an event based model for parsing the	DOM	SAX	XML	JAXP	SAX
31	SAX was developed by----------	Tim Bernes Lee	H.M.Dietel and team	members of XML-	Charles Goldfarb,	members of XML-DEV
32	In SAX, notifications called _____ is raised when the	nodes	methods	subroutines	events	events
33	DOM is a _____ model	tree based	event based	object based	procedure based	tree based
34	Which of these is not true about SAX	Document's data is	Raises events when parsed	Invokes methods	XML data is passed	Document's data is stored
35	SAX parsers invokes methods when _____ is encountered	EOF	Nodes	CDATA	Markup	Markup
36	In SAX2.0 HandlerBase class us replaced by _____	EventHandler	DocumentHandler	DefaultHandler	HandlerMethod	DefaultHandler

37	_____ package provides the SAX programmatic	org.xml.parsers	org.xml.sax	javax.xml.parsers	javax.xml.sax	org.xml.sax
38	_____ package provides classes and instantiating DOM	org.xml.parsers	org.xml.sax	javax.xml.parsers	javax.xml.sax	javax.xml.parsers
39	Class HandlerBase implements _____ interfaces	9	1	4	5	4
40	Class HandlerBase implements _____ interface for	EventHandler	DocumentHandler	DefaultHandler	EntityResolver	EntityResolver
41	Class HandlerBase implements _____ interface for	ErrorHandler	DocumentHandler	DTDHandler	EntityResolver	DTDHandler
42	Error Handler is a/an _____ for handling errors.	class	interface	method	event	interface
43	____ interface is for handling parsing events.	EventHandler	DocumentHandler	DTDHandler	EntityResolver	DocumentHandler
44	Which of these is not an interface that is implemented	EventHandler	DocumentHandler	DTDHandler	EntityResolver	EventHandler
45	_____ method provides access to the parsed	setDocumentLocator	getDocumentURL	getDocumentLocato	getDocument	setDocumentLocator
46	getSystemID method is called to retrieve a document's	SystemIP address	XML document's	XML document	XML elements	XML document's
47	_____ method is called when the document's root	startDocument	setDocumentLocator	BOF	getSystemID	startDocument
48	startElement method is called _____	when the root	when a start tag is encountered.	when the BOF is	WhenEOF is	when a start tag is
49	Which method is called last and only once	startDocument	endDocument	startElement	endElement	endDocument
50	What are the 2 arguments passed to method	Element name and	Element name and its parent	Element name and	Element name and	Element name and its
51	AttributList method getLength returns the ____	number of attributes	size of the attribute	position of the last	size of the attributes	number of attributes the
52	In AttributeList the first attribute is at _____ position	0	1	Element position +	Parent position +	0
53	HandlerBase class member characters method is invoked	Text data	CDATA	comments	character data	character data
54	SAX parsers generate _____ when the XML document	fatal errors	validation errors	non fatal errors	warnings	fatal errors
55	SAX parsers generate warnings if the _____	XML document	XML document is not valid	DTD is inconsiste	Xml document	DTD is inconsistent
56	SAX Parsers generates _____ while processing Invalidation	fatal errors	nonfatal errors	warnings	no errors	nonfatal errors
57	_____ method of the HandlerBase class in invoked	ErrorHandler	error	exception	warning	warning

UNIT IV

SYLLABUS

Other XML Concepts: XML as Data, Linking with XML

XML as Data:

Extensible Markup Language (**XML**) is used to describe **data**. The **XML** standard is a flexible way to create information formats and electronically share structured **data** via the public Internet, as well as via corporate networks.

Primitive XML Data Types

The following table lists primitive XML schema data types, facets that can be applied to the data type, and a description of the data type. For descriptions of the facets.
Facets can only appear once in a type definition except or **enumeration** and **pattern** facets. **Enumeration** and **pattern** facets can have multiple entries and are grouped together.

Data Type	Facets	Description
string	length, pattern, maxLength, minLength, enumeration, whiteSpace	Represents character strings.
boolean	pattern, whiteSpace	Represents Boolean values, which are either **true** or **false**.
decimal	enumeration, pattern, totalDigits,	Represents arbitrary precision numbers.

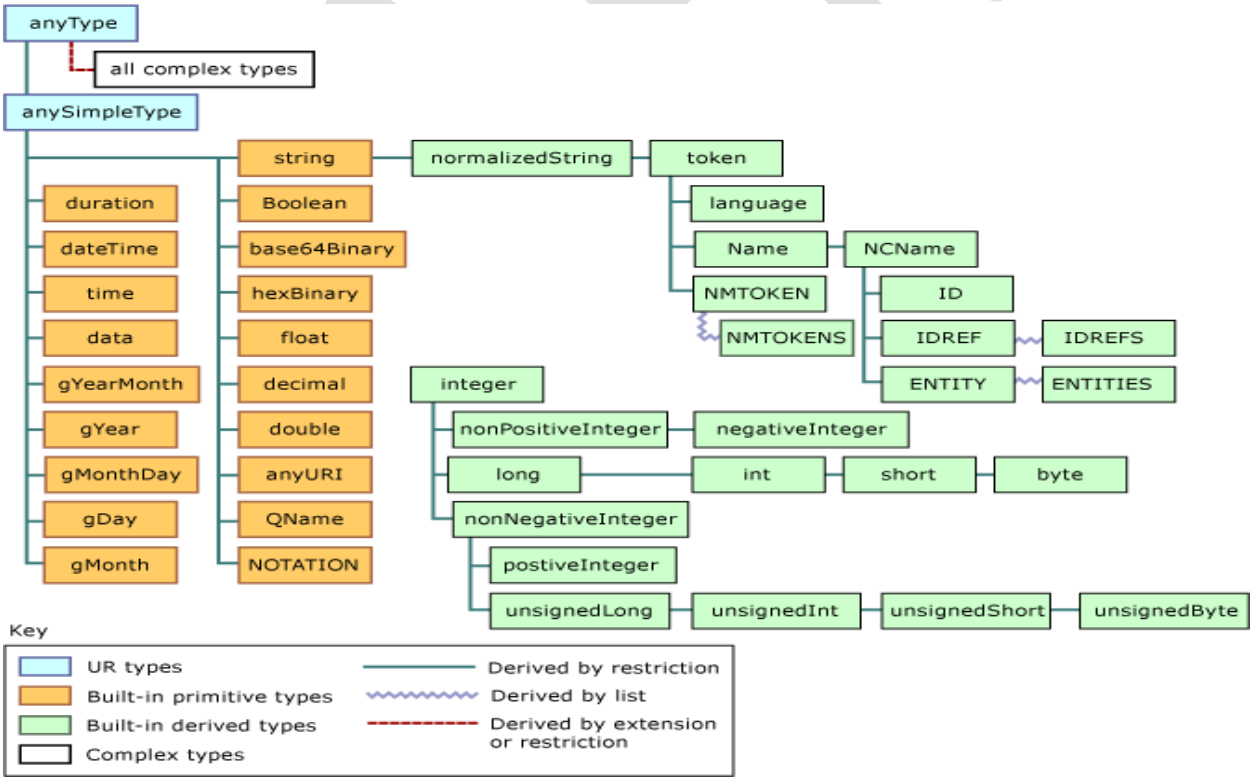
	fractionDigits, minInclusive, maxInclusive, maxExclusive, whiteSpace	
float	pattern, enumeration, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace	Represents single-precision 32-bit floating-point numbers.
double	pattern, enumeration, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace	Represents double-precision 64-bit floating-point numbers.
duration	enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace	Represents a duration of time. The pattern for **duration** is PnYnMnDTnHnMnS, where nY represents the number of years, nM the number of months, nD the number of days, T the date/time separator, nH the number of hours, nM the number of minutes, and nS the number of seconds.
dateTime	enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace	Represents a specific instance of time. The pattern for **dateTime** is CCYY-MM-DDThh:mm:ss where CC represents the century, YY the year, MM the month, and DD the day, preceded by an optional leading negative (-) character to indicate a negative number. If the negative character is omitted, positive (+) is assumed. The T is the date/time separator and hh, mm, and ss represent hour, minute, and second respectively. Additional digits can be used to increase the precision of fractional seconds if desired. For example, the format ss.ss... with any number of digits

		after the decimal point is supported. The fractional seconds part is optional. This representation may be immediately followed by a "Z" to indicate Coordinated Universal Time (UTC) or to indicate the time zone. For example, the difference between the local time and Coordinated Universal Time, immediately followed by a sign, + or -, followed by the difference from UTC represented as hh:mm (minutes is required). If the time zone is included, both hours and minutes must be present.
time	enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace	Represents an instance of time that recurs every day. The pattern for **time** is hh:mm:ss.sss with optional time zone indicator.
date	enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace	Represents a calendar date. The pattern for **date** is CCYY-MM-DD with optional time zone indicator as allowed for **dateTime**.
gYearMonth	enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace	Represents a specific Gregorian month in a specific Gregorian year. A set of one-month long, nonperiodic instances. The pattern for **gYearMonth** is CCYY-MM with optional time zone indicator.
gYear	enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive,	Represents a Gregorian year. A set of one-year long, nonperiodic instances. The pattern for **gYear** is CCYY with optional time zone indicator as allowed for **dateTime**.

	whiteSpace	
gMonthDay	enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace	Represents a specific Gregorian date that recurs, specifically a day of the year such as the third of May. A **gMonthDay** is the set of calendar dates. Specifically, it is a set of one-day long, annually periodic instances. The pattern for **gMonthDay** is --MM-DD with optional time zone indicator as allowed for **date**.
gDay	enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace	Represents a Gregorian day that recurs, specifically a day of the month such as the fifth day of the month. A **gDay** is the space of a set of calendar dates. Specifically, it is a set of one-day long, monthly periodic instances. The pattern for **gDay** is ---DD with optional time zone indicator as allowed for **date**.
gMonth	enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace	Represents a Gregorian month that recurs every year. A **gMonth** is the space of a set of calendar months. Specifically, it is a set of one-month long, yearly periodic instances. The pattern for **gMonth** is --MM-- with optional time zone indicator as allowed for **date**.
hexBinary	length, pattern, maxLength, minLength, enumeration, whiteSpace	Represents arbitrary hex-encoded binary data. A **hexBinary** is the set of finite-length sequences of binary octets. Each binary octet is encoded as a character tuple, consisting of two hexadecimal digits ([0-9a-fA-F]) representing the octet code.
base64Binary	length, pattern, maxLength, minLength, enumeration, whiteSpace	Represents Base64-encoded arbitrary binary data. A **base64Binary** is the set of finite-length sequences of binary octets.
anyURI	length, pattern, maxLength,	Represents a URI as defined by RFC 2396. An **anyURI** value can be absolute or relative, and may

	minLength, enumeration, whiteSpace	have an optional fragment identifier.
QName	length, enumeration, pattern, maxLength, minLength, whiteSpace	Represents a qualified name. A qualified name is composed of a prefix and a local name separated by a colon. Both the prefix and local names must be an NCName. The prefix must be associated with a namespace URI reference, using a namespace declaration.
NOTATION	length, enumeration, pattern, maxLength, minLength, whiteSpace	Represents a **NOTATION** attribute type. A set of QNames.

Primitive XML Data Types



XML Namespaces

XML Namespaces provide a method to avoid element name conflicts.

Name Conflicts

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

This XML carries HTML table information:

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a table (a piece of furniture):

```
<table>
  <name>AfricanCoffee``       Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

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

A user or an XML application will not know how to handle these differences.

Solving the Name Conflict Using a Prefix

Name conflicts in XML can easily be avoided using a name prefix.

This XML carries information about an HTML table, and a piece of furniture:

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

```
<f:table>
  <f:name>AfricanCoffeeTable</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

In the example above, there will be no conflict because the two <table> elements have different names.

XML Namespaces - The xmlns Attribute

When using prefixes in XML, a **namespace** for the prefix must be defined.
The namespace can be defined by an **xmlns** attribute in the start tag of an element.
The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".

```
<root>
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="https://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
</root>
```

In the example above:

The xmlns attribute in the first <table> element gives the h: prefix a qualified namespace.
The xmlns attribute in the second <table> element gives the f: prefix a qualified namespace.
When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

Namespaces can also be declared in the XML root element:

```
<root xmlns:h=http://www.w3.org/TR/html4/  xmlns:f="https://www.w3schools.com/furniture">

<h:table>
  <h:tr>
    <h:td>Apples</h:td>
```

```
  <h:td>Bananas</h:td>
 </h:tr>
</h:table>

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

</root>
```

Note:

- The namespace URI is not used by the parser to look up information.
- The purpose of using an URI is to give the namespace a unique name.
- However, companies often use the namespace as a pointer to a web page containing namespace information.

Uniform Resource Identifier (URI)

A **Uniform Resource Identifier** (URI) is a string of characters which identifies an Internet Resource.

The most common URI is the **Uniform Resource Locator** (URL) which identifies an Internet domain address. Another, not so common type of URI is the **Universal Resource Name** (URN).

Default Namespaces

Defining a default namespace for an element saves us from using prefixes in all the child elements. It has the following syntax:

xmlns="*namespaceURI*"

This XML carries HTML table information:

```
<table xmlns="http://www.w3.org/TR/html4/">
 <tr>
  <td>Apples</td>
  <td>Bananas</td>
 </tr>
</table>
```

This XML carries information about a piece of furniture:

```
<table xmlns="https://www.w3schools.com/furniture">
 <name>African Coffee Table</name>
```

```
  <width>80</width>
  <length>120</length>
</table>
```

Namespaces in Real Use

XSLT is a language that can be used to transform XML documents into other formats. The XML document below is a document used to transform XML into HTML.

The namespace "http://www.w3.org/1999/XSL/Transform" identifies XSLT elements inside an HTML document:

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
  <h2>MyCDCollection</h2>
  <table border="1">
    <tr>
      <th style="text-align:left">Title</th>
      <th style="text-align:left">Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Using the XML Data Source Object.

The **XML Data Source Object** (DSO) is a Microsoft ActiveX control that's built into Microsoft Internet Explorer 4+. **Using** this **object**, it is possible to extract content from an external **XML** file, or **XML data** embedded in an HTML file, into an HTML page.

What is XML Data Source Object?

XML Data Source Object is a Microsoft Active X object built into the web browser so we can use ActiveX control to extract data from the XML code embedded directly in the HTML file and the external XML file using data binding into HTML web pages.

Data islands

- XML code is embedded in an HTML document to create data islands.
- The file should be saved with .html or .htm extension.

Data islands are created with two methods:

1. Explicit method: Embedding data directly.

Syntax:

```
<XML ID="xmlID">
   <!-- XML data-->
</XML>
```

2. Implicit method: Embedding XML data with reference to external XML file.

Syntax:

```
<XML ID="xmlID" SRC="filename.xml"></XML>
```

- The DSO object is implicitly created when we use a XML data island.
- To extract data from external data file use the HTML tags.
  <IMG>, <LABLE>, <TABLE>
- Two attributes are very important along with html tags.

a) DATASRC: Specifies the source of data.

b) DATAFLD: Specifies the field from where the data is to be displayed.

XML-DSO object is used to load external XML document.

The steps to do this are as follows:

- Create and initialize XML-DSO object.

Syntax:

<OBJECT ID="SomeID CLASSID=CLSID"></OBJECT>

- Load the external XML file in HTML page using the XMLDSO. We have to use Javascript in the <HEAD> section of HTML page.
- Extract data from loaded XML file through HTML tags using DATASRC and DATAFLD.

Examples

First, we'll take a look at how to extract data from XML data islands (XML data that's included in the HTML page itself).

Take a look at the following code:

```
<!-- example1.htm -->
<html>
<head>
<title>XML DSO-example1.htm</title>
</head>
<body bgcolor="#FFFFFF">
<xml id="xmldb">
  <db>
    <member>
      <name>Premshree Pillai<name>
      <sex>male</sex>
    </member>
    <member>
      <name>Vinod</name>
      <sex>male</sex>
    </member>
  </db>
</xml>
<span datasrc="#xmldb" datafld="name"<</span>
<br>
<span datasrc="#xmldb" datafld="sex"></span>
```

```
</body>
</html>
```

The output of the above is:

Premshree Pillai
male

Note that, in the code for example1.htm, we have not initialized an XML-DSO object. Thus, when you use a XML data island, the object is implicitly created.

In the above code, we've included an XML data island using the <XML> tag. We have assigned it an ID, xmldb, for use later. Now, we can extract data from the XML data island using HTML tags like <A>, <SPAN>, <DIV> etc. As you can see, we have extracted data here using the <SPAN> tag. Note the attributes datasrc and datafld in the <SPAN> tag. datasrc is used to specify the ID of the data island you want to extract data from. datafld is used to specify the XML tag you want to extract the data from (here, name in first <SPAN> and sex in second <SPAN>).
Note that we have two <name> and <sex> tags in our XML data island, but that, using the above method, we can extract only the first instances of these tags. To extract all instances, we have to use the <TABLE> tag. Take a look at the following example:

Example: Write a program to display content in table using XML-DSO

```
<!-- example2.htm -->
<html>
<head>
<title>XML DSO-example2.htm</title>
</head>
<body bgcolor="#FFFFFF">
<xml id="xmldb">
  <db>
    <member>
      <name>Premshree Pillai<name>
      <sex>male</sex>
    </member>
    <member>
      <name>Vinod</name>
      <sex>male</sex>
```

```
    </member>
  </db>
</xml>
<table datasrc="#xmldb" border="1">
  <thead>
    <th>Name</th>
    <th>Sex</th>
  </thead>
  <tr>
    <td><div datafld="name"></div></td>
    <td><div datafld="sex"></div></td>
  </tr>
</table>
</body>
</html>
```

Output:

Name	Sex
Premshree Pillai	male
Vinod	male

Here, we've used the <TABLE> tag and extracted the contents using the HTML tag, <DIV>, within the HTML column tag, <TD>. The table will automatically iterate through each instance of <member> (the parent of <name> and <sex>), thus, we can display all the names and ages in a formatted way.

It's also possible to manipulate the XML DSO object using JavaScript.

Consider the following XML file:

```
<!-- example4.xml -->
<?xml version="1.0" ?>
<myDB>
  <member>
    <name>Premshree Pillai</name>
      <sex>male</sex>
  </member>
  <member>
    <name>Vinod</name>
```

```
    <sex>male</sex>
  </member>
  <member>
    <name>Santhosh</name>
    <sex>male</sex>
  </member>
</myDB>
```

Now, consider the following HTML file:

```
<!-- example4.htm -->
<html>
<head>
<title>XML DSO-example4.htm</title>
<script language="JavaScript">
function load() {
  var xmlDso=myDB.XMLDocument;
  xmlDso.load("example4.xml");

  /* Get the complete record set */
  var memberSet=myDB.recordset;

  /* Go to next data */
  memberSet.moveNext();
}
</script>
</head>
<body bgcolor="#FFFFFF" onLoad="load()">

<object id="myDB" CLASSID="clsid:550dda30-0541-11d2-9ca9-
0060b0ec3d39" width="0" height="0">
</object>

<span datasrc="#myDB" datafld="name"></span>

</body>
</html>
```

Output:

Vinod

The above script is fairly self explanatory. Initially, we store the entire data of the data file into a variable memberSet using the recordset method. The moveNext() method points to the next data item (next row). Some of other methods that can be used here are:

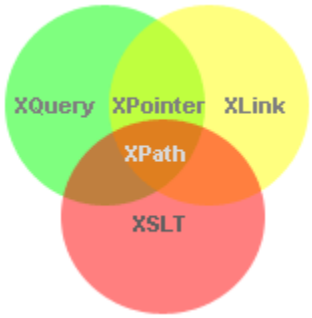
- movePrevious(): Point to the previous data item.
- moveFirst(): Point to the first data item.
- moveLast(): Point to the last data item.
- EOF: This property is used to check whether we've reached the end of the data.

Note that, in the above methods, the data is pointed relative to parent of the nodes being displayed.

Linking with XML:

Simple Links the HTML Way - **Links** are found in nearly all web pages. **Links** allow users to click their **way** from page to page. **HTML Links** - Hyperlinks. **HTML links** are hyperlinks. You can click on a **link** and jump to another document. When you move the mouse over a **link**, the mouse arrow will turn into a little hand. Note: A **link** does not have to be text.

- Linking in XML is divided into two parts: XLink and XPointer.
- XLink defines a standard way of creating hyperlinks in XML documents.
- XPointer allows the hyperlinks to point to more specific parts (fragments) in the XML document.



XLink: The XML Linking Mechanism

XLink is used to create hyperlinks in XML documents.

- XLink is used to create hyperlinks within XML documents
- Any element in an XML document can behave as a link
- With XLink, the links can be defined outside the linked files
- XLink is a W3C Recommendatio

XLink Browser Support

There is no browser support for XLink in XML documents. However, all major browsers support XLinks

XLink Syntax

In HTML, the <a> element defines a hyperlink. However, this is not how it works in XML. In XML documents, you can use whatever element names you want - therefore it is impossible for browsers to predict what link elements will be called in XML documents.

Below is a simple **example** of how to use XLink to create links in an XML document:

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

<homepages xmlns:xlink="http://www.w3.org/1999/xlink">
  <homepage xlink:type="simple" xlink:href="https://www.w3schools.com">Visit W3Schools</homepage>
  <homepage xlink:type="simple" xlink:href="http://www.w3.org">Visit W3C</homepage>
</homepages>
```

To get access to the XLink features we must declare the XLink namespace. The XLink namespace is: "http://www.w3.org/1999/xlink".

The xlink:type and the xlink:href attributes in the <homepage> elements come from the XLink namespace.

The xlink:type="simple" creates a simple "HTML-like" link (means "click here to go there").
The xlink:href attribute specifies the URL to link to.
XLink Example
The following XML document contains XLink features:

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

<bookstore xmlns:xlink="http://www.w3.org/1999/xlink">

<book title="Harry Potter">
  <description
  xlink:type="simple"
  xlink:href="/images/HPotter.gif"
  xlink:show="new">
```

```
  As his fifth year at Hogwarts School of Witchcraft and
  Wizardry approaches, 15-year-old Harry Potter is.....
  </description>
</book>

<book title="XQuery Kick Start">
  <description
  xlink:type="simple"
  xlink:href="/images/XQuery.gif"
  xlink:show="new">
  XQuery Kick Start delivers a concise introduction
  to the XQuery standard.....
  </description>
</book>

</bookstore>
```

Example explained:

- The XLink namespace is declared at the top of the document (xmlns:xlink="http://www.w3.org/1999/xlink")
- The xlink:type="simple" creates a simple "HTML-like" link
- The xlink:href attribute specifies the URL to link to (in this case - an image)
- The xlink:show="new" specifies that the link should open in a new window

XLink - Going Further

In the example above we have demonstrated simple XLinks. XLink is getting more interesting when accessing remote locations as resources, instead of standalone pages.

If we set the value of the xlink:show attribute to "embed", the linked resource should be processed inline within the page. When you consider that this could be another XML document you could, for example, build a hierarchy of XML documents.

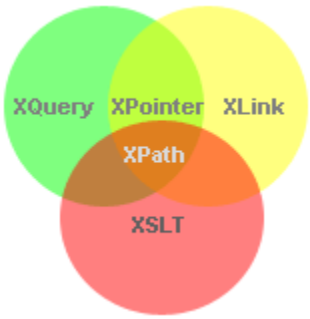
You can also specify WHEN the resource should appear, with the xlink:actuate attribute.

XLink Attribute Reference

Attribute	Value	Description

xlink:actuate	onLoad onRequest other none	Defines when the linked resource is read and shown: • onLoad - the resource should be loaded and shown when the document loads • onRequest - the resource is not read or shown before the link is clicked
xlink:href	*URL*	Specifies the URL to link to
xlink:show	embed new replace other none	Specifies where to open the link. Default is "replace"
xlink:type	simple extended locator arc resource title none	Specifies the type of link

XPointer:



- XPointer allows links to point to specific parts of an XML document
- XPointer uses XPath expressions to navigate in the XML document
- XPointer is a W3C Recommendation

XPointer Browser Support

There is no browser support for XPointer. But XPointer is used in other XML languages.

XPointer Example

In this example, we will use XPointer in conjunction with XLink to point to a specific part of another document.

We will start by looking at the target XML document (the document we are linking to):

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

<dogbreeds>

<dog breed="Rottweiler" id="Rottweiler">
  <picture url="https://dog.com/rottweiler.gif" />
  <history>The Rottweiler's ancestors were probably Roman
  drover dogs.....</history>
  <temperament>Confident, bold, alert and imposing, the Rottweiler
  is a popular choice for its ability to protect....</temperament>
</dog>

<dog breed="FCRetriever" id="FCRetriever">
  <picture url="https://dog.com/fcretriever.gif" />
  <history>One of the earliest uses of retrieving dogs was to
  help fishermen retrieve fish from the water....</history>
  <temperament>The flat-coated retriever is a sweet, exuberant,
  lively dog that loves to play and retrieve....</temperament>
</dog>

</dogbreeds>
```

Note that the XML document above uses id attributes on each element!

So, instead of linking to the entire document (as with XLink), XPointer allows you to link to specific parts of the document. To link to a specific part of a page, add a number sign (#) and an

XPointer expression after the URL in the xlink:href attribute, like this: xlink:href="https://dog.com/dogbreeds.xml#xpointer(id('Rottweiler'))". The expression refers to the element in the target document, with the id value of "Rottweiler".

XPointer also allows a shorthand method for linking to an element with an id. You can use the value of the id directly, like this: xlink:href="https://dog.com/dogbreeds.xml#Rottweiler".

The following XML document contains links to more information of the dog breed for each of my dogs:

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

<mydogs xmlns:xlink="http://www.w3.org/1999/xlink">

<mydog>
  <description>
  Anton is my favorite dog. He has won a lot of.....
  </description>
  <fact xlink:type="simple" xlink:href="https://dog.com/dogbreeds.xml#Rottweiler">
  Fact about Rottweiler
  </fact>
</mydog>

<mydog>
  <description>
  Pluto is the sweetest dog on earth.....
  </description>
  <fact xlink:type="simple" xlink:href="https://dog.com/dogbreeds.xml#FCRetriever">
  Fact about flat-coated Retriever
  </fact>
</mydog>

</mydogs>
```

POSSIBLE QUESTIONS

2 MARKS

1. What is XML as Data?
2. Define Primitive XML Data Types?
3. What is XML Namespaces?
4. What is a Uniform Resource Identifier (URI)?
5. What is XML Data Source Object?
6. Define Data islands?
7. What are the methods are used to create Data islands?
8. What is Links? Define HTML Links?
9. What is XLink?
10. What is XPointer?

8 MARKS

1. Explain in detail about Primitive XML Data Types?
2. What is XML Namespaces? Explain with an example.
3. What are the methods are used to create Data islands? Give example.
4. Enlighten the features of XLink with example.
5. Explain Xpointer with example.

KARPAGAM ACADEMY OF HIGHER EDUCATION

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

II B.Sc CS (Batch 2017-2020)

XML PROGRAMMING

PART - A OBJECTIVE TYPE/MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS

UNIT-IV

SNO	QUESTION	OPT1	OPT2	OPT3	OPT4	ANSWER
1	In Xpath the XML document is conceptually viewed as a _____	text file	method	event	tree	tree
2	Nodes in the Xpath tree have an ordering called _____	node order	document order	node index	list index	document order
3	Each node except the root node has a _____	parent node	child node	attribute	node number	parent node
4	_____ is the reverse ordering of the nodes in a document.	document order	reverse document	negative node order	negative node	reverse document
5	_____provides a syntax for locating specific parts of an XML	DOM	Schema	Xpath	XSL	Xpath
6	XPath is a _____ based language	object	event	structure	string	string
7	Which of these is not an Xpath node type	comment	entity	attribute	namespace	entity
8	Which if these node types cannot be a child node	element	processing instruction	text	namespace	namespace
9	_____ nodes has a parent node but they are considered as the child	element	processing instruction	text	namespace	namespace
10	Namespace nodes describes the namespace in which its _____ is	parent node	child node	root node	element node	parent node
11	Each Xpath tree nodes has _____	attribute value	string value	node value	child node	string value
12	The string value of the text node consists of the _____	normalised attribute	character data contined	text within the CDATA	normalized text	character data
13	The _____ nodes has a string value that consists of the normalized	text	element	attribute	comment	attribute
14	The string value of the _____ node is determined by concatinating	text	element	attribute	comment	element
15	Namespace nodes string value consists of _____	default namespace	a namespace prefix	URI of the namespace	URL of the	URI of the namespace

16	_____ consists of both the local part and a namespace URI.	Node name	Document order	Expanded name	String value	Expanded name
17	The Local part of the expanded name of a Processing Instruction node	target	value	node order	String value	target
18	The local part of the expanded name of a namespace node corresponds to	default namespace	namespace prefix	empty	namespaceURI	namespace prefix
19	The namespace URI of the expanded name of a namespace node is	default namespace	namespace prefix	always null	value	always null
20	_____ is an expression that specifies how to navigate an Xpath tree from	Document order	String value	Location path	Expanded name	Location path
21	Searching through an XML document begins at a _____	root node	context node	Location path	Expanded name	context node
22	_____ indicates which node should be included in the search relative to	Document order	String value	Node number	Axis	Axis
23	_____ dictates the ordering of the nodes in the set.	Document order	String value	Node number	Axis	Axis
24	Axes that select nodes that follow the context node in the document order	forward axes	backward axes	reverse axes	straight axes	forward axes
25	Axes that select nodes that precede the context node in the document	forward axes	diagonal axes	reverse axes	straight axes	reverse axes
26	The set of nodes selected is refined with _____	node types	node tests	Node number	Node	node tests
27	An axis has a _____ that corresponds to the type of the node	Xpath node type	Principle node type	String value	Expanded name	Principle node type
28	_____ is composed of a sequence of location steps.	Document order	String value	Location path	Expanded name	Location path
29	The location step consists of _____ seperated by double	axis and node	location path and axis	axis and node set	axis and node test	axis and node test
30	Node set _____ perform an action on a node-set returned by	operators	tests	functions	characters	functions
31	Node set _____ allow us to manipulate the node-set	operators	tests	functions	characters	operators
32	_____ operator performs the union of two nodesets.	| (pipe)	/ (slash)	// (double slash)	:: (double	| (pipe)
33	Node set funtion _____ returns the position number of the last node	count( )	last( )	Id( )	position( )	last( )
34	Node set funtion name(ns) returns the qualified name of the ___ in the	first node	node at current	last node	nodeset itself	first node
35	Extensible Stylesheet Language is used to format XML documents and	XSL, XSLT	XSL and Schema	XSLF,XSLT	XSLT and XSL	XSLT and XSL

36	Xpath expression is specified using _____	< >	[ ]	{ }	( )	{ }
37	_____ transforms XML document into other text based	Xpath	XSLT	XSLFormatting objects	Xlink	XSLT
38	Apache's Xalan is a _____	XSLT Processor	XML Parser	DTD validator	SAX Parser	XSLT Processor
39	The root element of the XSLT document is _____	ms:Schema	xsl:stylesheet	xmlns:transform	xsl:transform	xsl:stylesheet
40	_____ element matches specific XML document nodes by using an	stylesheet	transform	variable	template	template
41	XSLT uses the element variable with an attribute _____ to define a	type	value	order	name	name
42	In XSLT _____symbol is precedes the variable name to refer	@	$	#	?	$
43	_____XSLT element is used to output the value of an element or	output	value	value-of	contents	value-of
44	Element _____in XSLt duplicates all children	clone	duplicate	copy-of	copy	copy-of
45	copy element in XSLT is used to copy only _____ nodes	element	attrbute	text	context	context
46	XSLT allows for modularity in stylesheets through _____ element to	import	module	match	select	import
47	Local templates have _____ imported templates.	higher precedence	lower precedence	equal precedence	context based	higher precedence
48	Local templates have _____ included templates.	higher precedence	lower precedence	equal precedence	context based	equal precedence
49	_____ is used when the result of the transformation of for	XSLT	Formatting Objects	Structured document	Dom tree	Formatting Objects
50	Xlink elements that specifies the linking information are called _____	hyperlinks	linking elements	inline links	x-links	linking elements
51	Xlink attribute _____ is to specify how to display a resource when it is	show	view	display	actuate	show
52	Xlink attribute _____ is to specify when the resource should be	show	view	retrieve	actuate	actuate
53	If attribute show is assigned a value _____ the linked resourse replaces	new	replace	embed	other	embed
54	If attribute actuate is assigned a value onRequest the resourses is retrieved	as soon as it is loaded	when the user clicks	when the mouse	replace	when the user clicks
55	Value of the attribute _____ is used to identify the resourse	label	name	type	id	label

56	Xlink attribute type is assigned a value locator for _____	local	extended	simple	remote	remote
57	_____ is used to reference fragments of a document	Xlink	Xpath	Xpointer	Xbase	Xpointer
58	_____ provide a method for including XML documents within	Xinclude	Xpath	Xpointer	Xbase	Xinclude
59	Links between remote resources and local resourses are known as	simple	extended	inbound	inline	inbound
60	Xlink attribute _____ contains a human readable description of a link	name	title	description	label	title
61	An Xpath node is called a _____ in Xpointer	element	location	link	node	location
62	_____ allows the document author to change the base URI for any	Xinclude	Xpath	Xpointer	Xbase	Xbase
63	Xlink attribute _____ describes the relationship between resourses	arcrole	relate	locator	link	arcrole

UNIT V

SYLLABUS

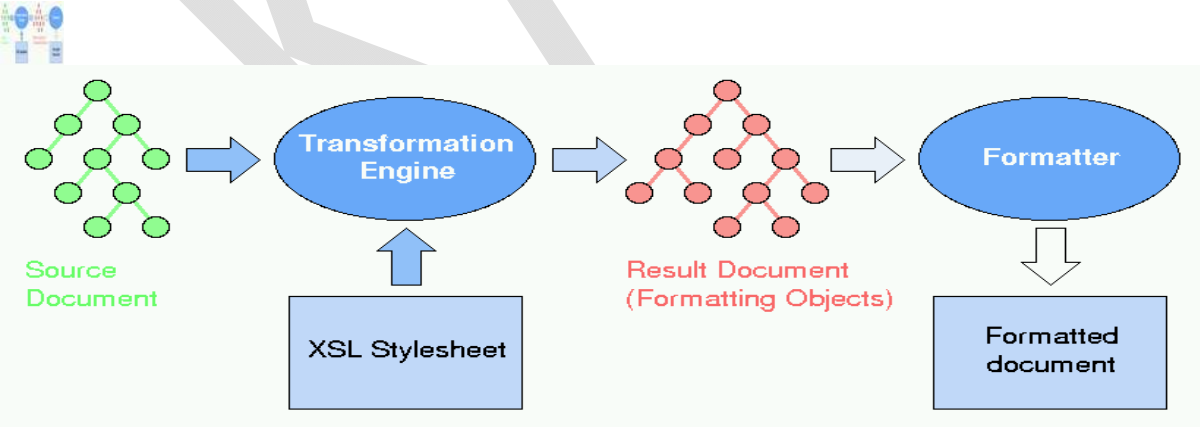
XML with Style: XSL –Style Sheet Basics, XSL basics, XSL style sheets.

Extensible Stylesheet Language (XSL)

XSL is a language for expressing stylesheets

- support for browsing, printing, and aural rendering
- formatting highly structured documents (XML)
- performing complex publishing tasks: tables of contents, indexes, reports,...
- addressing accessibility and internationalization issues
- written in XML

XSL Architecture



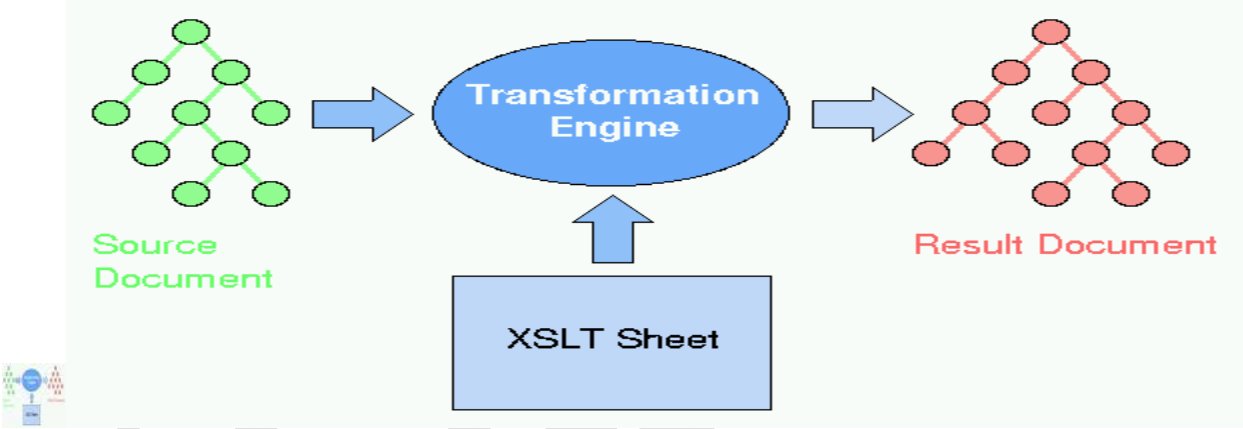
XSL Components

XSL is constituted of three main components:

- XSLT: a transformation language
- XPath: an expression language for addressing parts of XML documents
- FO: a vocabulary of *formatting objects* with their associated formatting properties

XSL uses XSLT which uses XPath

XSL Transformations



XSLT - Basic Principle

Patterns and Templates

- A style sheets describes transformation rules
- A transformation rule: a pattern + a template
- Pattern: a configuration in the source tree
- Template: a structure to be instantiated in the result tree
- When a pattern is matched in the source tree, the corresponding pattern is generated in the result tree

An Example: Transformation

```
<xsl:template match="Title">
<H1>
<xsl:apply-templates/>
</H1>
</xsl:template>
```

Input : <Title>Introduction</Title>

Output : <H1>Introduction</H1>

An Example: Formatting

```
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    result-ns="fo">
<xsl:template match="/">
<fo:page-sequence font-family="serif">
<xsl:apply-templates/>
</fo:page-sequence>
</xsl:template>
<xsl:template match="para">
<fo:block font-size="10pt" space-before="12pt">
<xsl:apply-templates/>
</fo:block>
</xsl:template>
```

```
</xsl:stylesheet>
```

XSL Usage

- Format XML documents by generating FOs
- Generate HTML or XHTML pages from XML data/documents
- Transform XML documents into other XML documents
- Generate some textual representation of an XML document
- ...and more

XSL may be used server-side or client-side, but is not intended to send FOs over the wire

XSLT Introduction

XSL (eXtensible Stylesheet Language) is a styling language for XML.

XSLT stands for XSL Transformations.

Online XSLT Editor

With our online editor, you can edit XML and XSLT code, and click on a button to view the result.

XSLT Example

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
```

```
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
      </tr>
    </xsl:for-each>
  </table>
 </body>
 </html>
</xsl:template>

</xsl:stylesheet>
```

XSL(T) Languages:

XSLT is a language for transforming XML documents.

XPath is a language for navigating in XML documents.

XQuery is a language for querying XML documents.

It Started with XSL

XSL stands for E**X**tensible **S**tyle sheet **L**anguage.

The World Wide Web Consortium (W3C) started to develop XSL because there was a need for an XML-based Style sheet Language.

CSS = Style Sheets for HTML

HTML uses predefined tags. The meaning of and how to display each tag is well understood.

CSS is used to add styles to HTML elements.

XSL = Style Sheets for XML

XML does not use predefined tags, and therefore the meaning of each tag is not well understood.

A <table> element could indicate an HTML table, a piece of furniture, or something else - and browsers do not know how to display it!

So, XSL describes how the XML elements should be displayed.

XSL - More Than a Style Sheet Language

XSL consists of four parts:

- XSLT - a language for transforming XML documents
- XPath - a language for navigating in XML documents
- XSL-FO - a language for formatting XML documents (discontinued in 2013)
- XQuery - a language for querying XML documents

What is XSLT?

- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation

XSLT = XSL Transformations

XSLT is the most important part of XSL.

XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element.

With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.

A common way to describe the transformation process is to say that **XSLT transforms an XML source-tree into an XML result-tree**.

XSLT Uses XPath

XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

How Does it Work?

In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document.

XSLT Browser Support

All major browsers support XSLT and XPath.

XSLT is a W3C Recommendation

XSLT became a W3C Recommendation 16. November 1999

XSLT <xsl:template> Element

An XSL style sheet consists of one or more set of rules that are called templates.

A template contains rules to apply when a specified node is matched.

The <xsl:template> Element

The <xsl:template> element is used to build templates.

The **match** attribute is used to associate a template with an XML element. The match attribute can also be used to define a template for the entire XML document. The value of the match attribute is an XPath expression (i.e. match="/" defines the whole document).

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:template match="/">
  <html>
  <body>
  <h2>MyCDCollection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <tr>
      <td>.</td>
      <td>.</td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Example Explained

Since an XSL style sheet is an XML document, it always begins with the XML declaration: **<?xml version="1.0" encoding="UTF-8"?>**.

The next element, **<xsl:stylesheet>**, defines that this document is an XSLT style sheet document (along with 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 to the output.

The last two lines define the end of the template and the end of the style sheet.

The result from this example was a little disappointing, because no data was copied from the XML document to the output. In the next chapter you will learn how to use the **<xsl:value-of>** element to select values from the XML elements.

XSLT <xsl:value-of> Element

The <xsl:value-of> element is used to extract the value of a selected node.

The <xsl:value-of> Element

The <xsl:value-of> element can be used to extract the value of an XML element and add it to the output stream of the transformation:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <tr>
      <td><xsl:value-of select="catalog/cd/title"/></td>
      <td><xsl:value-of select="catalog/cd/artist"/></td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Example Explained

Note: The **select** attribute, in the example above, contains an XPath expression. An XPath expression works like navigating a file system; a forward slash (/) selects subdirectories.

The result from the example above was a little disappointing; only one line of data was copied from the XML document to the output. In the next chapter you will learn how to use the **<xsl:for-each>** element to loop through the XML elements, and display all of the records.

XSLT <xsl:for-each> Element

The <xsl:for-each> element allows you to do looping in XSLT.

The <xsl:for-each> Element

The XSL <xsl:for-each> element can be used to select every XML element of a specified node-set:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
```

```
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Note: The value of the **select** attribute is an XPath expression. An XPath expression works like navigating a file system; where a forward slash (/) selects subdirectories.

Filtering the Output

We can also filter the output from the XML file by adding a criterion to the select attribute in the <xsl:for-each> element.

<xsl:for-each select="catalog/cd[artist='Bob Dylan']">

Legal filter operators are:

- = (equal)
- != (not equal)
- < less than
- > greater than

Take a look at the adjusted XSL style sheet:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
```

```
    <th>Title</th>
    <th>Artist</th>
   </tr>
   <xsl:for-each select="catalog/cd[artist='Bob Dylan']">
   <tr>
    <td><xsl:value-of select="title"/></td>
    <td><xsl:value-of select="artist"/></td>
   </tr>
   </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

XSLT <xsl:sort> Element

The <xsl:sort> element is used to sort the output.

Where to put the Sort Information

To sort the output, simply add an <xsl:sort> element inside the <xsl:for-each> element in the XSL file:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
```

```
    <th>Artist</th>
   </tr>
   <xsl:for-each select="catalog/cd">
    <xsl:sort select="artist"/>
    <tr>
     <td><xsl:value-of select="title"/></td>
     <td><xsl:value-of select="artist"/></td>
    </tr>
   </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Note: The **select** attribute indicates what XML element to sort on.

XSLT <xsl:if> Element

The <xsl:if> element is used to put a conditional test against the content of the XML file.

The <xsl:if> Element

To put a conditional if test against the content of the XML file, add an <xsl:if> element to the XSL document.

Syntax

```
<xsl:if test="expression">
  ...some output if the expression is true...
</xsl:if>
```

Where to Put the <xsl:if> Element

To add a conditional test, add the <xsl:if> element inside the <xsl:for-each> element in the XSL file:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
      <th>Price</th>
    </tr>
    <xsl:for-each select="catalog/cd">
      <xsl:if test="price &gt; 10">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
          <td><xsl:value-of select="price"/></td>
        </tr>
      </xsl:if>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Note: The value of the required **test** attribute contains the expression to be evaluated.

The code above will only output the title and artist elements of the CDs that has a price that is higher than 10.

XSLT <xsl:choose> Element

The <xsl:choose> element is used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests.

The <xsl:choose> Element

Syntax

```
<xsl:choose>
  <xsl:when test="expression">
    ... some output ...
  </xsl:when>
  <xsl:otherwise>
    ... some output ....
  </xsl:otherwise>
</xsl:choose>
```

Where to put the Choose Condition

To insert a multiple conditional test against the XML file, add the <xsl:choose>, <xsl:when>, and <xsl:otherwise> elements to the XSL file:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
```

```
    <td><xsl:value-of select="title"/></td>
    <xsl:choose>
      <xsl:when test="price &gt; 10">
        <td bgcolor="#ff00ff">
        <xsl:value-of select="artist"/></td>
      </xsl:when>
      <xsl:otherwise>
        <td><xsl:value-of select="artist"/></td>
      </xsl:otherwise>
    </xsl:choose>
  </tr>
  </xsl:for-each>
 </table>
 </body>
 </html>
</xsl:template>

</xsl:stylesheet>
```

The code above will add a pink background-color to the "Artist" column WHEN the price of the CD is higher than 10.

XSLT <xsl:apply-templates> Element

The <xsl:apply-templates> element applies a template to the current element or to the current element's child nodes.

The <xsl:apply-templates> Element

The <xsl:apply-templates> element applies a template to the current element or to the current element's child nodes.

If we add a select attribute to the <xsl:apply-templates> element it will process only the child element that matches the value of the attribute. We can use the select attribute to specify the order in which the child nodes are processed.

Look at the following XSL style sheet:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>

<xsl:template match="cd">
  <p>
  <xsl:apply-templates select="title"/>
  <xsl:apply-templates select="artist"/>
  </p>
</xsl:template>

<xsl:template match="title">
  Title: <span style="color:#ff0000">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>

<xsl:template match="artist">
  Artist: <span style="color:#00ff00">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>

</xsl:stylesheet>
```

Addressing Data with XSL Patterns:

What Is XSL Patterns?

XSL Patterns can be used as a general-purpose query notation for addressing and filtering the elements and text of XML documents. XSL Patterns are supported in XSL and in the Document Object Model.

XSL (eXtensible Stylesheet Language) has three broad functions relating to XML documents: **transforming**, **formatting** and **addressing**. Transforming can mean changing the XML to HTML, to another XML document, or perhaps to a totally different file format. Formatting means the capability to add, delete, or even reorder the content of your source document. Addressing lets us select certain parts of the XML document and ignore the rest. To retrieve the desired elements of an XML document, a query syntax called XSL Pattern Language was introduced.

XSL Patterns Language Syntax:

XSL includes a simple query facility called **XSL Patterns**. **Patterns** are an inherent part of the **XSL** transformation **language**, but can also be used in the XML DOM.

The XSL Pattern syntax provides methods that fall into two general groups: Information Methods and Collection Methods. The information methods provide information about particular nodes, such as the node type, node name, its text, or other information. The collection methods return collections of all nodes of a particular type, such as all comment nodes, all processing instruction nodes, and all element nodes.

Method names are case sensitive.

For example, the textnode method returns the text contained within a node, without any structure. (That is, it is the concatenation of all text nodes contained within an element and its descendants.)

textnode Method

Returns the collection of text nodes.

Syntax

textnode()

Remarks

This is an Extended XSL Pattern method described in the XQL Proposal .

Example

Find the second text node in each p element in the current context:

p/textnode()[1]

Examples

Find all text children of author elements:

author/text()

Find all elements that have text node or CDATA node children:

*[text()]

Find all author elements containing text nodes whose value is "Bob":

author[textnode() = "Bob"]

Information Methods

The XSL Pattern syntax supports methods to provide information about nodes in a collection. These methods return strings or numbers, and can be used with comparison operators in filter patterns.

Information Methods

date	Casts values to date format.
end	Returns true for the last element in a collection.
index	Returns the index number of the node within the parent.
nodeName	Returns the tag name of the node, including the namespace prefix.
nodeType	Returns a number to indicate the type of the node.

number Casts values to number format.

value Returns a typed version of the value of an element.

Remarks

If data types are provided, the value function uses them to determine the type for an element. For purposes of comparison, the lvalue is always cast to the type of the rvalue, thereby guaranteeing that the types will be constant through the comparison. Any lvalues that cannot be coerced are omitted from the result set.

Collection Methods

The XSL Pattern syntax supports collection methods to access the various types of nodes in a document. Any of these collections can be constrained and indexed. The collections return the set of children of the reference node meeting the particular restriction.

Collection Methods

attribute * Returns the collection of all attribute nodes. Synonymous with @*.

cdata * Returns the collection of CDATA nodes.

comment Returns the collection of comment nodes.

element * Returns the collection of all element nodes. Synonymous with *.

node * Returns the collection of all nodes except attributes and the root node of the document. Synonymous with "* | pi() | comment() | text()"

pi Returns the collection of processing instruction nodes.

text Returns the collection of nodes representing text strings. Synonymous with "textnode() | cdata()"

textnode * Returns the collection of text nodes.

Remarks

Note that the XML Document Object Model (DOM) defines a document object that contains comments, processing instructions, and declarations, as well as what is termed the "root element." The XML working drafts use the term "document entity" for the root of the DOM tree—the document object—instead of the "root element." This allows access to nodes at the document entity level, such as comments.

XML Schema

What is an XML Schema?

An XML Schema describes the structure of an XML document.

The XML Schema language is also referred to as XML Schema Definition (XSD).

XSD Example

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

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the elements and attributes that can appear in a document
- the number of (and order of) child elements

- data types for elements and attributes
- default and fixed values for elements and attributes

Why Learn XML Schema?

In the XML world, hundreds of standardized XML formats are in daily use.

Many of these XML standards are defined by XML Schemas.

XML Schema is an XML-based (and more powerful) alternative to DTD.

XML Schemas Support Data Types

One of the greatest strength of XML Schemas is the support for data types.

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

XML Schemas use XML Syntax

Another great strength about XML Schemas is that they are written in XML.

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schema with the XML DOM
- You can transform your Schema with XSLT

XML Schemas are extensible, because they are written in XML.

With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types
- Reference multiple schemas in the same document

XML Schemas Secure Data Communication

When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.

With XML Schemas, the sender can describe the data in a way that the receiver will understand.

A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.

However, an XML element with a data type like this:

<date type="date">2004-03-11</date>

ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

Well-Formed is Not Enough

A well-formed XML document is a document that conforms to the XML syntax rules, like:

- it must begin with the XML declaration
- it must have one unique root element
- start-tags must have matching end-tags
- elements are case sensitive
- all elements must be closed
- all elements must be properly nested
- all attribute values must be quoted

- entities must be used for special characters

Even if documents are well-formed they can still contain errors, and those errors can have serious consequences.

Think of the following situation: you order 5 gross of laser printers, instead of 5 laser printers. With XML Schemas, most of these errors can be caught by your validating software.

The Need for a Schema Language

The main difference between **XSD** and DTD is that **XSD** is written in **XML** and is considered easier to read and understand. Without **XML Schema** (**XSD** file) an **XML**file is a relatively free set of elements and attributes. The **XSD** file defines which elements and attributes are permitted and in which order.

XML Data Islands

[This feature was implemented for MSXML 3.0. Later versions of MSXML do not support it.]

When XML was first gaining acceptance, there was an increasing need to be able to embed "islands" of data inside HTML pages. In Microsoft® Internet Explorer 5.0 and later, these data islands can be written in XML.

The following topics describe the syntax used for embedding these data islands within a page, and detail the object model exposed by the browser to enable them to be used. This method of embedding XML in HTML follows the note published by the World Wide Web Consortium (W3C) as the "XML in HTML Meeting Report." The W3C expects to evolve the HTML specification to include the capability of embedding XML in HTML documents.

Embedding an XML Data Island into an HTML Page

An XML data island can be embedded using one of the following methods.

- Using the Dynamic HTML (DHTML) <XML> element within the HTML document
- Overloading the HTML <SCRIPT> element

Using the XML Element within the HTML Document

This syntax is valid for Internet Explorer 5.0.

There are two syntactically correct ways of using the <XML> element within the HTML document.

- The XML data can exist in line, surrounded by <XML></XML> start and end tags.

<XML ID="XMLID">

```
<XMLDATA>
<DATA>TEXT</DATA>
</XMLDATA>
</XML>
```

- The <XML> element can have a SRC attribute, the value of which is the URL for an XML data source.

```
<XML SRC="http://localhost/xmlFile.xml"></XML>
```

The <XML> element is present in the HTML Document Object Model. It is in the DHTML all collection and is seen by the browser as just a regular node. The XML data within the <XML> element can then be accessed by calling the DHTML XMLDocument property on the <XML> element.

The XMLDocument property returns the root node of the XML within the <XML> element or the root node of the XML referenced by the value of the SRC attribute. From this root, the XML data island can be navigated using the XML Document Object Model (DOM). The following JScript function returns the data from the data island with the ID of "XMLID".

```
function returnXMLData(){
 return document.all("XMLID").XMLDocument.nodeValue;
 }
```

The <XML> element can also be referenced by ID alone. For example, the following JScript function has the identical functionality as the preceding example.

```
function returnXMLData(){
 return XMLID.documentElement.text;
 }
```

Because the XMLDocument property was not used, the documentElement property must be called to retrieve the root element of the XML.

Overloading the HTML <SCRIPT> Element

This syntax has been deprecated and is intended only for down-level cases.

There are three syntactically correct ways of overloading the HTML <SCRIPT> element.

- The LANGUAGE attribute can be given the value "XML".

```
<SCRIPT LANGUAGE="XML">
```

- The TYPE attribute can be given the value "text/xml".

```
<SCRIPT TYPE="text/xml">
```

- As with the <XML> element, a SRC attribute can be added, the value of which is the URL for an XML data source.

<SCRIPT LANGUAGE="XML" SRC="http://localhost/xmlFile.xml"></SCRIPT>

The following HTML fragment illustrates how to embed data by overloading the <SCRIPT> element.

HTML

```
<SCRIPT ID="XMLID" LANGUAGE="XML">
<XMLDATA>
<DATA>TEXT</DATA>
</XMLDATA>
</SCRIPT>
```

The <SCRIPT> element is present in the HTML page's object model. (It is in the DHTML all collection and is seen by the browser as a regular script node.) The XML data within the <SCRIPT> elements can be accessed by calling the XMLDocument property on the <SCRIPT> object.

The following script accesses the XML data island in the preceding HTML fragment, and returns the name of the root node of the XML data island.

```
function returnIslandRootName(){
  var islandRoot = document.all.("SCRIPT").XMLDocument;
  return islandRoot.nodeName;
  }
```

Note
A tag that uses the name "XML" cannot be nested within an XML data island.

Vector Images with XML

Converting Scalable **Vector Graphics**. **Vector images** are typically saved in the Scalable **Vector Graphics** (SVG) format. This is an **XML**-based open standard developed by the W3C. Most **image** editing applications support exporting an **image**as an SVG file.

Scalable Vector Graphics (**SVG**) is an XML-based**vector** image **format** for two-dimensional **graphics**with support for interactivity and animation. The **SVG**specification is an open standard developed by the World Wide Web Consortium (W3C) since 1999. **SVG images** and their behaviors are defined in XML text**files**.

Scalable Vector Graphics (SVG)

SVG stands for Scalable Vector Graphics.

SVG defines vector-based graphics in XML format.

SVG Example

```
<html>
<body>

<h1>My first SVG</h1>

<svg width="100" height="100">
  <circle cx="50" cy="50" r="40" stroke="green" stroke-width="4" fill="yellow" />
</svg>

</body>
</html>
```

What is SVG?

- SVG stands for Scalable Vector Graphics
- SVG is used to define vector-based graphics for the Web
- SVG defines the graphics in XML format
- SVG graphics do NOT lose any quality if they are zoomed or resized
- Every element and every attribute in SVG files can be animated
- SVG is a W3C recommendation
- SVG integrates with other W3C standards such as the DOM and XSL

SVG is a W3C Recommendation

SVG 1.0 became a W3C Recommendation on 4 September 2001.
SVG 1.1 became a W3C Recommendation on 14 January 2003.
SVG 1.1 (Second Edition) became a W3C Recommendation on 16 August 2011.

SVG Advantages

Advantages of using SVG over other image formats (like JPEG and GIF) are:

- SVG images can be created and edited with any text editor
- SVG images can be searched, indexed, scripted, and compressed

- SVG images are scalable
- SVG images can be printed with high quality at any resolution
- SVG images are zoomable (and the image can be zoomed without degradation)
- SVG is an open standard
- SVG files are pure XML

The main competitor to SVG is Flash.
The biggest advantage SVG has over Flash is the compliance with other standards (e.g. XSL and the DOM). Flash relies on proprietary technology that is not open source.

Creating SVG Images

SVG images can be created with any text editor, but it is often more convenient to create SVG images with a drawing program, like Inkscape.

SVG Shapes

SVG has some predefined shape elements that can be used by developers:

- Rectangle <rect>
- Circle <circle>
- Ellipse <ellipse>
- Line <line>
- Polyline <polyline>
- Polygon <polygon>
- Path <path>

SVG Rectangle - <rect>

Example 1

The <rect> element is used to create a rectangle and variations of a rectangle shape:

Example

```
<svg width="400" height="110">
  <rect width="300" height="100" style="fill:rgb(0,0,255);stroke-width:3;stroke:rgb(0,0,0)" />
</svg>
```

Code explanation:

- The width and height attributes of the <rect> element define the height and the width of the rectangle
- The style attribute is used to define CSS properties for the rectangle
- The CSS fill property defines the fill color of the rectangle
- The CSS stroke-width property defines the width of the border of the rectangle
- The CSS stroke property defines the color of the border of the rectangle

Example 2

```
<svg width="400" height="180">
  <rect x="50" y="20" width="150" height="150"
  style="fill:blue;stroke:pink;stroke-width:5;fill-opacity:0.1;stroke-opacity:0.9" />
</svg>
```

Code explanation:

- The x attribute defines the left position of the rectangle (e.g. x="50" places the rectangle 50 px from the left margin)
- The y attribute defines the top position of the rectangle (e.g. y="20" places the rectangle 20 px from the top margin)
- The CSS fill-opacity property defines the opacity of the fill color (legal range: 0 to 1)
- The CSS stroke-opacity property defines the opacity of the stroke color (legal range: 0 to 1)

Example 3

Define the opacity for the whole element:

Example

```
<svg width="400" height="180">
  <rect x="50" y="20" width="150" height="150"
  style="fill:blue;stroke:pink;stroke-width:5;opacity:0.5" />
</svg>
```

Code explanation:

- The CSS opacity property defines the opacity value for the whole element (legal range: 0 to 1)

Example 4

Last example, create a rectangle with rounded corners:

Example

```
<svg width="400" height="180">
  <rect x="50" y="20" rx="20" ry="20" width="150" height="150"
  style="fill:red;stroke:black;stroke-width:5;opacity:0.5" />
</svg>
```

Code explanation:

- The rx and the ry attributes rounds the corners of the rectangle

Synchronized Multimedia Integration Language (SMIL)

Synchronized Multimedia Integration Language (SMIL (/ˈsmaɪl/)) is a World Wide Web Consortium recommended Extensible Markup Language (XML) markup language to

describe multimedia presentations. ... SMIL markup is written in XML, and has similarities to HTML.

What Is SMIL?

- SMIL stands for Synchronized Multimedia Integration Language
- SMIL is pronounced "smile"
- SMIL is a language for describing audiovisual presentations
- SMIL is easy to learn and understand
- SMIL is an HTML-like language
- SMIL is written in XML
- SMIL presentations can be written using a text-editor
- SMIL is a W3C recommendation

A Simplified SMIL Example

```
<smil>
<body>
<seq repeatCount="indefinite">
<img src="image1.jpg" dur="3s" />
<img src="image2.jpg" dur="3s" />
</seq>
</body>
</smil>
```

From the example above you can see that SMIL is an XML based, easy to understand, HTML-like language that can be written using a simple text-editor.

The **<smil></smil>** tags defines the SMIL document. A **<body>** element defines the body of the presentation. A **<seq>** element defines a sequence to display. The **repeatCount** attribute defines an indefinite loop. Each **<img>** element has a **src** attribute to define the image source and a **dur** attribute to define the duration of the display.

What Can SMIL Do?

- SMIL can be used to create Internet or Intranet presentations
- SMIL can be used to create slide-show presentations
- SMIL has been described as the Internet answer to PowerPoint
- SMIL presentations can display multiple file types (text, video, audio...)
- SMIL presentations can display multiple files at the same time
- SMIL presentations can display files from multiple web servers
- SMIL presentations can contain links to other SMIL presentations
- SMIL presentations can contain control buttons (stop, start, next, ...)
- SMIL has functions for defining sequences and duration of elements
- SMIL has functions for defining position and visibility of elements

SMIL is a W3C Recommendation

W3C has been developing SMIL since 1997, as a language for choreographing multimedia presentations where audio, video, text and graphics are combined in real-time.

SMIL became a W3C Recommendation 15. June 1998.

SMIL Files

A SMIL file contains all the information necessary to describe a multimedia presentation.

SMIL files are stored with the file extension *.smil

A SMIL file contains the following:

- The layout of the presentation

- The timeline of the presentation
- The source of the multimedia elements

SMIL Markup

Since SMIL is based on XML, the tags are case sensitive. All SMIL tags requires lowercase letters.

A SMIL document must start with a <smil> tag and end with a </smil> closing tag. It may contain a <head> element and must contain a <body> element.

The <head> element is used to store information about the presentation layout and other meta information.

The <body> element contains the media elements.

```
<smil>
<body>
<seq repeatCount="indefinite">
<img src="image1.jpg" dur="3s" />
<img src="image2.jpg" dur="3s" />
</seq>
</body>
</smil>
```

Document Content Description (DCD):

The **Document Content Description** facility for **XML** (abbreviated DCD) is an RDF vocabulary designed for describing constraints to be applied to the structure and**content** of **XML documents**.

Design Principles

DCD is based on the following design principles:

1. DCD semantics shall be a superset of those provided by XML DTDs.
2. The DCD data model and syntax shall be conformant with that of RDF.
3. The constraints in a DCD shall be straightforwardly usable by authoring tools and other applications which wish to retrieve information about a document's content and structure.
4. DCD shall use mechanisms from other W3C working groups wherever they are appropriate and efficient.
5. DCDs should be human-readable and reasonably clear.

XML RDF

What is RDF?

- RDF stands for **R**esource **D**escription **F**ramework
- RDF is a framework for describing resources on the web
- RDF is designed to be read and understood by computers
- RDF is not designed for being displayed to people
- RDF is written in XML
- RDF is a part of the W3C's Semantic Web Activity
- RDF is a W3C Recommendation from 10. February 2004

RDF - Examples of Use

- Describing properties for shopping items, such as price and availability
- Describing time schedules for web events
- Describing information about web pages (content, author, created and modified date)
- Describing content and rating for web pictures
- Describing content for search engines
- Describing electronic libraries

RDF is Designed to be Read by Computers

RDF was designed to provide a common way to describe information so it can be read and understood by computer applications.

RDF descriptions are not designed to be displayed on the web.

RDF is Written in XML

RDF documents are written in XML. The XML language used by RDF is called RDF/XML.

By using XML, RDF information can easily be exchanged between different types of computers using different types of operating systems and application languages.

RDF and "The Semantic Web"

The RDF language is a part of the W3C's Semantic Web Activity. W3C's "Semantic Web Vision" is a future where:

- Web information has exact meaning
- Web information can be understood and processed by computers
- Computers can integrate information from the web

RDF uses Web identifiers (URIs) to identify resources.

RDF describes resources with properties and property values.

RDF Resource, Property, and Property Value

RDF identifies things using Web identifiers (URIs), and describes resources with properties and property values.

Explanation of Resource, Property, and Property value:

- A **Resource** is anything that can have a URI, such as "https://www.w3schools.com/rdf"
- A **Property** is a Resource that has a name, such as "author" or "homepage"
- A **Property value** is the value of a Property, such as "Jan Egil Refsnes" or "https://www.w3schools.com" (note that a property value can be another resource)

The following RDF document could describe the resource "https://www.w3schools.com/rdf":

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

<RDF>
  <Description about="https://www.w3schools.com/rdf">
    <author>Jan Egil Refsnes</author>
    <homepage>https://www.w3schools.com</homepage>
  </Description>
</RDF>
```

The example above is simplified. Namespaces are omitted.

RDF Statements

The combination of a Resource, a Property, and a Property value forms a **Statement** (known as the **subject, predicate and object** of a Statement).

Let's look at some example statements to get a better understanding:

Statement: "The author of https://www.w3schools.com/rdf is Jan Egil Refsnes".

- The subject of the statement above is: https://www.w3schools.com/rdf
- The predicate is: author
- The object is: Jan Egil Refsnes

Statement: "The homepage of https://www.w3schools.com/rdf is https://www.w3schools.com".

- The subject of the statement above is: https://www.w3schools.com/rdf
- The predicate is: homepage
- The object is: https://www.w3schools.com

RDF Example

Here are two records from a CD-list:

Title	Artist	Country	Company	Price	Year
Empire Burlesque	Bob Dylan	USA	Columbia	10.90	1985
Hide your heart	Bonnie Tyler	UK	CBS Records	9.90	1988

Below is a few lines from an RDF document:

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

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">

<rdf:Description
rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
  <cd:artist>Bob Dylan</cd:artist>
  <cd:country>USA</cd:country>
  <cd:company>Columbia</cd:company>
  <cd:price>10.90</cd:price>
  <cd:year>1985</cd:year>
</rdf:Description>

<rdf:Description
rdf:about="http://www.recshop.fake/cd/Hide your heart">
  <cd:artist>Bonnie Tyler</cd:artist>
  <cd:country>UK</cd:country>
  <cd:company>CBS Records</cd:company>
  <cd:price>9.90</cd:price>
  <cd:year>1988</cd:year>
</rdf:Description>
.
.
.
</rdf:RDF>
```

The first line of the RDF document is the XML declaration. The XML declaration is followed by the root element of RDF documents: **<rdf:RDF>**.

The **xmlns:rdf** namespace, specifies that elements with the rdf prefix are from the namespace "http://www.w3.org/1999/02/22-rdf-syntax-ns#".

The **xmlns:cd** namespace, specifies that elements with the cd prefix are from the namespace "http://www.recshop.fake/cd#".

The **<rdf:Description>** element contains the description of the resource identified by the **rdf:about** attribute.

The elements: **<cd:artist>, <cd:country>, <cd:company>,** etc. are properties of the resource.

Cross-Platform XML

XML is a cross-platform, software and hardware independent tool for transmitting information.

"XMl is going to be everywhere" – W3C

XML DOM - The NodeList Object

The NodeList object represents an ordered list of nodes.

The NodeList object

The nodes in the node list can be accessed through their index number (starting from 0).

The node list keeps itself up-to-date. If an element is deleted or added, in the node list or the XML document, the list is automatically updated.

Note: In a node list, the nodes are returned in the order in which they are specified in the XML document.

NodeList Object Properties

Property	Description
length	Returns the number of nodes in a node list

NodeList Object Methods

Method	Description
item()	Returns the node at the specified index in a node list

XML DOM - The Node Object

The Node Object

The Node object represents a single node in the document tree.

A node can be an element node, an attribute node, a text node, or any other of the node types

Notice that while all objects inherits the Node properties / methods for dealing with parents and children, not all objects can have parents or children. For example, Text nodes may not have children and adding children to such nodes results in a DOM error.

Node Object Properties

Property	Description
attributes	A NamedNodeMap containing the attributes of this node (if it is an Element)
baseURI	Returns the absolute base URI of a node
childNodes	Returns a NodeList of child nodes for a node
firstChild	Returns the first child of a node
lastChild	Returns the last child of a node
nextSibling	Returns the node immediately following a node
nodeName	Returns the name of a node,

	depending on its type
nodeType	Returns the type of a node
nodeValue	Sets or returns the value of a node, depending on its type
ownerDocument	Returns the root element (document object) for a node
parentNode	Returns the parent node of a node
prefix	Sets or returns the namespace prefix of a node
previousSibling	Returns the node immediately before a node
textContent	Sets or returns the textual content of a node and its descendants

Node Object Methods

Method	Description
appendChild()	Appends a new child node to the end of the list of children of a node
cloneNode()	Clones a node

compareDocumentPosition()	Compares the placement of two nodes in the DOM hierarchy (document)
getFeature(feature,version)	Returns a DOM object which implements the specialized APIs of the specified feature and version
getUserData(key)	Returns the object associated to a key on a this node. The object must first have been set to this node by calling setUserData with the same key
hasAttributes()	Returns *true* if the specified node has any attributes, otherwise *false*
hasChildNodes()	Returns *true* if the specified node has any child nodes, otherwise *false*
insertBefore()	Inserts a new child node before an existing child node
isDefaultNamespace(URI)	Returns whether the specified namespaceURI is the default
isEqualNode()	Tests whether two nodes are equal
isSameNode()	Tests whether the two nodes are the same node

lookupNamespaceURI()	Returns the namespace URI associated with a given prefix
lookupPrefix()	Returns the prefix associated with a given namespace URI
normalize()	Puts all Text nodes underneath a node (including attribute nodes) into a "normal" form where only structure (e.g., elements, comments, processing instructions, CDATA sections, and entity references) separates Text nodes, i.e., there are neither adjacent Text nodes nor empty Text nodes
removeChild()	Removes a specified child node from the current node
replaceChild()	Replaces a child node with a new node
setUserData(key,data,handler)	Associates an object to a key on a node

XML DOM PARSE ERROR OBJECT

To retrieve error information from the microsoft XML parser use Microsoft's parseError object.

The parseError Object

The parser-error may occur, when you trying to open an XML document.

You can retrieve the error code, the error text, the line that caused the error, with the parse error object.

Note: The parseError object is not a part of the W3C DOM standard

File Error

```
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async=false;
xmlDoc.load("unknown.xml");

document.write(" The Error code: " + xmlDoc.parseError.errorCode);
document.write("<br />The Error reason: " + xmlDoc.parseError.reason);
document.write("<br />The Error line: " + xmlDoc.parseError.line);
```

Note: In the following code trying to load non-existing file that is why it's gives the error property

XML Error

```
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async=false;
xmlDoc.load("Sevral_error.xml");

document.write(" The Error code: " + xmlDoc.parseError.errorCode);
document.write("<br />The Error reason: " + xmlDoc.parseError.reason);
document.write("<br />The Error line: " + xmlDoc.parseError.line);
```

Note: In the following code the parser load an XML document that is not well-formed. And it gives the error

The parseError Object's Properties

Property	Description

Errorcode	It returns the long integer error code
Reason	It returns the string value that contain the reason of error
Line	It returns the integer that represent no of the error
Linepos	It returns the integer that represent the line position for the error
Srctext	It returns the string that containg the line which have the error
Url	It returns the url of loaded document
Filepos	It returns the integers that contain the file position that caused error

POSSIBLE QUESTIONS

2 MARKS

1. What is XSL?

2. Sketch the XSL Architecture diagram?

3. Define XSL Components?

4. Illustrate the XSL Transformations diagram?

5. Write a small snippet of Transformation of XSL?

6. Write a small snippet of Formatting of XSL?

7. Identify the XSL Usage?

8. What are the XSL(T) Languages?

9. What is XSLT?

10. What is XSL Patterns?

11. Define textnode Method?

12. Write a short note on Information Methods?

13. Write a short note on Collection Methods?

14. What is an XML Schema?

15. Engrave the Need for a Schema Language?

16. How to Embedding an XML Data Island into an HTML Page?

17. Define Vector Images with XML?

18. Define Scalable Vector Graphics (SVG)?

19. List the SVG Advantages?

20. What Is SMIL?

21. What is Document Content Description (DCD)?

22. List the Design Principles of DCD?

23. What is RDF?

24. Is XMLCross-Platform Language?

25. Define the NodeList object?

26. Define the Node Object?

27. Define the parseError Object?

8 MARKS

1. Explain in detail about XSLT with example.
2. Explain the <xsl:template> Element with example.
3. Explain XSLT <xsl:value-of> Element with example.
4. Explain XSLT <xsl:for-each> Element with example.
5. Explain XSLT <xsl:sort> Element with example.
6. Explain XSLT <xsl:if> Element with example.
7. Explain XSLT <xsl:choose> Element with example.
8. Explain XSLT <xsl:apply-templates> Element with example.
9. Demonstrate Addressing Data with XSL Patterns.
10. Explain XML Schema with example.
11. Explain in detail about XML Data Islands.
12. Explain Scalable Vector Graphics (SVG).
13. Give explanation of SVG Shapes.
14. Explain Synchronized Multimedia Integration Language (SMIL)
15. Explain the following XML DOM object with example.
    i) The NodeList Object
    ii) Node Object
    iii) parseError Object

KARPAGAM ACADEMY OF HIGHER EDUCATION

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

II B.Sc CS (Batch 2017-2020)

XML PROGRAMMING

PART - A OBJECTIVE TYPE/MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS

UNIT-V

Sno	Questions	Opt1	Opt2	Opt3	Opt4	Answer
1	Which XML technology is used for searching and retrieving data from the	XML Query Language	XML Topic Maps	XML Metadata	Virtual HyperGloss	XML Query
2	P3P stands for _____	point to point	platform for privacy	Information and	Privacy Protocol	platform for privacy
3	Directories gain the ability to handle distributed Web applications like e-	RSS	XML Topic Maps	DSML	ICE Protocol	DSML
4	information about information is called _____	metadata	information description	data description	markup data	metadata
5	_____ is XML based language for describing information contained in	XML Topic Maps	XML Query Language	Resource Definition	Channel Definition	Resource Definition
6	BXXP is an acronym of	Block Extensible	Block Express Exchange	Block Exchange	Blocks Extensible	Blocks Extensible
7	Search Engines use _____ to list or catalog inforamtion on the	Virtual HyperGlossa	RDF's metadata	XML Query Language	XML Topic Maps	RDF's metadata
8	Channel Definition Format is an XML application that uses _____	push	pull	broadcast	packet switching	push
9	A CDF document has root element _____	CDF	format	channel	any	channel
10	_____ is an alternative to HTTP for transfering data over the internet.	FTP	P3P	ICE	BXXP	BXXP
11	RSS stands for	Religious Social	Rich Site Summary	Region Survey	Resource Definition	Rich Site Summary
12	_____ implements push technology	Rich Site Summary	XML Topic Maps	XML Query Language	Resource Definition	Rich Site Summary
13	In RDF documents the RDF elements uses the namespace prefix _____	rdf	xmlns	dc	XML Topic Maps	rdf
14	In RDF documents the metadata elements uses the namespace prefix	rdf	xmlns	dc	XML Topic Maps	dc
15	Element _____ in RDF is used to represent an ordered list of resources.	Ord	Bag	Seq	Res	Seq
16	Element _____ in RDF is a container of unordered list of	unOrd	Bag	Seq	Res	Bag
17	_____ provides the document with a terminological environment	Virtual HyperGlossa	RDF's metadata	XML Query Language	XML Topic Maps	Virtual HyperGlos

18	_____ specifies a framework of hyperlinked glossaries, dictionaries and	RSS	CDF	VHG	XTM	VHG
19	Communication with the ICE protocol is based on	push technology	pull technology	request/response	packet switching	request/response
20	ICE is a _____	protocol	language	edible	metadata	protocol
21	_____ governs the interaction between the media syndicator and a	Ecom	ICE	P3P	BXXP	ICE
22	Information and Content Exchange Protocol is abbrevated as _____	(TCP) IP	BXXP	IXP	ICE Protocol	ICE Protocol
23	_____ is a security technology in XML	Public Key Cryptograph	Encryption	Hashing Technology	XML Signature	XML Signature
24	The _____ describes all software modeling environment	Software Project	Meta Object Facility	Document Object	Software Modeling	Meta Object
25	The ____ provides a rigorous definition of Object Oriented	XMI	MOF	UML	OMG	MOF
26	UML and MOF are integrated with XML to create _____	UMI	OMG	XMI	DOM	XMI
27	_____ is an OMG standard for sharing and storing object oriented	XML Metadata	Meta Object Facility	Document Object	CDF	XML Metadata
28	_____ enables programmers to collaborate and create compatible	XML Metadata	Meta Object Facility	Document Object	OMG	XML Metadata
29	User must possess a _____ licence to access the e-content	XMI	XrML	XML signature	CDF	XrML
30	_____ provides a framework for legal agreements, certificationetc.	XMI	XrML	XML signature	CDF	XrML
31	XrML stands for Extensible rights Markup Language	Extensible rights	Extensible Markup	XML Digital Signature	XML licence	Extensible rights
32	The basic units of XML topic maps is _____	blocks	metadata	topic	element	topic
33	Each occurance of a topic has a role and _____ that provides information	name	type	scope	identity	type
34	Associations of topic map are represented by _____ element	associate	association	assoc	occurs	assoc
35	Occurance of a topic are described by _____ element	occurs	assoc	topic	basename	occurs
36	_____ gives the topic a name to be used by applications	topname	topicname	basename	dispname	basename
37	The root element of topic map is _____	topic	topicmap	topname	tmproc	topicmap
38	The root element of RDFdocument is _____	rdf:RDF	rdf:Root	rdf:Description	RDF	rdf:RDF
39	In _____ technology contents are sent automatically to the user.	push technology	pull technology	request/response	packet switching	push technolog
40	CDF stands for	Common Document	Channel Delivery	Common Delivery	Channel Definition	Channel Definition