Webdev Intro : 1
HTML tags : 2
Void elements : 3
List element : 3
Anchor tag : 3
Global attributes : 3
<img>, Comments : 4
HTML5 : 4
File paths : 4
Block vs Inline : 5
HTML entities : 6
dev.w3.org (charref)
Semantics div : 5, 6
Table : 6
forms : 7
Inputs of forms : 7
validations : 8
Detailed HTML : 8
Bookmarks : 10
Image map : 11
Background Image : 11
picture element : 11
Tables : 12
Table padding : 13
List : 14
div : 15
class : 15
ID : 15
head : 16
Responsive : 17
Computer code : 17
Semantics : 17
HTML guide : 18
XHTML : 19
Forms : 20
Graphics : 21
APIS : 22 - 24
XML : 24
XML tree, rules : 25
attributes, prefix : 26
namespace : 27
XSLT, xmlns : 27
XML parser : 28
XML Dom : 28
Xpath : 28
Xquery : 29
XLink : 30
Xpointer : 30
Validation : 30
DTD, Schema : 31
AJAX, XML : 32
XML Dom : 33
sibling, child : 34
Skip empty : 35
edit, add node : 36
Xpath : 37
Xpath syntax : 38
axes : 39
XSLT : 39
template : 40
for-each, value of, filtering : 41
Sort : 41
xsl:if : 41
xsl:when : 41
xsl:apply-templates : 42
XSLT : client : 42
Xquery : 43
Predicate : 44
FLWOR : 44
FLWOR + HTML : 44
Comparison : 45
functions : 46
DTD : 46
elements : 47
attributes : 47
Entities : 48
XSD : 49
Schema, ref in xml : 50
Simple elements : 50
attributes : 50
Restr, facets : 51
whitespace : 51
length restr : 51
empty : 53
ComplexType : 52
elem only : 52
Text only : 53
mixed : 53
Indicators : 53
<any> : 53
Substitute : 54
Datatype : 54
String : 54
Date & time : 54
webservices : 55
HTML CORS : 57
CSS : 58

XML (W3C recommended: 1998)

* Extensible markup language: store & transport data (human & machine-readable)
* markup language like HTML: self descriptive (W3C recommendation)
* XML: does nothing!

```
<note>
  <to> Tove </to>
  <from> John </from>
  <heading> —— </heading>
</note>
```

"Info wrapped in tags"

XML: carry data (no predefined tags)
HTML: Display data

* No predefined tags

Extensible:

* Most XML app will work: even if data added/removed

XML:

* Simplify data sharing, transport
* platform changes
* Availability.

XML: plaintext (used to expand/update: OS/app/browsers)

Use of XML:

* Separate data from presentation
* Complement to HTML (with JS)
* eg: data of shares, medicine, science, news, weather, ..

```
<?xml version="1.0" encoding="UTF-8" ?>
<current observation>
    credit
    creditURL
    image
        |→ url, title, link
    location, station, lat, long, time
    weather, temp_f, temp_c
    humidity, wind...
</current observation>
```

→ XML prolog (optional, if exists: must be 1st line)

XML Tree

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

Sibling: Same level
parent, child: 1 level above, below resp.

Self describing syntax:

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

Root element is must!

* eg: `<note>`, `<current observation>`

Rules:

* Root element is must
* closing tag is must
* Tags: case sensitive
* proper nesting is must
* attribute must be quoted `<note date="12/11/1999">`
* '<': special means → use unicode value (<)
* Comment: `<!-- -->`
* white space is preserved in XML
* XML stores new line as LF (windows: CR+LF, mac: LF, old mac: CR)
* Empty: `<book></book>` (or) `<book/>`

XML element:

* everything from start to end tag!
* element may has text, attributes, other elements, mix of them!

Naming rules: (elements)

* Case sensitive
* Start with letter or underscore
* Can't start with 'xml' / 'XML'
* Can have letters, digits, hyphens, underscore, periods
* No spaces!

> 'No reserved words : except "xml"

Practice:

* descriptive names
* avoid '_' : Sub
* avoid '.' : Property of object
* Colon : reserved for namespaces
* non-english letters supported (ensure the software does!)

<firstname> : lower
<FIRSTNAME> : upper
<first_name> : underscore
<FirstName> : Pascal
<firstName> : Camel

> 'xml: extend without breaking!'

Attributes:

* data related to the specific element

<person gender = 'male'>

<person name = 'George "Shotgun" ziegler'>

Has ', hence using "

or use &quot!

> no rules about attribute & element
> gender: attribute, can also be an element!

Avoid attributes?

* attributes can't have multiple values (elements can)
* attributes: no tree
* Not easily expandable.

'use for metadata'

<note id = '501'>

Name conflict:

* XML: table: name, width, length
* XML has html: table > tr: td, td

> 'name element: diff content'

Use name prefix (avoid conflict):

<h:table>, <h:tr>, <h:td>
<f:table>, <f:name>, <f:width>, <f:length>

XML namespaces: xmlns attribute:

* when prefix are used: define namespace (xmlns)
* xmlns:prefix: URI

<h:table xmlns:h="http://____">. VRI endpoint resource!

> 'only for parent/root: child with prefix will have the same namespace'

default namespace (no need to define namespace for child)

<table xmlns="uri"> ⟶ tr ⟶ td, td

Why namespace:

* XSLT: Transform XML to other format
* namespace "http://www.w3.org/1999/xsl/Transform"
  ↳ Identifies XSLT inside HTML document!

> Refer w3.org

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://____/Transform">
<xsl:template match="/">
<html>
<body>
  <h2>____</h2>
  <table border="1">
    tr: th, th
    <xsl:for-each select="catalog/cd">
    tr ⟶ <td><xsl:value-of select="title"/></td>
                          "artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
</body>
</xsl:template>
</xsl:stylesheet>
```

Display XML: (browser)

* XML (Browser) doesn't know content (whether HTML) : Just display tree!

Browser XML HttpRequest object:

* Update without reloading the pag
* Request data, receive / send data to the server

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function () {
    if (this.readyState == 4 && this.status == 200) {
        // Change inner html
    }
};
xhttp.open ('GET', 'filename', true);
xhttp.send ();
```

→ everytime when status changes!

XML parser (browser: Inbuilt):

* XML Dom object is loaded

`xmlhttp.responseXML` : Inbuilt parser of XMLHttpRequest!

```
parser = new DomParser();
xmlDoc = parse.parseFromString
    (xmlStr, 'text/xml');
xmlDoc.getElementByTagName
    ('title')[0].
    childNodes[0].
    nodeValue;
```

XML Dom:

* Document Object Model (tree-structure)
* document.getElementById ('demo');
  ↓
  HTML
* document.getElementsByTagName ('title')[0].childNodes[0].nodeValue;

Xpath:

* Syntax for defining parts of XML document
* Xpath: uses path expressions to navigate in XML doc
* Xpath: has lib of Std functions
* Xpath is a major element in XSLT & Xquery
* Xpath: W3C recommendation.

XML: bookstore ⟶ book (category) > title, author, year, price

/bookstore/book[1] : 1st book

/bookstore/book[last()] : last book

/bookstore/book[last()-1] :

/bookstore/book[position() < 3] : 1 st 2 books

//title[@lang] ⟶ elements (title) with attribute lang!

→ //title [@lang = 'en'] ⟶ title with lang = 'en'

→ /bookstore/book [price > 35.00]

→ /bookstore/book [price > 35.00]/title : Select title of all books with price > 35.00

xml with XSLT:

* XSLT: Transform xml to html

xml: breakfast_menu > food > name, price, description, calories

XSLT stylesheet:

```
<? xml version = '1.0' Encoding = 'UTF-8' ? >
<html xsl : version = '1.0' xlmns: xsl = 'http://www.w3.org/1999/xSL/
                                                    Transform' >

<body style = ——————>
<xsl : for-each  Select = 'breakfast_menu / food' >
    <div style ——————>
        <span ——— > <xsl: value-of Select = 'name' /> - </span>
        <xsl: value-of Select = 'price' >
    </div>
    <p>
      <xsl: value-of Select = 'description' />
    ——————
  ———
<xsl : for-each>
</body>
</html>
```

XQuery:

* XQuery is what Sql to DB!
* query xml data - built on XPath
* Supported by all major databases

Select all cD records price < 10 from cd Collection (cdcatalog.xml)

Use in app, generate report, xml ⟶ XHTML

Xml, Xlink, XPointer:

* Create hyperlinks in xml doc.
* any element: behave link link (define outside the linked files)
* W3C recommendation.

Xlink (no direct browser support: xlinks in SVG)

```
<?xml version = '1.0' encoding = 'UTF-8' ? >
<homepages xmlns: xlink = 'http://www.w3.org../xlink' >
    <homepage xlink : type = 'simple' xlink:href = '____' > visit w3 </homepage>
```

*Declare XLink namespace (root), type = "simple" (html-like)
xlink:show = "new" (new window)

xlink:actuate → onload, onrequest, other, none
∴ when linked resource is read & shown (loading / clicked link)

xlink:href → URL

xlink:show → embed, new, replace (default), other, none
"where to open"

xlink:type → simple, extended, locator, arc, resource, title, none

XPointer:

* allow link to point to specific parts of an XML doc
* xpointer uses XPath to navigate
* W3 recommendation!

"No browser support"

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

dogbreed
    dog: Rottweiler (id)
        picture, history, temperament
```

Other doc:

```
<mydogs xmlns:xlink="http://www.w3.org/1999/xlink">
  mydog
    description
    fact xlink:type="simple" xlink:href="https://dog.com/dogbreeds.
                                          xml#Rottweiler">
```

XML validation:

* errors: Stop! (root, closing tag, case sensitive, proper nesting, quoted)
* document type definition → DTD : orginal doc type def
  → XML Schema : XML alternative to DTD

DTD:

```
<!DOCTYPE note
[
  <!ELEMENT note (to, from, heading, body)>
  <!ELEMENT to (#PCDATA)>
]>
```

* !DOCTYPE : root element note
  note
* !ELEMENT note : note must have (to, from, heading, body)
* !ELEMENT to : type : #PCDATA (parseable char data)

* DOCTYPE : use to define special char / string used in the doc

When to use DTD:

* Independent people can agree to use a std DTD : data interchange
* validate Doc data

When not to:

* XML : no need for a DTD (experiment / small files : waste of time)

DTD : possible error ! (validation is not satisfied)

XML Schema:

* describe XML Structure (validated for well formed + valid)

```
<xs:element name="note">  ——→ note
  <xs:complexType>  ——→ complexType
    <xs:sequence>  ——→ sequence of elements
      <xs:element name="to" type="xs:string"/>
                  from
                  body
                  heading
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Why use

* easy describe
* define restrict
* validation
* data conversion

Powerful → written in XML
→ extensible (add / edit)
→ support data types
→ namespaces

XML Syntax : no need to learn !
XML editor : write / edit
XML parser, XML Dom
Transform with XSLT

AJAX & XML:

* Update page (no reload)
* Request data / receive data: after page load
* Send data in the background

eg:

```
<button onclick="loadDoc()"> —— </button>

function loadDoc(){
    var xhttp = new XmlHttpRequest();
    xhttp.onreadystatechange = function () {
        if (this.readystate == 4 && this.status == 200){
            innerhtml ("demo") = this.responseText; }}
    xhttp.open ("GET", "ajax.txt", true);
    xhttp.send();
}
```

AJAX:

* Not a programming language! (Async Javascript And XML)
* Combination of builtin XmlHttpRequest + HTML & JS Dom!
* AJAX app may transport text / Json also (name: misleading)

working:

Browser
event occurs
* create XmlHttpRequest object
* send HttpRequest

→ Internet → **Server**
* process
* response : data

→ Internet → **Browser**
* process (JS)
* update page!

eg: button click → send req → Receive data → update page!

XmlHttpObject:

variable = new XmlHttpRequest();

Security:

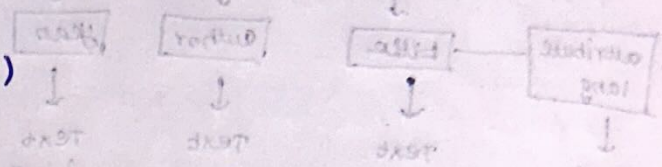
* webpage, XML file: must be from the same domain!
* modern browsers: won't allow access across domains.

methods

* new XmlHttpRequest()
* abort() : cancel current req
* getAllResponseHeaders()
* getResponseHeader("")
* open (method, url, async, user, psw) → username, password (optional)
* send(), send(String) → Post
* setRequestHeader()

Properties:

* onReadyStateChange : define function to be called when state change
* readState (0: not init, 1: server conn est
  2: req received, 3: processing req
  4: req finished & response is ready)
* responseText, responseXml
* Status (200, 403, 404)
* StatusText ("OK", "Not Found")



AJAX request:

* open, send : get, post → async (true)
  → async (false)

Callback function:

when status == 4 → call a function!

Example: Php, ASP, DB (data) → use AJAX replace html (constructs)!

Dom

* Standard: accessing & manupulating documents (platform & language neutral interface)
* Html/Xml Dom: tree structure

  Html: getElementById ("demo")

  Xml: getElementsByTagName ("h1")[0]

Xml Dom:

→ var xmlDoc = xml.responseXml;

→ xmlDoc.getElementsByTagName ("title")[0].childNodes[0].nodeValue

→ xmlDoc = new DomParser().parseString(text, "text/xml");

Dom properties:

* x.nodeName
* x.nodeValue
* x.parentNode
* x.childNodes
* x.attributes

Xml Dom methods

x.getElementsByTagName (name)
x.appendChild (node)
x.removeChild (node)

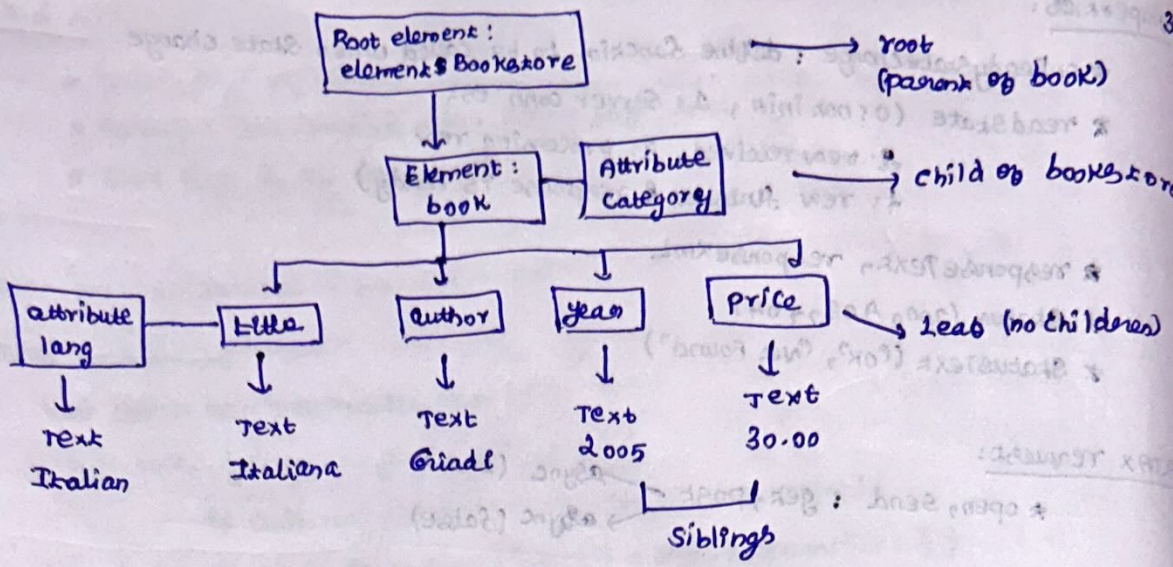
Node:

* Everything is a node (element/text/attribute/comment node)

Bookstore

↳ book (Category)
  ↳ title
  ↳ author
  ↳ year → <year> holds text node "2005" (Hence text node is the value, not "2005"
  ↳ price

Dom:



3rd book title:

→ xmlDoc.getElementsByTagName('title')[2] : Index starts at 0

- documentElement (root node)
- nodeName, nodeType

Properties: (nodeName, nodeValue, nodeType)

* nodeName: readonly (same as tag name)
* nodeName of an attribute node: attribute name
* nodeName of a text node: text!
* nodeName of the document node is always #document.

Nodevalue:

* elementNode: undefined
* text nodes: text
* attribute node: attribute value → can change value (unlike nodeName)

element	1
attribute	2
Text	3
Comment	8
Document	9

→ xmlDoc.getElementsByTagName('title').length

→ xmlDoc.getElementsByTagName('book')[0].attributes

Traverse:

```
x=xmlDoc.documentElement.childNodes;
for (i=0; i<x.len; i++) {
    txt += x[i].nodeName + ": " + x[i].childNodes[0].
           nodeValue + "<br>";
```

Browsers : Spaces are considered as nodes! (CR/LF)

PCDATA: parsed char data
CDATA: unparsed data (<, & : illegal in XML)
↳ avoid this : using CDATA!

```
<script>
<![CDATA[
    function matchwo(a,b) {
        if (a<b && a<0) {
            return 1;
        } else {
            return 0;
        }
    }
}
]]>
</script>
```

→ everything inside CDATA : ignored by the parser!

]]> : end of cdata (can't have spaces/lines)

Navigation:

1. Avoid empty text nodes:

```
function getNextSibling(n) {
    var y = n.nextSibling;
    while(y.nodeType != 1) {
        y = y.nextSibling;
    }
    return y
}
```

→ I want only element (no empty text nodes)

→ until text node!

(while: until element node)

2. First child:

// Same as above: y.firstChild

→ x.parentNode (exactly one)
→ x.nextSibling
→ x.

```
<book>
    <title> — </title>
</book>
```

→ 1st child : text node
book.firstchild.nextSibling = <title>
(no siblings: so it's self)

Why skip ≠ 1:

book
- → #text (new line, indentation)
- → title (newline, indentation)
- → #text (after title / below)

→ book.nextNode != textNode != title(element)
→ book.nextNode.sibling: No sibling (only textNode : skip)

type: text node (3)

Skip!

"I want element!")

Get values:

* xmlDoc.getElementsByTagName('title')[0].childNodes[0].nodeValue
  ↓ textNode

Value of attribute:

* xmlDoc.getElementByTagName('title')[0].getAttribute('lang')

all attributes (category):

* bookstore > book (category) > title, author, ...
  → getElementsByTagName('book');

book[i].getAttributeNode('category').nodeValue

> getAttribute: return value!
> getAttributeNode: node!

Dom: change nodeValue:

→ xmlDoc.getElementsByTagName('title')[0].childNodes[0].nodeValue = "new content"

→ .setAttribute("attribute", value);

→ .getAttributeNode("attribute_name").valueNode = "value";

Remove node:

→ xmlDoc.getElementsByTagName("book")[0] ⇒ y

child → xmlDoc.documentElement.removeChild(y);

self → y.parentNode.removeChild(y);

Remove text node:

→ x = ——— ("title")[0];

→ y = x.childNodes[0]; ⟶ textNode

→ x.removeChild(y);

clean text node:

x.nodeValue = "";

Remove attribute:

.removeAttribute("category");

.removeAttributeNode(attributeNode); ⟶ book.removeAttributeNode

Replace nodes:

x = xmlDoc.documentElement;

newBook = xmlDoc.createElement("book"), title, Text node ↗ createTextNode("_")

newTitle.appendChild(newTextNode);
newBook.appendChild(newTitle);
("book")[0].replaceChild(newBook); → replace book[0] by y!

Replace data in text node:

→ x.replaceNode (offset, length, string) ⟶ x: textNode.

→ other way: nodeValue = '_____';

Create:

xmlDoc • createElement ('book')

xmlDoc • createAttribute ('category') + setAttribute (book: element) / setAttributeNode

• createTextNode

CDATA: (formatted!) : preserve whitespaces

newCDATA = xmlDoc.createCDATASection ('_____');

xmlDoc.getElementsByTagName('book') [0].appendChild (newCDATA);

Comment:

xmlDoc.createComment ('_____');

Insert:

→ x.insertBefore (newNode, y) ⟵ book[3]

↳ xmlDoc.documentElement

→ x.insertData (offset, string);

↳ textNode

Clone:

oldNode.cloneNode (true: include all attributes & child)

XPATH

★ XSLT standard: major element (navigate) : 200+ built in functions

↳ JS, Java, Xml schema, PHP, C, C++, Python, ...

★ Xpath: node types: element, attribute, text, namespace, process-instruction, root & comment.

Atomic values: (no child/parent)

★ Items: atomic values / nodes

Relations → parent (only one)

→ child (0+)

→ Sibling (same parent)

→ Ancestors (parent, parent's parent, ...)

→ Descendant (children, children's children)

Syntax:

/ : root

// : current node : start from : match selection!

. : current node

.. : parent node

@ : select attributes

bookstore : select all nodes with this name (Select all nodes)

/bookstore : from root (all nodes ● bookstore)

bookstore/book : select book elements

//book : all book elements from the doc!

bookstore//book : select all descendants : book (from bookstore : parent)

//@lang : select all attributes : lang

Predicates: specific nodes:

/bookstore/book[1] ⟶ index : [JS: 'Selection Language', 'XPath']

/bookstore/book[last()]

last()-1

position()<3 ⟶ 1, 2

//title[@lang] ⟶ title elements with lang

//title[@lang='en']

/bookstore/book[price>35.00] : book with price > 35.00

/bookstore/book[price>35.00]/title : title of book //

Un known nodes:

* : match any node

@* : any attribute

node() : any node of any kind

/bookstore/* : all node of bookstore

//* : all elements of the doc

//title[@*] : all title with atleast one attribute (any type)

Several paths:

//book/title | //book/price : title & price of all book

//title | //price : all title & price elements

/bookstore/book/title | //price : all title of book and all price elements

Xpath axes (relationship context):

* ancestor, ancestor-or-self (ancestor + self)
* attribute, child, descendant, descendant-or-self
* following (everything after the closing tag of the current node)
* following-sibling (all siblings after the node
* namespace : namespace nodes of the current element node.
* self, parent, preceding (before), preceding-sibling

axisname::nodetest [predicate]

Example:

child:: book ⟶ children of current node (books)
attribute::lang ⟶ lang attribute of the current node
child::* ⟶ all children element // //
attribute::* ⟶ attribute of //
child::text() ⟶ textNode child of C.N
child::node() ⟶ all child nodes of C.N
descendant:: book ⟶ book desc of C.N
ancestor :: book ⟶ book anc of C.N
ancestor-or-self :: book ⟶ book anc + self of C.N
child::*/child::price ⟶ all child of children of C.N (grand child)

operators:

→ | (compute 2 node-sets)
→ +, -, *, div, =, !=, <, <=, >, >=, or, and, mod

eg:

/bookstore/book [price > 35] / title.

XSLT

* Extensible stylesheet language: Styling language for xml (XSL)
* XSLT: XSL transformation (xml ⟶ HTML)

CSS: Stylesheet for HTML
XSL: Stylesheet for xml

↳ xml has no predefined tags (no meaning)
<table>: doesn't mean html table (It may be 'furniture' anything)

XSL (more than just stylesheet)

→ XSLT (transforming lang)
→ xpath (navi //)
→ XSL-FO (format : discontinued)
→ xQuery (query Lang)

XSLT:

* XSL transformation (xml doc to another xml doc)/another type like html, xhtml
* with XSLT: transform each XSLT element ⟶ X(HTML) element.
* add/remove/rearrange/sort elements, perform tests (whether to display)
* XSLT uses Xpath (match predefined template : when matched transform)
* Browsers support: XSLT, Xpath!

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```
↳ namespace + version (To access XSLT elements)

(or)

```
<xsl:transform ... >
```

Stylesheet/Transform: Can use any!

example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="____" version="1.0">
<xsl:template match="/"> ⟶ root
  <html>
  <body>
  <h2>
  <table border="1">
    <tr bgcolor="#9acd32">
    ____
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      :
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Reference Style Sheet

```
<?xml ⟶ >
<?xml stylesheet
type="text/xsl"
href ⟶ >
```

Catalog ⟶ cd
- title
- artist
- company
- country
- price

<xsl:template>:

* Build a template (match with an xml element)
  ↳ entire xml / part of an xml '/': entire xml doc
* <xsl:stylesheet> : defines it as an XSLT doc!

<xsl:value-of>:

* <td><xsl:value-of select="catalog/cd/title" /></td>

* extract value of an xml element : add to o/p stream

<xsl:for-each>

```
* <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
    </tr>
  </xsl:for-each>
```

filter + for-each:

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

<xsl:sort select="artist">

```
* <xsl:for-each select="catalog/cd">
    <xsl:sort select="artist"/>
    <tr><td>...</td></tr>
  </xsl:for-each>
```

add as an element of xsl:for-each

<xsl:if> element:

```
<xsl:for-each select="catalog/cd">
  <xsl:if test="price &gt; 10">
  <tr>
  ——————
  </tr>
</xsl:if>
</xsl:for-each>
```

test: evaluated

xsl:choose [case]:

```
<xsl:for-each select="title"/>
  <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>
      ——————
      </xsl:otherwise>
    </xsl:choose>
  </tr>
</xsl:for-each>
```

multiple when, end: otherwise

<xsl: apply-templates>

* apply template to the current element's child nodes / current element
* Select attribute added: only process child elements matches attribute

```
<?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
      <xsl: apply-templates/>
    </body>
  </html>
</xsl:template>

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

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

'Same for artist'

xml to html:

* apply-templates: Based on xml elements
  → + matching templates ⟶ Give HTml!
* eg: title —xml→ matched template 'title'

XSLT on client

* previous: Include XSL stylesheet to the xml file (Browser does the transform)
* Issue: Non XSLT aware browser won't work!
* Solution: JS!

xml ⟶ catalog > cd → title, artist, country, year, price ⟶ no stylesheet reference

Stylesheet: (XSLT)

```
<xml ——— >
  <xsl:stylesheet xmlns:xsl='———'>
    <xsl:template match='/'>
      h2, table, tr, th
      xsl:for-each > tr > td
```

Javascript:

```
function loadXmlDoc (fileName) {
    if (window.ActiveXObject) {
        xhttp = new ActiveXObject ("Msxml2.XMLHTTP");
    } else {
        xhttp = new XMLHttpRequest ();
    }
    xhttp.open ("GET", filename, false);
    try {
        xhttp.responseType = "msxml-document"}
    catch (e) {}
    xhttp.Send ("");
    return xhttp.responseXml;
}
function displayResult () {
    xml = loadXmlDoc ("cd catalog.xml");
    xsl = loadXmlDoc ("cdcatalog.xsl");
    if (window.ActiveXObject || xhttp.responseType == "msxml-document")
    {
      ex.xml.transformNode (xsl);
      document.getElementById ("example").innerHTML = ex;
    }
    else if (document.implementation && document.implementation.createDocument)
    {
      xsltprocessor = new XSLTProcessor ();
      xsltprocessor.importStyleSheet (xsl);
      resultDoc = xsltprocessor.transformToFragment (xml, document);
      document.getElementById ("example").appendChild (resultDoc);
    }
}
```

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

Server:

* php, ASP, JS, ...

Xquery

* Query language : XML (Xquery 1.0, Xpath 2.0 : same data model + func + operators)
* Extract info, summary, XML → XHTML, search!
* Compatible with other W3C docs: XML, namespace, XSLT, Xpath, XML Schema.

```
mydoc = doc("books.xml")
```

→ mydoc/bookstore/book/title : all title elements under book

Predicate:

* xquery uses predicates to extract data from limited content of xml doc.
* myDoc /bookstore/book[price > 30]

FLWOR (For, Let, Where, order by, Return):

* for $x in doc("books.xml") /bookstore/book
  where $x/price > 30
  return $x/title

  Same as /bookstore/book[price > 30]/title

* for $x in doc("books.xml") /bookstore/book
  where $x/price > 30
  order by $x/title
  return $x/title

HTML + FLWOR:

```
<ul>
 {
  for $x in doc("books.xml")/bookstore/book/title
  order by $x
  return <li>{$x}</li>
 }
</ul>
```

↳ list eg (unordered) title.

Terminology:

* Node: element, attribute, text, namespace, processing-instruction, document (root), comment. (7 types)
  ↳ bookstore
* Atomic values:
  * nodes with no child/parent
  * J K. Rowling: eg
* Items: atomic values/nodes
* Relation: parent, children, sibling, Ancestors, Descendants
  ↓ only 1, ↓ 0/more, ↓ 0/more

Syntax:

* case sensitive
* must be: valid xml tags
* string: single/double quotes
* variable: $bookstore
* comments: : _____ :

eg:

```
for $x in doc("books.xml")/bookstore/book
return if ($x/category = "children")
then <child> {data ($x/title)} </child>
else <adult> {data ($x/title)} </adult>
```

Comparison:

* =, !=, <, <=, >, >= $bookstore//book/@v > 10
* Value: eq, ne, lt, le, gt, ge $bookstore//book/@v gt 10

Add elements & attributes to Result:

```
<ul> {
    for $x in doc("books.xml")/books/book/title
    order by $x
    return <li> {data ($x)} Category: {data($x)} </li>
} </ul>
```

attribute:

```
<li class = "{data ($x/@category)}" > {data ($x/title)} </li>
```

Select & filter:

```
for $x in doc('books-xml')/bookstore/book
where $x/price > 30
order by $x/title
return $x/title
```

FLWOR

for

```
for $x in (1 to 5)
return <test> ($x) </test>
```

at: (count iteration)

```
for $x at $i in doc('books.xml')/bookstore/book/title
return <book> {$i} . {data ($x)} </book>
```

multiple variables:

```
for $x in (20,30), $y in (100, 200)
```

print x, y

Variable assignment (let):

```
let $x := (1 to 5)
return <test> {$x} </test>
```

⟶ <test> 1 2 3 4 5

where:

```
where $x/price > 30 and $x/price < 100.
```

Orderby, return clause:

* orderby sort
* return: what to return

Xquery functions:

* Xquery (Same as Xpath) data types → String, Date, Numeric, misc

* Function calls
  - → {upper-case ($book title)}
  - → doc("books.com")/bookstore/book[substring(title, 1,5) = "Harry"]
  - → let $name := (substring ($book title, 1,4))

User defined:

```
declare function prefix : function_name ($parameter as datatype)
as returnData type
{
  ....
}

declare function local : minPrice ($p as xs: decimal ?, $d as xs: decimal?)
as xs: decimal ?
{
  let $disc := ($p * $d) div 100
  return ($p - $disc)
};

<minPrice> {local:minPrice ($book/price, $book/discount)} </minPrice>
```

XML DTD: Document Type Definition

```
<?xml version="1.0" ?>                          → root element: note
<!DOCTYPE note [
<!ELEMENT note (to, from, heading, body)>       → must (root) have 4 elements
<!ELEMENT to (#PCDATA)>                         → type PCDATA
:
]>
<note>
<to> Love </to>
<from> Jan 1 </from>
:
</note>
```

1. XML Doc: with internal DTD
2. External DTD declaration
   ↓
   `<!DOCTYPE note SYSTEM "note.dtd">`
   (XML): after `<?xml>` tag

Building blocks of DTD:

* elements: <body>, <message>
* attributes
* entities: < > &
* PCDATA: parsed char data (examined for entities & markup)
* CDATA: char data (won't be parsed)

Elements:

* Declare: <!ELEMENT element-name EMPTY> → category (empty elements)
* <!ELEMENT element-name (element-content)>
* <!ELEMENT from (#PCDATA)> : PCDATA
* <!ELEMENT note ANY> : Any content (combination of parsable data)
* <!ELEMENT element-name (child1, child2, ...)>
  → child1, child2 : must be there!

<!ELEMENT note (child1)> ⟶ only 1 child1 must be there under root!

<!ELEMENT note (child1+)> ⟶ min 1 occurrence.

<!ELEMENT note (child1*)> ⟶ 0/more

<!ELEMENT note (child1?)> ⟶ 0/1

either/or:

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

mixed content:

<!ELEMENT note (#PCDATA|to|from|header|message)*>
→ 0/more occ. of parsed data, to, from.. elements

DTD Attributes:

<!ATTLIST payment attribute-name att-type att-value>

Type:

CDATA
(en1|en2,..) : one from enumerated list
ID : unique ID
IDREF : another element (id of)
IDREFS : list of IDREFS of others
NMTOKEN : valid xml name
NMTOKENS : " names
ENTITY : value of an entity
ENTITIES : list of entities
NOTATION : name of a notation
xml : value : predefined xml name

attribute value

#REQUIRED
#IMPLIED (opt)
#FIXED

Default attribute:

Xml: <square width='100'/>

DTD: <!ELEMENT square EMPTY>

<!ATTLIST square width CDATA '0'>

REQUIRED:

<!ATTLIST person number CDATA #REQUIRED> ⟶ <person/> Invalid

↳ <person number='56'/> ↳ valid!

IMPLIED:

<!ATTLIST person fax CDATA #IMPLIED>

FIXED:

<!ATTLIST sender company CDATA #FIXED 'Microsoft'>

↳ fixed value!

<sender company='w3'> ⟶ Invalid xml

ENUMERATED Type:

<!ATTLIST payment type (check | cash) 'cash'>

Xml elements (vs) attributes:

* Data: store in child elements / attributes
* problem with attributes ⟶
  - can't have multiple values
  - Not expandable
  - can't describe structures
  - difficult to manipulate.
  - Not easy to test against a DTD.
* Exception: <note id='56'> : part of [illegible] file (not DOM)!

DTD Entities:

<!ENTITY entity-name 'entity-value'>

<!ENTITY writer 'Donald Duck'>

<!ENTITY copyright 'copyright'>

⟶ <author> &writer; &copyright; </author>

External entity:

<!ENTITY entity-name SYSTEM 'url/urn'>

<!ENTITY copyright SYSTEM 'https://____/entities.dtd'>

XSD : Xml Schema Definition

* Structure of Xml : legal (define) building blocks of an Xml
  - elements, attributes, no. of children
  - data types of elements/attributes
  - default & fixed values for elements & attributes

Why XSD:

* Xml based, powerful and an alternative to DTD

Xml Schemas supported data types:

* allowable document content (define, validate data correctness),
* Restrict data + define data patterns, convert data (b/w data types)

Advantages:

* Xml schema : written in Xml (no new language, parser, editor, manipulate Xml Dom, transform with XSLT)
* Extensible (other schemas: reuse, create own data types, multiple Schema: single document).

mutually agreed content for data communication

- Xml declaration at start
- unique root, start & end tags
- case sensitive, closing tags + nested
- quoted attribute values
- entities : used for special chars!

Xml

```
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>——</body>
</note>
```

DTD

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
:
```

XSD

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

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      :
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

<schema>:

* <xs: schema> .... </xs: schema>
* Contain attributes (may) : xmlns, targetNameSpace, xmlns, elementFormDefault
  - → xmlns:xs="http://www.w3.org/2001/XMLSchema"
    - ↳ elements & DS used from this namespace should be prefixed with xs:
  - → targetNamespace
    - ↳ elements (note, to, from, body, heading): Comes from this NS
  - → xmlns="http://www.w3schools.com" : Default namespace
  - → elementFormDefault="qualified"
    - ↳ xml instance elements in Doc must be namespace qualified

Reference a schema in xml doc:

<?xml version="1.0"?>

*

```
<note xmlns="____" xmlns:xsi="____"
      xsi:schemaLocation="____">
```

- xsi:schemaLocation → attribute (define XSD location)
- namespace for xsi (schema instance) prefix!
- default namespace.

XSD simple elements:

* only text (no element/attribute) : boolean, string, date, custom type, ...
* <xs: element name="___" type="yyy">
  - → xs: string
  - → xs: decimal, integer, boolean,
  - → xs: date, xs: time

default:

```
<xs: element name="color" type="xs:string"
             default="red" />
```

fixed:

```
<xs: element name="color" type="xs:string" fixed="red" />
```

XSD attributes:

* <xs: attribute name="___" type="___" />
* default, fixed : same as above! : optional by default
* required:

```
<xs: attribute name="lang" type="xs:string" use="required" />
```

XSD restrictions & Facets:

```
<xs: element name = 'age'>
    <xs: simpleType>
        <xs: restriction base = 'xs: Integer'>
Values:     <xs: minInclusive value = '0' />
            <xs: maxInclusive value = '120' />
        </xs: restriction>
    </xs: simpleType>
</xs: element>
```

Set of values:

```
<xs: restriction base = 'xs: String'>
    <xs: enumeration value = 'Audi' />
    :
</xs: restriction>
```

Series of values (pattern):

```
<xs: restriction base = 'xs: String'>
    <xs: pattern value = '[a-zA-z][a-zA-z][a-zA-z]' />
</xs: restriction>
```

* ([a-z][A-Z])+ Start small, any number (1/more): caps
* (male|female)

whitespace restrictions:

```
<xs: element name = 'address'>
    <xs: simpleType>
        <xs: restriction base = 'xs: string'>
            <xs: whitespace value = 'preserve' />
        </xs: restriction>
    </xs: simpleType>
</xs: element>
```

→ Won't remove spaces

→ replace: all whitespaces will be replaced by a space

→ collapse: (line feeds, tabs, spaces, carriage returns): replaced with spaces
Leading/trailing spaces, multiple spaces: replaced with single space.

length restriction:

```
<xs: restriction base = 'xs: string'>
    <xs: length value = '8' />          ⟶ Exactly
    <xs: minLength value = '2' />
    <xs: maxLength value = '8' />
```

totalDigits
whitespace
pattern

XSD: Complex elements:

* other xml elements and/or attributes
  - → empty elements
  - → elements having other elements
  - → elements contain only text
  - → elements with both other elements & text

eg:

2)
```
<employee>
    firstName
    lastName
</employee>
```
: elem having other elem

1) <Product pid='123'> : empty elem

3) <food type='dessert'>Ice</food>
   ↳ element with text only

4) <description>
   text
   <date ——> ——</date>
   </description>
   ↳ both text & other elements

XSD ComplexType:

```
<xs:element name='employee'>
    <xs:complexType>
        <xs:sequence>
            <xs:element name='firstName' type='xs:string'/>
            <xs:element name='lastName' type='xs:string'/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```
↳ directly define

(or)

```
<xs:element name='employee' type='personInfo'/>
<xs:complexType name='personInfo'>
    <xs:sequence>
        firstName
        lastName
    </xs:sequence>
</xs:complexType>
```
↳ Just refer, define elsewhere!
Adv: define once, use anywhere (1+times)

Compound:
- → employee (fullPersonInfo)
- → personInfo (fname, lname)
- → fullPersonInfo (personInfo, address, city, country)

one complex having other

1) Empty elements:

→ <product pid='1'/>

<xs:element name='product'>
ComplexType > SimpleContent
restriction (Integer)
attribute (name, type)

2) Elem only:

xs: complexType > xs:Sequence → xs: element (fname)
→ name

3) Text only:

xs: complexType > xs: SimpleContent > restriction.

4) mixed:

→ xs: element (letter) > xs: complexType (mixed='true')

→ name, orderId, ShipDate.

→ letter inside letterType!

Indicators (7 types)

order → all (child elements: any order but only once)
→ choice (either 1 or other child can occur)
→ Sequence (child elements: appear in order)

occurrence → maxoccurs (max no. of times: child/element can occur)
→ minoccurs (min " ")

Group → group name (related/set of elements)
→ attribute group name (attribute group)

<any>:

* <xs: element name='person'>
<xs:complexType>
<xs: Sequence>
fname
lname
<xs: any minoccurs='0'/> ⟶ any element can occur!

<anyAttribute>:

* extend: undeclared attributes (adds any number of attributes!)

Element Substitution:

* let the user choose which elements: English/Norway.

<xs: element name='name' type='xs: string'/>

<xs: element name='navn' substitution group='name'/>

<xs: ComplexType name='custinfo'>
<xs: Sequence>
<xs: element ref='name'/>
</xs: Sequence> </xs: ComplexType>.

<xs: element name = 'customer' type = 'custInfo' />
<xs: element name = 'Kunde' substitutionGroup = 'customer' />

Valid xml (as per above schema):

Customer > name (or) Kunde > navn

Block element substitution:

<xs: element name = 'name' type = 'xs:string' block = 'substitution' />

↳ can't substitute for the element 'name'!

Substitution Group:

* Substitution elements must be of the same type (derived/head)
* If same type: no need to specify type of substitute element!

Note: substituteGroup: must be global elements (immediate child from the root).

XSD data types: (String)

1) String (xs: string) → John Smith
→ ... John Smith .....

2) xs: normalizedString (replace tabs with spaces), remove linefeeds, carriage returns, tabs.

3) xs: token (remove feeds, carriage returns, tabs, leading/trailing spaces, multiple spaces)

4) ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCNAME, NMTOKEN, NMTOKENS, normalizedString, QNAME, string, token

Restrictions: length, minLength, maxLength, enumeration, patterns (NMTOKENS, IDREFS, ENTITIES: Last) whitespace

Date & time:

* xs: date (2002-09-24), (2002-09-24Z), (2002-09-24+06:00)
* xs: time (09:00:00, 09:30:10.5, 09:30:10Z, 09:30:10-06:00)

yyyy: year
mm: month
DD: Date

hh, mm, ss

* DateTime (yyyy, mm, DD, T, hh, mm, ss)
↓
Start of time section

* duration (P5Y, P5Y2M10DT15H) (-P10D)

P: period
nY: no. of years
nM: " month
nD: " Day
T: Start of time
nH, nM, nS

↓ period
↓ number of years

other: date, dateTime, duration, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Restriction: enumeration, minExclusive, minInclusive, maxInclusive, maxExclusive, pattern, whiteSpace

---

Numeric:

* xs: decimal (+999.540, −995.532)
* xs: integer (+99, −99)

byte, decimal, int, Integer, long, negativeInteger, NonPositiveInteger, NonNegativeInteger, positiveInteger, short, unsignedLong, unsignedInt, unsigned short, unsignedByte

(enum, fractionDigits, maxExclusive, maxInclusive, minInc, minExc, pattern, totalDigits, whiteSpaces)

Misc:

* xs: boolean (true, false: 1/0)
* xs: hexBinary, base64Binary (base64 encoded, hexdecimal encoded)
* xs: anyURI (URL)

double, float, NOTATION, QNAME

enu, length, maxLength, minLength, pattern, whiteSpace

---

webservices: (app components: use, publish, find)

* WSDL (Web Services Description language): xml based: W3C recomm.
* SOAP (Simple Object Access protocol), xml based protocol for accessing webservices, W3C recomm
* RDF: Resource description framework (describe web resources) written in XML, W3C recomm.
* RSS: Really Simple Syndication (syndicate web content)
  - easy way to share, view headlines & content
  - Auto update files, personalized views for diff sites
  - written in xml

webservices → app component (communicate using protocol)
- self contained, self descriptive
- discovered using UDDI
- used by other app
- Basis: HTML & XML

Different platforms interact (run by any browser/platform)

xml: datasharing, SOAP protocol

Types → Reusable webapp (currency conversion, ... services)
→ connect existing software (Solve interoperability)

XML WSDL:

* definitions > types > message > portType > binding
* <message name = 'getTermRequest'
  <part name = 'term' type = 'xs:string' />
  </message>
* portType : oneway, request – response, solicit – response, notification

Bind to SOAP

SOAP (Simple Object Access Protocol)

* HTTP / HTTPS : SOAP helps with communication
* envelope element (xml), header, body, fault tolerant (error, status info)
* Rules:
  - xml encoded, use SOAP envelope namespace
  - No DTD reference, xml processing instruction

<?xml > soap:envelope > soap:header, soap:body, soap:fault

↓ namespace, encodingStyle

soap:mustUnderstand = '0|1' : 1: must recognize child

* soap:actor = 'URI' → ultimate endpoint
* soap:encodingStyle = 'URI'
* soap:fault → code, string, actor, detail

SOAP binding
Content-type
Content-length

Request	Response
POST /InStock HTTP/1.1 Host: www.example.org Content-Type: application/soap+xml; charset = utf-8 Content-length: nnn <?xml version = '1.0' ?> <soap:envelope xmlns:soap = 'http://www.w3.org/2003/05/soap-envelope' soap:encodingStyle = 'http://www.w3.org/2003/05/soap-encoding'> <soap:Body xmlns = '____/stock'> <m:GetStockPrice> <m:StockName> IBm </m:StockPrice> </m:____> </soap:body> </soap:Envelope>	HTTP/1.1 200 OK Content-Type: application/soap+xml; char... Content-length: nnn xml ____ <soap:envelope ____> <soap:body ____> <m:getStockPriceResponse> <m:price> 34.5 </m:price> </soap:body> </soap:envelope>

RDF: Resource Description Framework:

* read & understand by computers: not people (written in XML)
* eg: describe properties of item (price, availability, content, rating, ...
* resource: http://www.w3schools.org/rdf'
  Property: author, homepage
  Property value: Jan Egil

```
<rdf: Description>
  <rdf: about xmlns...>
    <rd: Bag>
      <rdf: li> John </rdf: li>
      ⋮
```

RSS:

* Distribute upto date content (feed): small, fast loading
  news sites, calendars, site changes

Channel > title, link, description, item

HTML

<img> with Cross-origin & <canvas>:

* load from foreign origin: use in <canvas>
  ↳ no CORS approval: tainted (can't use getImageData, toBlob, toDataURL, gives exception)
* webserver: allow cross origin (site) in header!
  ↳ write in browser local storage, load in page)

Crossorigin attribute (audio, img, link, script, video):

* provide CORS support: fetch CORS requests.
  Crossorigin = 'anonymous | use-credentials'
  "" : means anonymous
* anonymous: same-origin (CORS header to set) + no cookie credential exchange under same origin
* use-credential: request CORS headers, flag (credentials): include

rel='preload'

* available as soon as the rendering starts! → fetch → font, image, script, style, track

Responsive Images:

* Works well in wide screen, In viewport below 1200px (Image size taken 100%, (body width), centered
* Shrink rathan than overflow (Lot of screen height : text)
* Options: display cropped image (art-direction problem)

<img

srcset = " .jpg 480w , .jpg 800w"

sizes = "(max-width: 600px) 480px, 800px"

src = " "

alt = " " />

srcset:

↳ image intrinsic_value (pixel width)

sizes:

↳ max-width: 600px : media condition

why no css/JS : In img (browser preloads img)