XML(REPRESENTING WEB DATA)

XML DOCUMENTS:

- An XML document is one that follows certain syntax rules (most of which we followed for XHTML)

- An XML document consists of
  - Markup
    - Tags, which begin with < and end with >
    - References, which begin with & and end with ;
      - Character, e.g.
      - Entity, e.g. <
  - Character data: everything not markup

Staff1.xml:

```
<?xml version="1.0"?>
<company>
    <staff id="1001">
        <firstname>yong</firstname>
        <lastname>mook kim</lastname>
        <nickname>mkyong</nickname>
        <salary>100000</salary>
    </staff>
    <staff id="2001">
        <firstname>low</firstname>
        <lastname>yin fong</lastname>
        <nickname>fong fong</nickname>
        <salary>200000</salary>
    </staff>
</company>
```

- Element tags and elements
  - Three types

        - Start, e.g. <message>
        - End, e.g. </message>
        - Empty element, e.g.
- XML Rules:
    - Start and end tags must properly nest
    - Corresponding pair of start and end element tags plus everything in between them defines an element
    - Character data may only appear within an element
    - Start and empty-element tags may contain attribute specifications separated by white space
        - Syntax: *name = quoted value*

Well-formed XML document:

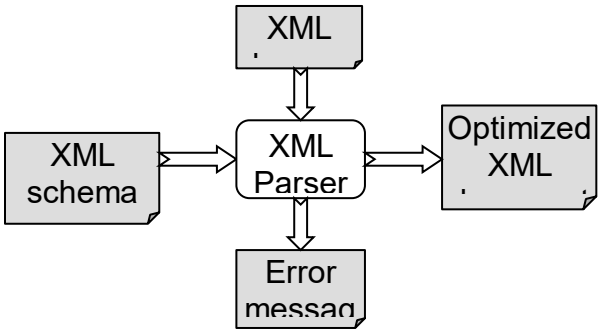
- A well-formed XML document
    - follows the XML syntax rules and
    - has a single root element
- Well-formed documents have a tree structure
- Many XML parsers (software for reading/writing XML documents) use tree representation internally
- Properties
    - Documents must have a root element
    - Elements must have a closing tag
    - Elements must be properly nested
    - Attribute values must be quoted
- Advantage
- Avoids fixed nature like HTML
- Flexible
- Expandable

XML parsers

- Two types of XML parsers:
    - Validating
        - Requires document type declaration
        - Generates error if document does not
            - Conform with DTD and
            - Meet XML validity constraints

» Example: every attribute value of type ID must be unique within the document

– Non-validating
  - Checks for well-formedness
  - Can ignore external DTD



XML NAMESPACES:

- XML Namespace: Collection of element and attribute names associated with an XML vocabulary (such as XHTML)
- Namespace Name: Absolute URI that is the name of the namespace
  – Ex: http://www.w3.org/1999/xhtml is the namespace name of XHTML 1.0
- Default namespace for elements of a document is specified using a form of the xmlns attribute:

Another form of xmlns attribute known as a namespace declaration can be used to associate a namespace prefix with a namespace name:

xmlns:h="http://www.w3.org/TR/html4/"

- In **a namespace-aware** XML application, all element and attribute names are considered qualified names
  – A qualified name has an associated expanded name that consists of a namespace name and a local name
  – Ex: <table> is a qualified name with expanded name <null, table>
  – Ex: <h:table> is a qualified name with expanded name
  < http://www.w3.org/TR/html4, table>

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>African Coffee 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.An XML parser will not know how to handle these differences.

Solving the Name Conflict Using a Prefix

When using prefixes in XML, a so-called **namespace** for the prefix must be defined. The namespace is defined by the **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="http://www.w3schools.com/furniture">
<f:name>African Coffee Table</f:name>
<f:width>80</f:width>
<f:length>120</f:length>
</f:table>
</root>
```

XML DOCUMENT TYPE DEFINITION:

- Allows developers to create a set of rules to specify legal content and place restrictions on an XML file
- Parser generates error, if XML document does not follow the rules contained within DTD
- Including a DTD
  - **Using internal declaration**
  - **Using external file**
  - **Both**
- For custom documents
- Uses DOCTYPE declaration

Internal (standalone) DTD:

hello.xml:

<!DOCTYPE greeting [

<!ELEMENT greeting (#PCDATA)>

]>

<greeting>Hello, world!</greeting>

- Specify in XML declaration

<?xml version="1.0" standalone="yes"?>

greeting>Hello, world!</greeting>

External DTD:

- Most common dtd.
- Use DOCTYPE declaration before root element

hello.dtd:

<!DOCTYPE greeting [

<!ELEMENT greeting (#PCDATA)>

]>

hello.xml:

<!DOCTYPE greeting SYSTEM "hello.dtd">

<?xml version="1.0" standalone=“no"?>

<greeting>Hello, world!</greeting>

External plus Internal DTD:

- Usually to declare entities
- Use DOCTYPE declaration before root element

hello.xml:

```
<!DOCTYPE greeting SYSTEM "hello.dtd" [
 <!ENTITY excl   "&#x21;">
]>
<greeting>Hello, world&excl;</greeting>
```

XML Building Blocks:

- XML documents consist of following blocks
  - Elements
  - Attributes
  - Entities
    - < > & " &apos
  - PCDATA
    - Parsed Character DATA
    - Entities will be expanded
  - CDATA
    - Character DATA
    - Entities will not be expanded

DECLARING ELEMENTS:

An empty element

<!ELEMENT elementName (EMPTY)>

Example

<!ELEMENT br (EMPTY)>

<!ELEMENT Bool (EMPTY)>

Usage:

```
<br/>
<Bool Value="True"></Bool>
```

Element with data

<!ELEMENT elementName (#PCDATA)>

Example

<!ELEMENT from (#PCDATA)>

<!ELEMENT question (#PCDATA)>

Usage:

<from>U. K. Roy</from>

<question>

What is the full form of DTD?

</question>

Example : Elements with Data

<!ELEMENT Month (#PCDATA)>

Valid Usage

<Month>April</Month>

<Month>This is a month</Month>

Invalid Usage:

<Month> <!—Invalid usage within XML file, can't have children!-->

<January>Jan</January>

<March>March</March>

</Month>

Element with Children (sequential)

<!ELEMENT elementName (child1, child2,…)>

Example

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

<!ELEMENT address (street, city, zip)>

Inner elements must also be declared

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

<!ELEMENT from (#PCDATA)>

<!ELEMENT to (#PCDATA)>

<!ELEMENT body (#PCDATA)>

<!ELEMENT address (street, city, zip)>

<!ELEMENT street (#PCDATA)>

<!ELEMENT city (#PCDATA)>

<!ELEMENT zip (#PCDATA)>

Occurrence Indicators:

Term	Meaning	Example
,	**Sequence Operators**	**a, b, c**
|	**Choice operators**	**a | b | c**
+	**One or more**	**a+**
*	**Zero or more**	**a***
?	**Single optional**	**a?**
()	**grouping**	**(a)**

DECLARING ATTRIBUTES:

General Syntax

<!ATTLIST elementName attributeName attributeType defaultType>

Example

<!ATTLIST HDD speed CDATA “7200”>

<!ATTLIST HDD unit CDATA #IMPLIED>

<!ATTLIST price currency CDATA “INR”>

<!ATTLIST question number ID #REQUIRED>

<HDD speed=“6000”> … </HDD>

<price currency=“USD”>10</price>

<question number=“1”> … </question>

The **attribute-type** can be one of the following:

Type	**Description**
CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

The default-value can be one of the following:

Value	**Explanation**
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is not required
#FIXED *value*	The attribute value is fixed

Examples--#REQUIRED:

DTD:

<!ATTLIST person number CDATA #REQUIRED>

Valid XML:

<person number="5677" />

Invalid XML:

<person />

Examples--#IMPLIED:

DTD:

<!ATTLIST contact fax CDATA #IMPLIED>

Valid XML:

<contact fax="555-667788" />

valid XML:

<contact />

Examples--#FIXED:

DTD:

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

Valid XML:

<sender company="Microsoft" />

Invalid XML:

<sender company="W3Schools" />

Examples—Enumerated:

DTD:

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

XML example:

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

DECLARING ENTITIES:

General Syntax

<!ENTITY entityName “entityValue”>

Example

<!ENTITY euro “€”> //€

<!ENTITY language “XML”>

<!ENTITY W3C “World Wide Web Consortium”>

<!ENTITY copyright “©”> //©

<!ENTITY USD SYSTEM “currency.dtd”>

<tutorial>

&language; is standardized by &W3C;

&copyright; UKR

</tutorial>

XML SCHEMA

XML Schema is an XML-based alternative to DTD.

An XML schema describes the structure of an XML document.

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

The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML Schema:

defines elements that can appear in a document

defines attributes that can appear in a document

defines which elements are child elements

defines the order of child elements

defines the number of child elements

defines whether an element is empty or can include text

defines data types for elements and attributes

defines default and fixed values for elements and attributes

Simple Xml Document:

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

DTD FILE:

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

note.xsd:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<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>
```

Reference to Schema:

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

<note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Simple Element

A simple element is an XML element that contains only text. It cannot contain any other elements or attributes.

A default value is automatically assigned to the element when no other value is specified.

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

A fixed value is also automatically assigned to the element, and you cannot specify another value.

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

Complex Element

A complex element is an XML element that contains other elements and/or attributes.

Kinds of complex elements:

- Empty elements
- Elements that contain only other elements

A complex XML element, "product", which is empty:

```
<product pid="1345"/>
```

Define a complex element that contain attributes.

```
<xs:attribute name="lang" type="xs:string" default="EN"/>

<xs:attribute name="lang" type="xs:string" fixed="EN"/>

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

A complex XML element, "employee", which contains only other elements:

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

Define a complex element that contain other elements in an XML Schema two different ways

1. Only the "employee" element can use the specified complex type

```
<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>
```

2. Several elements can refer to the same complex type

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>
```

```
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Indicators

Order indicators:

Order indicators are used to define the order of the elements.

- **All**

  The <all> indicator specifies that the child elements can appear in any order, and that each child element must occur only once:

- **Choice**

  The <choice> indicator specifies that either one child element or another can occur:

- **Sequence**

  The <sequence> indicator specifies that the child elements must appear in a specific order:

Occurrence indicators:

- **maxOccurs**

  The <maxOccurs> indicator specifies the maximum number of times an element can occur:

- **minOccurs**

  The <minOccurs> indicator specifies the minimum number of times an element can occur:

  Schema:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence> or  <xs:choice> or   <xs:all>
```

```
    <xs:element name="full_name" type="xs:string"/>
    <xs:element name="child_name" type="xs:string"
    maxOccurs="10" minOccurs="0"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

Xml:

```
<person>
  <full_name>Tove Refsnes</full_name>
  <child_name>Hege</child_name>
  <child_name>Stale</child_name>
  <child_name>Jim</child_name>
  <child_name>Borge</child_name>
</person>
```

Restrictions for Datatypes

enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable

totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

Inclusive and Exclusive:

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxExclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Enumeration:

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Pattern:

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Minlength and Maxlength:

```
<xs:element name="password">
  <xs:simpleType>
```

```
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

X-Files(XFiles,XPointer,Xlink):

XPath axes

An XPath axis is a path through the node tree making use of particular relationship between nodes. We use the "child::*" axis and the "attribute::*" axis all the time but mostly their short form: "*" and "@*".

1. Axis examples

Most often the axes, e.g.: "descendant::", are followed by an "*" (element) or an element name
but comment(), processing-instruction() andtext-node() can also be used. Only root node and elements can have children.

The construct node() only selects nodes being children of other nodes: element(), text(), comment(), processing-instruction(). An attribute has a parent element but attribute and namespace nodes are not children and not included in the node() construct or type.

descendant::*	All elements being children and children's children, etc, of context node.
descendant::stock	All "stock" elements being children and children's children, etc, of context node.
descendant::comment()	All comment nodes being children of context node.
descendant::*|comment()	All element and comment nodes being children or children's children, etc, of context node.
descendant::node()	All nodes being children or children's children, etc, of context node.

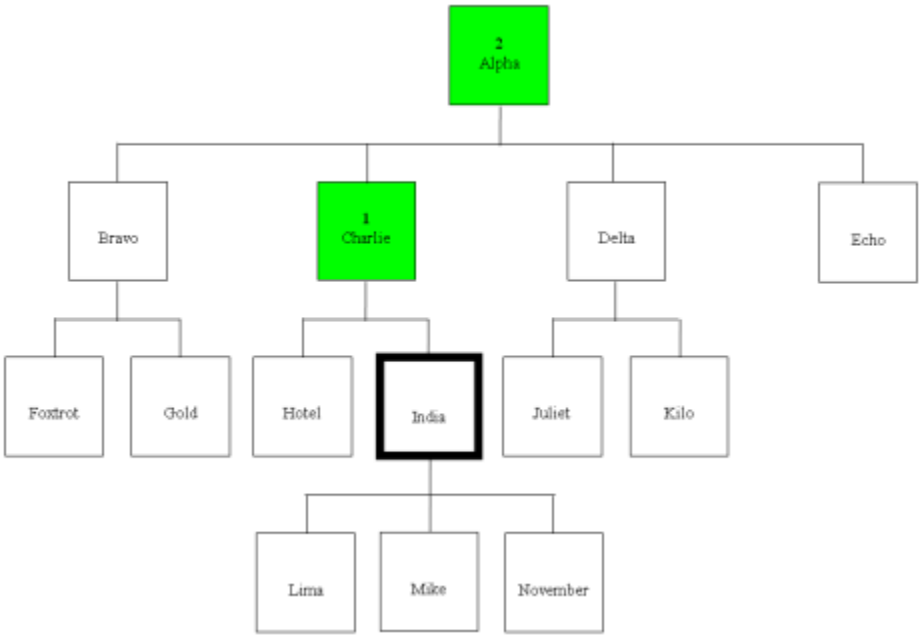
2. Axis diagrams

The diagrams are not showing attribute() and namespace() nodes and I have not made diagrams for the attribute and namespace axes. The namespace axis is deprecated. The functions in-scope-prefixes() and namespace-uri-for-prefix() should be used instead.

When using axes it is often important to remember that the nodes are processed in document order (indicated below). If both a node's children and siblings are included in an axis, the node's children are always processed before its following siblings. The order is sometimes a little tricky especially for reverse axes until you get used to it. See the preceding axis as example.

2.1 ancestor

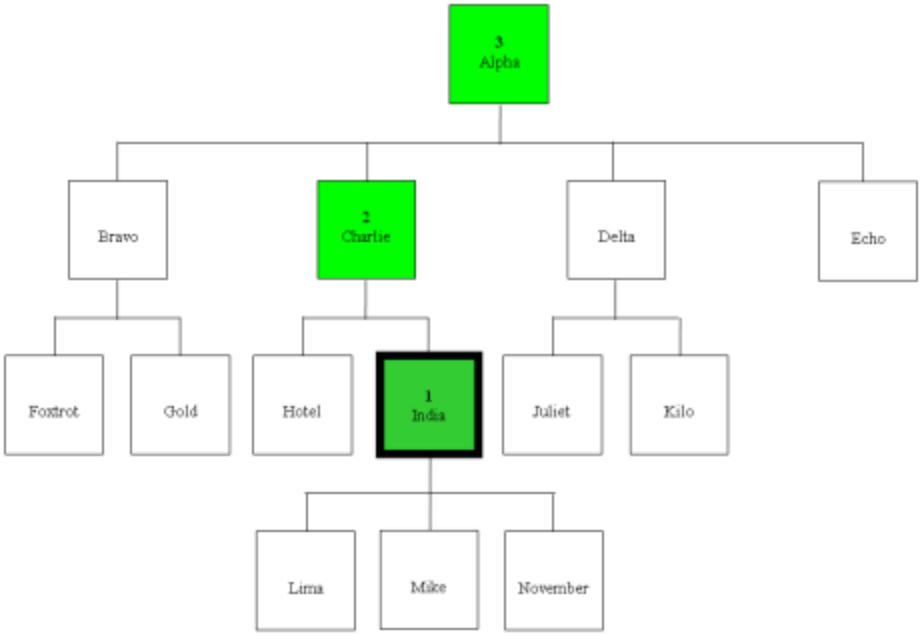
Ancestor axis: parent and parent's parent, etc.



India is context node: count(ancestor::*) returns 2.

2.2 ancestor-or-self

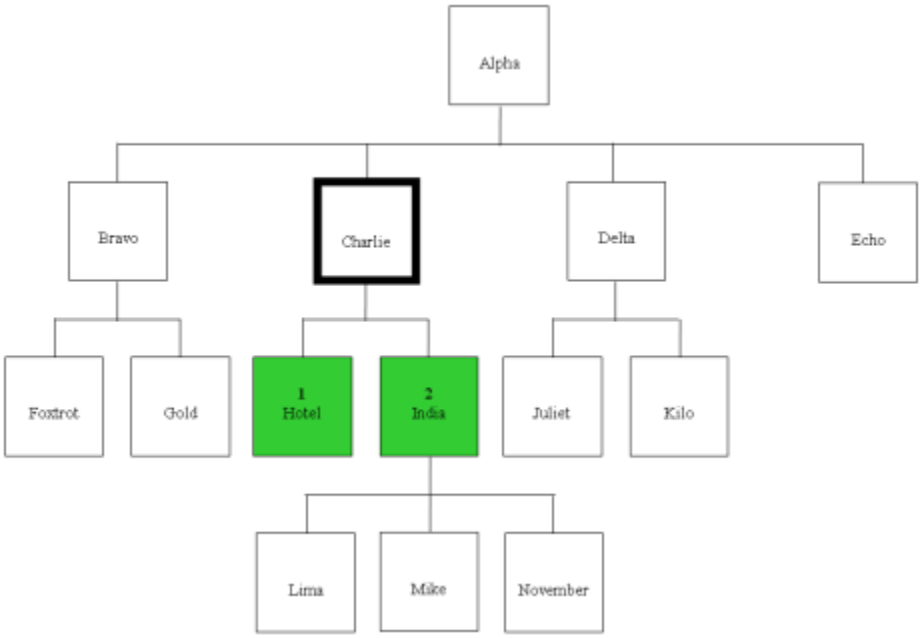
Ancestor-or-self axis: context node and parent and parent's parent, etc.



India is context node: count(ancestor-or-self::*) returns 3.

2.3 child

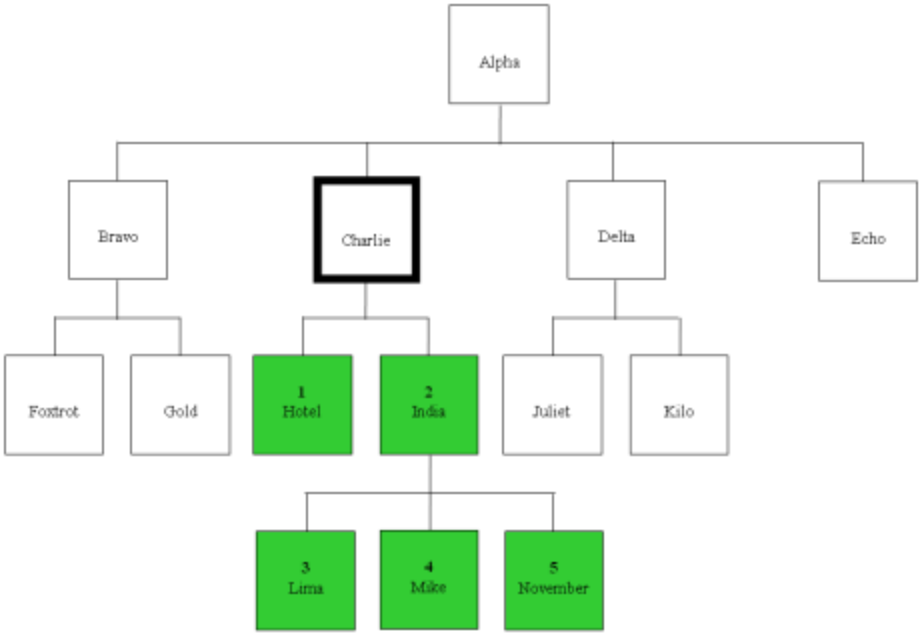
Child axis: children of context node.



Charlie is context node, e.g.: count(child::*) returns 2. Same as count(*).

2.4 descendant

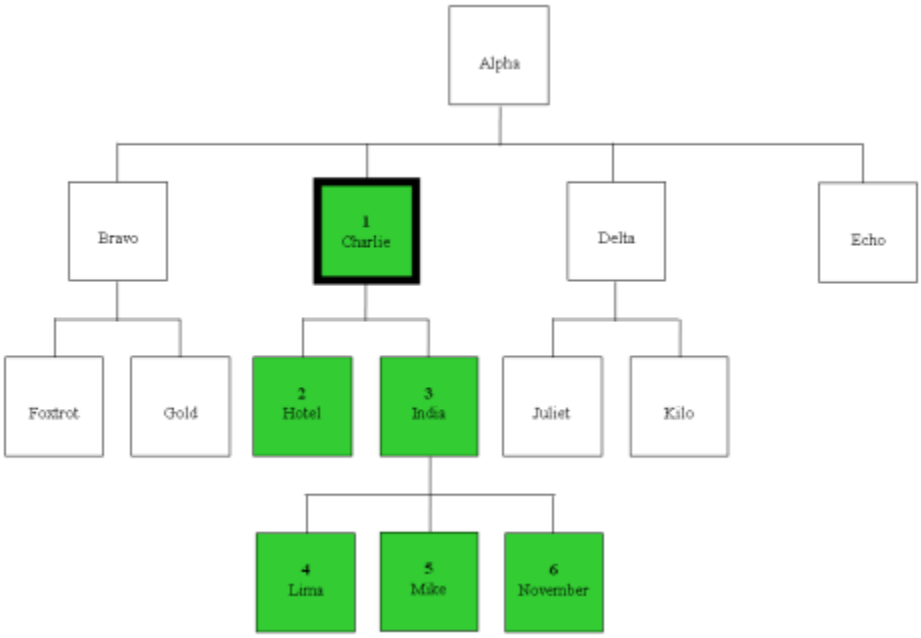
Descendant axis: children and their children, etc.



Charlie is context node, e.g.: count(descendant::*) returns 5.

2.5 descendant-or-self

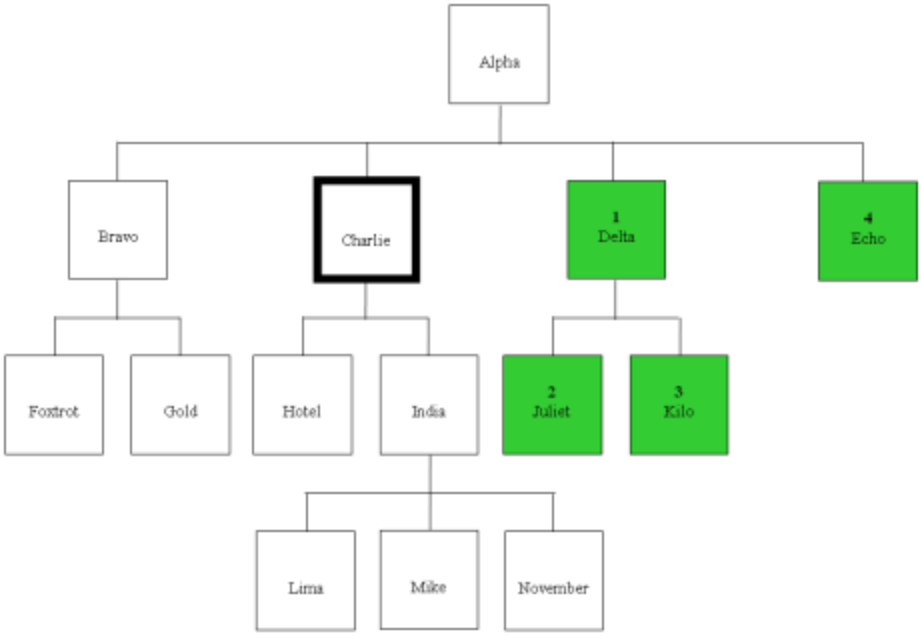
Descendant-or-self axis: context node and children and their children, etc.



Charlie is context node, e.g.: count(descendant-or-self::*) returns 6.

2.6 following

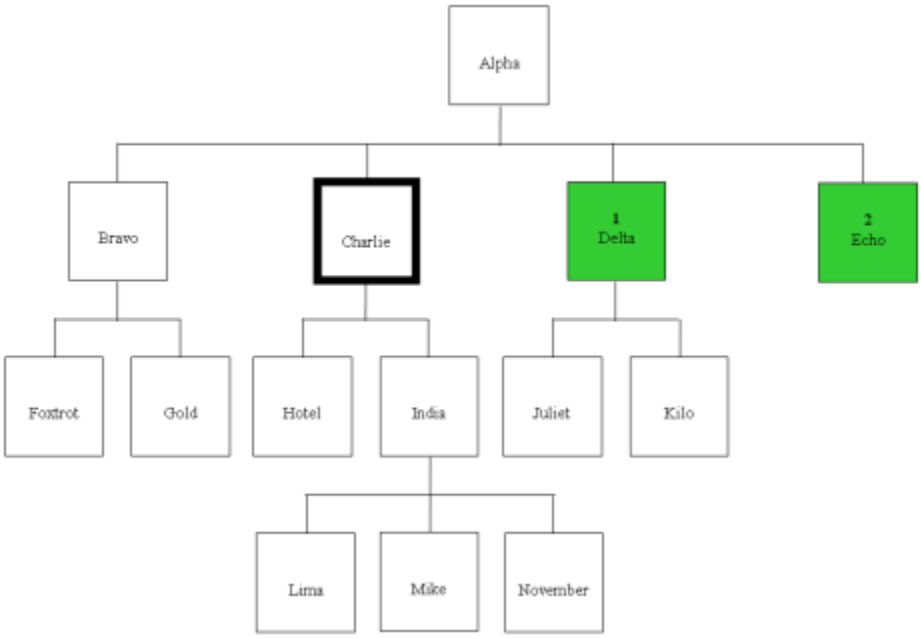
Following axis: following siblings and their children and their children, etc.



Charlie is context node, e.g.: count(following::*) returns 4.

2.7 following-sibling

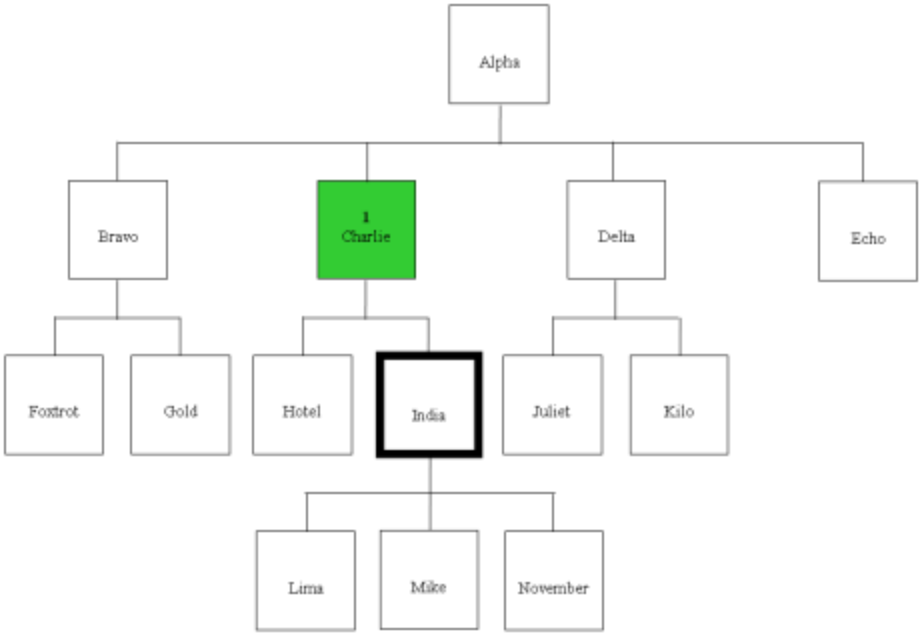
Following-sibling axis: following siblings.



Charlie is context node, e.g.: count(following::*) returns 2.

2.8 parent

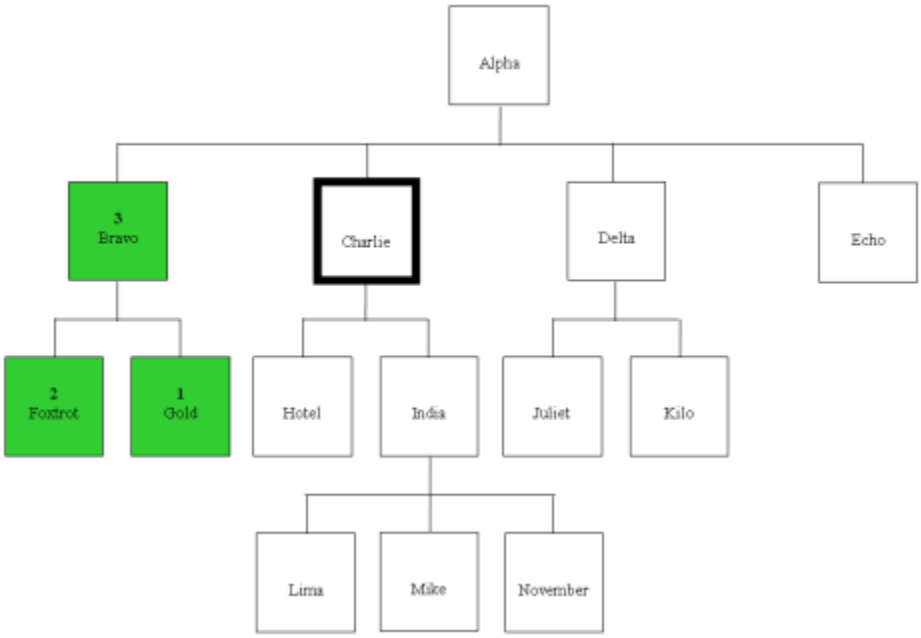
Parent axis: The parent of context node.



India is context node, e.g.: if (parent::* eq 'Charlie') then '...' else '...'

2.9 preceding

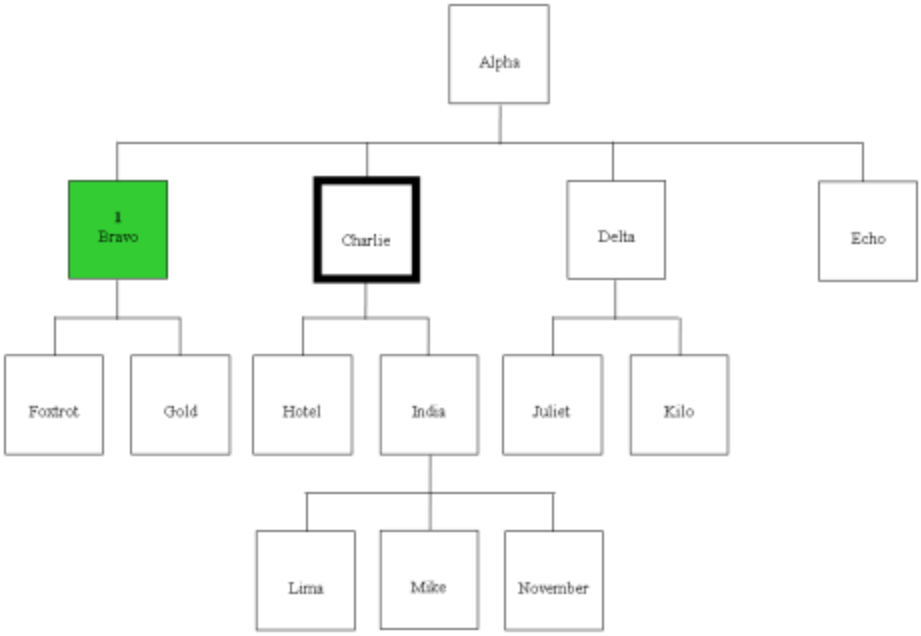
Preceding axis: preceding siblings and their children and their children, etc.



Charlie is context node, e.g.: count(preceding::*) returns 3

2.10 preceding-sibling

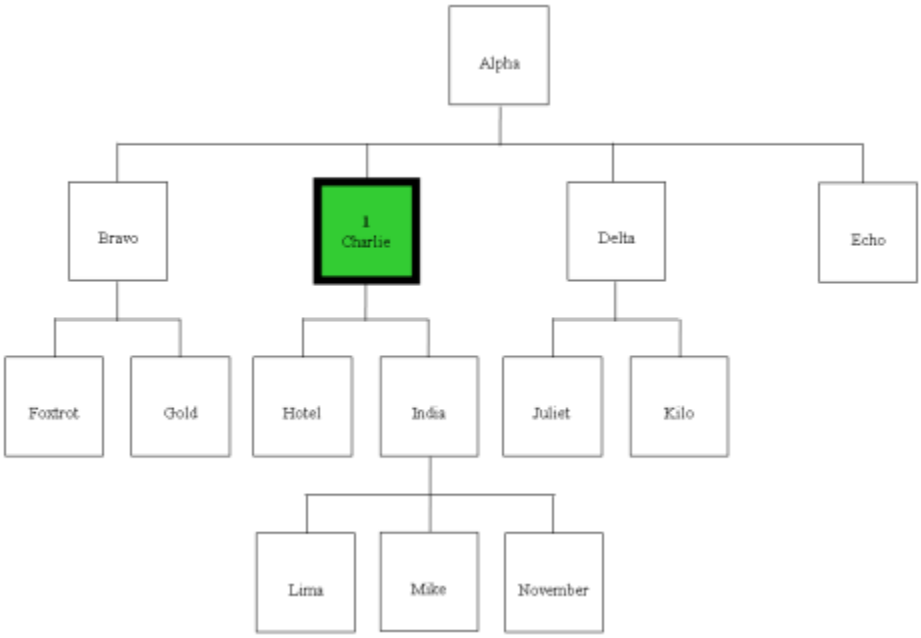
Preceding-sibling axis: preceding siblings.



Charlie is context node, e.g.: count(preceding-sibling::*) returns 1

2.11 self

Self axis: context node.



Charlie is context node, e.g.: if (self::* eq 'Charlie') then '...' else '...'

XML:

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

<!-- Edited by XMLSpy® -->

```
<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
</catalog>
```

XSL:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by XMLSpy® -->
<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="count(descendant::node())" /></td>
        <td><xsl:value-of select="child::artist" /></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
```

</html>

</xsl:template>

</xsl:stylesheet>

OUTPUT:

Title	Artist
19	Bob Dylan

descendant::node()=19; Which includes all nodes like text,element,comment etc

descendant::text()=13; Which includes all text nodes alone.

<catalog>

<cd>**(t1)**

<title>Empire Burlesque**(t2)**</title>**(t3)**

<artist>Bob Dylan**(t4)**</artist>**(t5)**

<country>USA**(t6)**</country>**(t7)**

<company>Columbia**(t8)**</company>**(t9)**

<price>10.90**(t10)**</price>**(t11)**

<year>1985**(t12)**</year>**(t13)**

</cd>

</catalog>

Total **text** nodes:13

Total nodes: Text nodes**(13)**+Elements**(6)=19**

Selecting Nodes

XPath uses path expressions to select nodes in an XML document. The node is selected by following a path or steps. The most useful path expressions are listed below:

Expression	Description
Nodename	Selects all nodes with the name "*nodename*"

/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

ABBREVIATIONS

Parent::node()=**..(two dots)**

Self::node()=**.(single dot)**

[attribute display=”visible”]=> [@display=”visible”]

XLink:

- 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 Recommendation

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

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<homepages xmlns:xlink="http://www.w3.org/1999/xlink">
  <homepage xlink:type="simple" xlink:href="http://www.w3schools.com">Visit W3Schools</homepage>
  <homepage xlink:type="simple" xlink:href="http://www.w3.org">Visit W3C</homepage>
</homepages>
```

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.

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

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

<dogbreeds>

```
<dog breed="Rottweiler" id="Rottweiler">
  <picture url="http://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="http://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>
```

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="http://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="http://dog.com/dogbreeds.xml#Rottweiler".

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="http://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="http://dog.com/dogbreeds.xml#FCRetriever">
  Fact about flat-coated Retriever
  </fact>
</mydog>
</mydogs>
```