XML

XML

- XML stands for eXtensible Markup Language

- HTML is used to mark up text so it can be displayed to users
- HTML describes both structure (e.g. <p>, <h2>,) and appearance (e.g.
-
, <font>, <i>)
- HTML uses a fixed, unchangeable set of tags

- XML is used to mark up data so it can be processed by computers
- XML describes only content, or “meaning”
- In XML, you make up your own tags

HTML & XML

- HTML and XML look similar, because they are both SGML languages (SGML = Standard Generalized Markup Language)
- Both HTML and XML use elements enclosed in tags (e.g. <body>This is an element</body>)
- Both use tag attributes (e.g., <font face="Verdana" size="+1" color="red">)
- Both use entities (<, >, &, ", ')
  - HTML is defined in SGML
  - XML is a (very small) subset of SGML

HTML & XML

- HTML is for humans
- HTML describes web pages
  - You don’t want to see error messages about the web pages you visit
  - Browsers ignore and/or correct as many HTML errors as they can, so HTML is often sloppy
- XML is for computers
- XML describes data
  - The rules are strict and errors are not allowed
  - In this way, XML is like a programming language
  - Current versions of most browsers can display XML
  - However, browser support of XML is spotty at best

Technologies

- DTD (Document Type Definition) and XML Schemas are used to define legal XML tags and their attributes for particular purposes
- CSS (Cascading Style Sheets) describe how to display HTML or XML in a browser
- XSLT (eXtensible Stylesheet Language Transformations) and XPath are used to translate from one form of XML to another
- DOM (Document Object Model), SAX (Simple API for XML, and JAXP (Java API for XML Processing) are all APIs for XML parsing

XML Basics

- XML documents are readable by both humans and machines
- XML permits document authors to create custom markup for any type of information
  - Can create entirely new markup languages that describe specific types of data, including mathematical formulas, chemical molecular structures, music and recipes
- An XML parser is responsible for identifying components of XML documents (typically files with the `.xml` extension) and then storing those components in a data structure for manipulation
- An XML document can optionally reference a Document Type Definition (DTD) or schema that defines the XML document's structure
- An XML document that conforms to a DTD/schema (i.e., has the appropriate structure) is valid
- If an XML parser (validating or nonvalidating) can process an XML document successfully, that XML document is well-formed

Example

```
<?xml version="1.0"?>
<weatherReport>
    <date>7/14/97</date>
    <city>North Place</city>, <state>NX</state>
    <country>USA</country>
    High Temp: <high scale="F">103</high>
    Low Temp: <low scale="F">70</low>
    Morning: <morning>Partly cloudy, Hazy</morning>
    Afternoon: <afternoon>Sunny &amp;amp; hot</afternoon>
    Evening: <evening>Clear and Cooler</evening>
</weatherReport>
```

Structure

- An XML document may start with one or more processing instructions (PIs) or directives:
  - <?xml version="1.0"?>
  - <?xml-stylesheet type="text/css" href="ss.css"?>
- Following the directives, there must be exactly *one* root element containing all the rest of the XML:

```
<weatherReport>
...
</weatherReport>
```

Building Blocks

- Aside from the directives, an XML document is built from:
  - elements: high in <high scale="F">103</high>
  - tags, in pairs: <high scale="F">103</high>
  - attributes: <high scale="F">103</high>
  - entities: <afternoon>Sunny & hot</afternoon>
  - character data, which may be:
    - parsed (processed as XML)--this is the default
    - unparsed (all characters stand for themselves)

Elements & Attributes

- Attributes and elements are somewhat interchangeable
- Example using just elements:

```
<name>
    <first>Wahyu</first>
    <last>Wibowo</last>
</name>
```

- Example using attributes:

```
<name first="Wahyu" last="Wibowo"></name>
```

- You will find that elements are easier to use in your programs-- this is a good reason to prefer them
  - Attributes often contain metadata, such as unique IDs
  - Generally speaking, browsers display only elements (values enclosed by tags), not tags and attributes

Well-formed XML

- Every element must have *both* a start tag and an end tag, e.g.

  <name> ... </name>
- But empty elements can be abbreviated: <break />.
- XML tags are case sensitive
- XML tags may not begin with the letters xml, in any combination of cases
- Elements must be properly nested, e.g. *not* bold and italic
- Every XML document must have one and only one root element
- The values of attributes must be enclosed in single or double quotes, e.g. <time unit="days">
- Character data cannot contain < or &

Entities

- Five special characters must be written as entities:

  & for & (almost always necessary)

  < for < (almost always necessary)

  > for > (not usually necessary)

  " for " (necessary inside double quotes)

  ' for ' (necessary inside single quotes)

- These entities can be used even in places where they are not absolutely required
- These are the *only* predefined entities in XML

XML Declaration

- The XML declaration looks like this:

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

- The XML declaration is not required by browsers, but *is* required by most XML processors (so include it!)
- If present, the XML declaration must be first--*not even whitespace* should precede it
- Note that the brackets are <? and ?>
- version="1.0" is required (this is the *only* version so far)
- encoding can be "UTF-8" (ASCII) or "UTF-16" (Unicode), or something else, or it can be omitted
- standalone tells whether there is a separate DTD

Processing Instruction

- PIs (Processing Instructions) may occur anywhere in the XML document (but usually first)
- A PI is a command to the program processing the XML document to handle it in a certain way
- XML documents are typically processed by more than one program
- Programs that do not recognize a given PI should just ignore it
- General format of a PI: <?*target instructions*?>
- Example: <?xml-stylesheet type="text/css“ href="mySheet.css"?>

Structuring Data

- XML comments begin with `<!--` and end with `-->`
- An XML document contains text that represents its content (i.e., data) and elements that specify its structure. XML documents delimit an element with start and end tags
- The root element of an XML document encompasses all its other elements
- XML element names can be of any length and can contain letters, digits, underscores, hyphens and periods
  - Must begin with either a letter or an underscore, and they should not begin with "`xml`" in any combination of uppercase and lowercase letters, as this is reserved for use in the XML standards

Structuring Data (Cont.)

- When a user loads an XML document in a browser, a parser parses the document, and the browser uses a style sheet to format the data for display
- IE and Firefox each display minus (−) or plus (+) signs next to all container elements. A minus sign indicates that all child elements are being displayed. When clicked, a minus sign becomes a plus sign (which collapses the container element and hides all the children), and vice versa
- Data can be placed between tags or in attributes (name/value pairs that appear within the angle brackets of start tags). Elements can have any number of attributes

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

<!-- Article -->
<article>
	<title>Simple XML</title>
	<date>September 11, 2014</date>
	<author>
		<firstname>John</firstname>
		<lastname>Widodo</lastname>
	</author>
	<summary>XML is fun</summary>
	<content>Thank you</content>
</article>
```

CDATA

- By default, all text inside an XML document is parsed
- You can force text to be treated as unparsed *character data* by enclosing it in <![CDATA[...]]>
- Any characters, even & and <, can occur inside a CDATA
- Whitespace inside a CDATA is (usually) preserved
- The only real restriction is that the character sequence]]> cannot occur inside a CDATA
- CDATA is useful when your text has a lot of illegal characters (for example, if your XML document contains some HTML text)

Common Programming Error

XML is case sensitive. Using different cases for the start tag and end tag names for the same element is a syntax error.

Common Programming Error

Using a white-space character in an XML element name is an error.

Good Programming Practice

XML element names should be meaningful to humans and should not use abbreviations.

Common Programming Error

Nesting XML tags improperly is a syntax error. For example, `<x><y>hello</x></y>` is an error, because the `</y>` tag must precede the `</x>` tag.

Outline

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 14.4: letter.xml -->
4  <!-- Business letter marked up as XML -->
5  <!DOCTYPE letter SYSTEM "letter.dtd">
6
7  <letter>
8     <contact type = "sender">
9        <name>Jane Doe</name>
10       <address1>Box 12345</address1>
11       <address2>15 Any Ave.</address2>
12       <city>Othertown</city>
13       <state>Otherstate</state>
14       <zip>67890</zip>
15       <phone>555-4321</phone>
16       <flag gender = "F" />
17    </contact>
18
19    <contact type = "receiver">
20       <name>John Doe</name>
21       <address1>123 Main St.</address1>
22       <address2></address2>
23       <city>Anytown</city>
24       <state>Anystate</state>
25       <zip>12345</zip>
26       <phone>555-1234</phone>
27       <flag gender = "M" />
28    </contact>
29
30    <salutation>Dear Sir:</salutation>
```

The `DOCTYPE` specifies an external DTD in the file `letter.dtd`

Data can be stored as attributes, which appear in an element's start tag

`flag` is an empty element because it contains no child elements or content

```
31
32    <paragraph>It is our privilege to inform you about our new database
33       managed with XML. This new system allows you to reduce the
34       load on your inventory list server by having the client machine
35       perform the work of sorting and filtering the data.
36    </paragraph>
37
38    <paragraph>Please visit our website for availability and pricing.
39    </paragraph>
40
41    <closing>Sincerely,</closing>
42    <signature>Ms. Jane Doe</signature>
43 </letter>
```

Outline

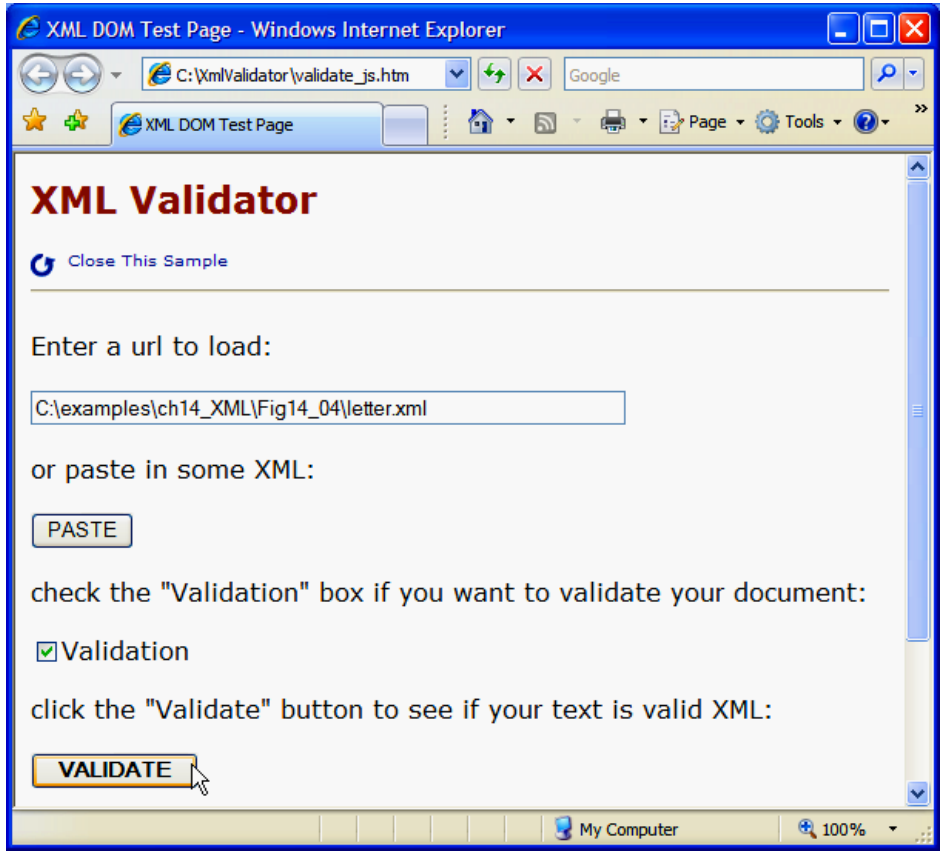
Error-Prevention Tip

An XML document is not required to reference a DTD, but validating XML parsers can use a DTD to ensure that the document has the proper structure.

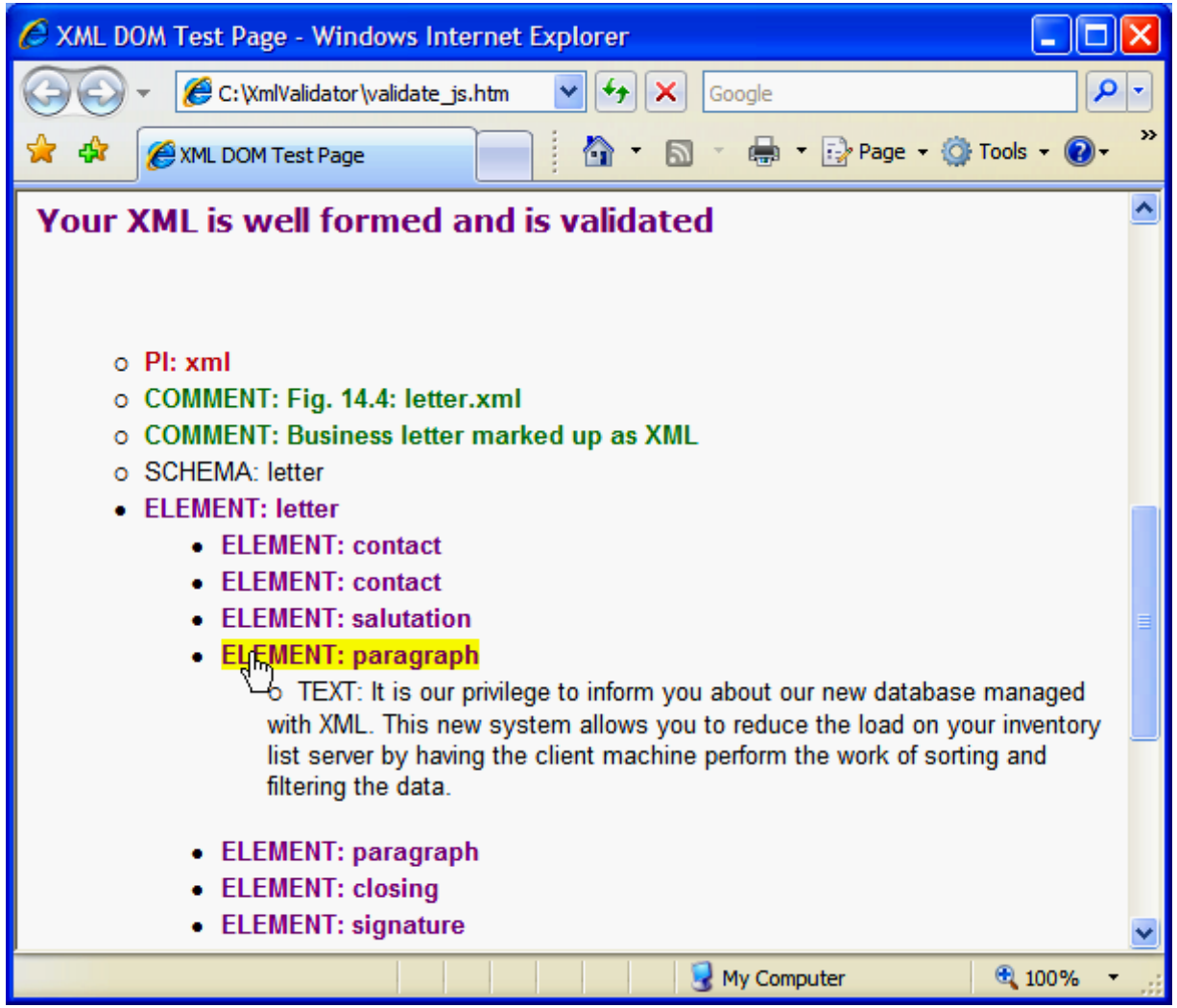
Portability Tip

Validating an XML document helps guarantee that independent developers will exchange data in a standardized form that conforms to the DTD.

http://www.w3schools.com/xml/xml_validator.asp



Validating an XML document with Microsoft's XML Validator.



Validation result using Microsoft's XML Validator.

Common Programming Error

Failure to enclose attribute values in double ("") or single (' ') quotes is a syntax error.

Namespaces

- XML namespaces provide a means for document authors to prevent naming collisions
- Each namespace prefix is bound to a uniform resource identifier (URI) that uniquely identifies the namespace
  - A URI is a series of characters that differentiate names
  - Document authors create their own namespace prefixes
  - Any name can be used as a namespace prefix, but the namespace prefix `xml` is reserved for use in XML standards
- To eliminate the need to place a namespace prefix in each element, authors can specify a default namespace for an element and its children
  - We declare a default namespace using keyword `xmlns` with a URI (Uniform Resource Identifier) as its value
- Document authors commonly use URLs (Uniform Resource Locators) for URIs, because domain names (e.g., `deitel.com`) in URLs must be unique

Common Programming Error

Attempting to create a namespace prefix named `xml` in any mixture of uppercase and lowercase letters is a syntax error—the `xml` namespace prefix is reserved for internal use by XML itself.

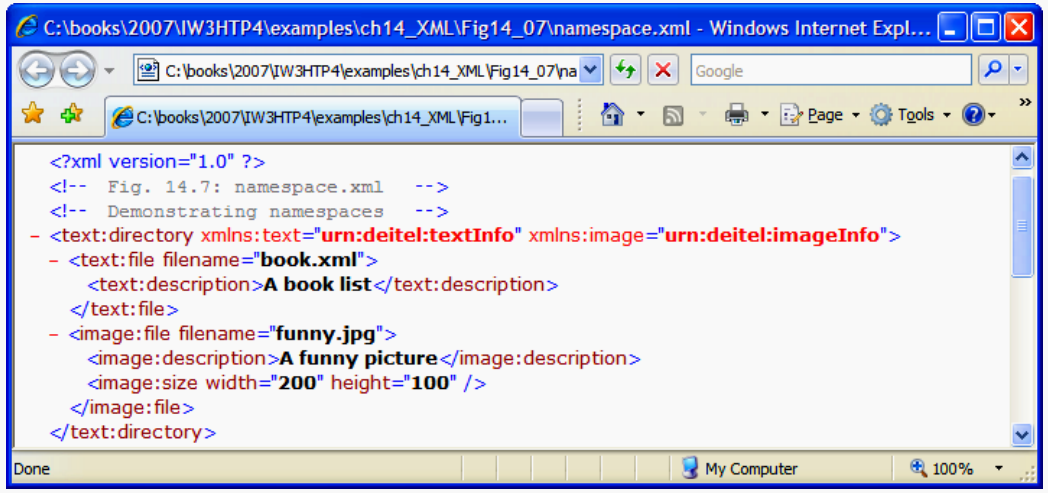
Outline

namespace.xml

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 14.7: namespace.xml -->
4  <!-- Demonstrating namespaces -->
5  <text:directory
6     xmlns:text = "urn:deitel:textInfo"
7     xmlns:image = "urn:deitel:imageInfo">
8
9     <text:file filename = "book.xml">
10       <text:description>A book list</text:description>
11    </text:file>
12
13    <image:file filename = "funny.jpg">
14       <image:description>A funny picture</image:des
15       <image:size width = "200" height = "100" />
16    </image:file>
17 </text:directory>
```

Two namespaces are specified using URNs

The namespace prefixes are used in element names throughout the document



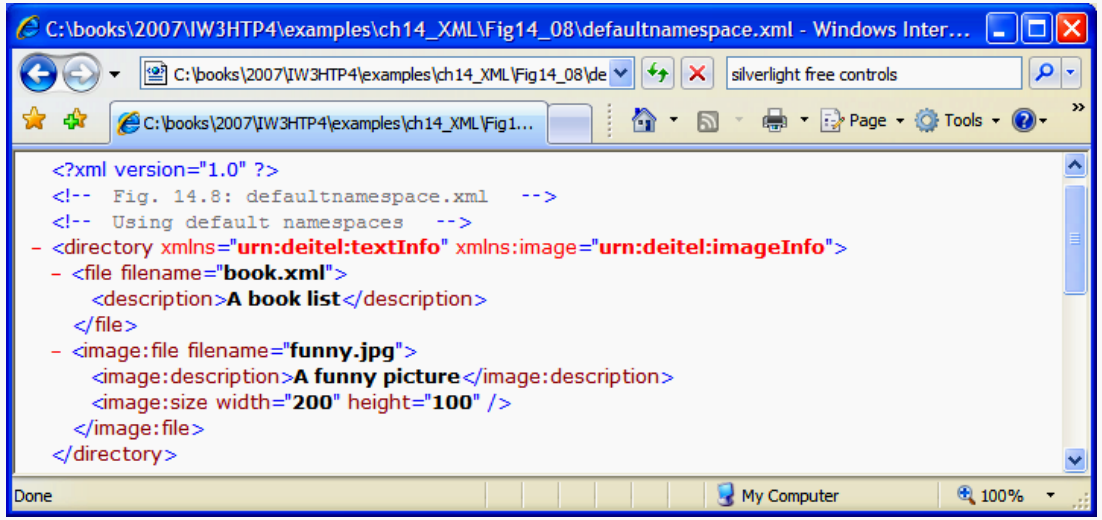
Outline

defaultnamespace.xml

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 14.8: defaultnamespace.xml -->
4  <!-- Using default namespaces -->
5  <directory xmlns = "urn:deitel:textInfo"
6     xmlns:image = "urn:deitel:imageInfo">
7
8     <file filename = "book.xml">
9        <description>A book list</description>
10    </file>
11
12    <image:file filename = "funny.jpg">
13       <image:description>A funny picture</image:description>
14       <image:size width = "200" height = "100" />
15    </image:file>
16 </directory>
```

The default namespace is set in the `directory` element

Elements with no namespace prefix use the default namespace



Document Type Definitions (DTDs)

- DTDs and schemas specify documents' element types and attributes, and their relationships to one another
- DTDs and schemas enable an XML parser to verify whether an XML document is valid (i.e., its elements contain the proper attributes and appear in the proper sequence)
- A DTD expresses the set of rules for document structure using an EBNF (Extended Backus-Naur Form) grammar
- In a DTD, an `ELEMENT` element type declaration defines the rules for an element. An `ATTLIST` attribute-list declaration defines attributes for a particular element

Software Engineering Observation

XML documents can have many different structures, and for this reason an application cannot be certain whether a particular document
it receives is complete, ordered properly, and not missing data. DTDs and schemas solve this problem by providing an extensible way to describe XML document structure. Applications should use DTDs or schemas to confirm whether XML documents are valid.

Software Engineering Observation

Many organizations and individuals are creating DTDs and schemas for a broad range of applications. These collections—called repositories—are available free for download from the web (e.g., `www.xml.org`, `www.oasis-open.org`).

Outline

letter.dtd

```
1  <!-- Fig. 14.9: letter.dtd       -->
2  <!-- DTD document for letter.xml -->
3
4  <!ELEMENT letter ( contact+, salutation, paragraph+,
5     closing, signature )>
6
7  <!ELEMENT contact ( name, address1, address2, city, state,
8     zip, phone, flag )>
9  <!ATTLIST contact type CDATA #IMPLIED>
10
11 <!ELEMENT name ( #PCDATA )>
12 <!ELEMENT address1 ( #PCDATA )>
13 <!ELEMENT address2 ( #PCDATA )>
14 <!ELEMENT city ( #PCDATA )>
15 <!ELEMENT state ( #PCDATA )>
16 <!ELEMENT zip ( #PCDATA )>
17 <!ELEMENT phone ( #PCDATA )>
18 <!ELEMENT flag EMPTY>
19 <!ATTLIST flag gender (M | F) "M">
20
21 <!ELEMENT salutation ( #PCDATA )>
22 <!ELEMENT closing ( #PCDATA )>
23 <!ELEMENT paragraph ( #PCDATA )>
24 <!ELEMENT signature ( #PCDATA )>
```

Define the requirements for the `letter` element

Define the requirements for the `contact` element

A `contact` element may have a `type` attribute, but it is not required

Each of these elements contains parsed character data

The `flag` element must be empty and its `gender` attribute must be set to either `M` or `F`. If there is no `gender` attribute, gender defaults to `M`

Common Programming Error

For documents validated with DTDs, any document that uses elements, attributes or nesting relationships not explicitly defined by a DTD is an invalid document.

Software Engineering Observation

DTD syntax cannot describe an element's or attribute's data type. For example, a DTD cannot specify that a particular element or attribute can contain only integer data.

W3C XML Schema Documents

- Unlike DTDs
  - Schemas use use XML syntax not EBNF grammar
  - XML Schema documents can specify what type of data (e.g., numeric, text) an element can contain
- An XML document that conforms to a schema document is schema valid
- Two categories of types exist in XML Schema: simple types and complex types
  - Simple types cannot contain attributes or child elements; complex types can
- Every simple type defines a restriction on an XML Schema-defined schema type or on a user-defined type
- Complex types can have either simple content or complex content
  - Both can contain attributes, but only complex content can contain child elements
- Whereas complex types with simple content must extend or restrict some other existing type, complex types with complex content do not have this limitation

Outline

book.xml

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 14.11: book.xml -->
4  <!-- Book list marked up as XML -->
5  <deitel:books xmlns:deitel = "http://www.deitel.com/booklist">
6     <book>
7        <title>Visual Basic 2005 How to Program, 3/e</title>
8     </book>
9     <book>
10       <title>Visual C# 2005 How to Program, 2/e</title>
11    </book>
12    <book>
13       <title>Java How to Program, 7/e</title>
14    </book>
15    <book>
16       <title>C++ How to Program, 6/e</title>
17    </book>
18    <book>
19       <title>Internet and World Wide Web How to Program, 4/e</title>
20    </book>
21 </deitel:books>
```

Outline

book.xsd

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 14.12: book.xsd          -->
4  <!-- Simple W3C XML Schema document -->
5  <schema xmlns = "http://www.w3.org/2001/XMLSchema"
6     xmlns:deitel = "http://www.deitel.com/booklist"
7     targetNamespace = "http://www.deitel.com/booklist">
8
9     <element name = "books" type = "deitel:BooksType"/>
10
11    <complexType name = "BooksType">
12       <sequence>
13          <element name = "book" type = "deitel:SingleBookType"
14             minOccurs = "1" maxOccurs = "unbounded"/>
15       </sequence>
16    </complexType>
17
18    <complexType name = "SingleBookType">
19       <sequence>
20          <element name = "title" type = "string"/>
21       </sequence>
22    </complexType>
23 </schema>
```

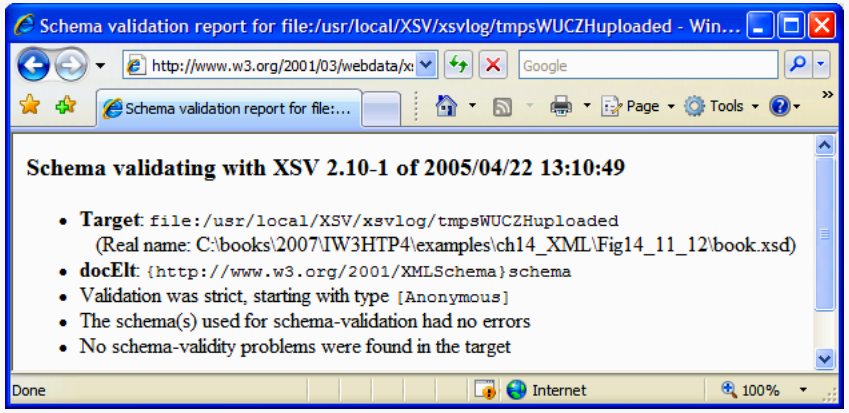
Specify the namespace of the elem

Define the `books` element

Define the requirements for any element of type BooksType

An element of type `BooksType` must contian one or more book elements, which have type `SingleBookType`

A `SingleBookType` element has a `title` element, which contains a `string`



Portability Tip

W3C XML Schema authors specify URI `http://www.w3.org/2001/XMLSchema` when referring to the XML Schema namespace. This namespace contains predefined elements that comprise the XML Schema vocabulary. Specifying this URI ensures that validation tools correctly identify XML Schema elements and do not confuse them with those defined by document authors.

Outline

computer.xsd

(1 of 2)

```
1  <?xml version = "1.0"?>
2  <!-- Fig. 14.14: computer.xsd -->
3  <!-- W3C XML Schema document  -->
4
5  <schema xmlns = "http://www.w3.org/2001/XMLSchema"
6     xmlns:computer = "http://www.deitel.com/computer"
7     targetNamespace = "http://www.deitel.com/computer">
8
9     <simpleType name = "gigahertz">
10       <restriction base = "decimal">
11          <minInclusive value = "2.1"/>
12       </restriction>
13    </simpleType>
14
15    <complexType name = "CPU">
16       <simpleContent>
17          <extension base = "string">
18             <attribute name = "model" type = "string"/>
19          </extension>
20       </simpleContent>
21    </complexType>
22
```

Define a `simpleType` that contains a decimal whose value is `2.1` or greater

Define a `complexType` with simpleContent so that it can contain only attributes, not child elements

The `CPU` element's data must be of type `string`, and it must have an attribute `model` containing a `string`

```
23    <complexType name = "portable">
24       <all>
25          <element name = "processor" type = "computer:CPU"/>
26          <element name = "monitor" type = "int"/>
27          <element name = "CPUSpeed" type = "computer:gigahertz"/>
28          <element name = "RAM" type = "int"/>
29       </all>
30       <attribute name = "manufacturer" type = "string"/>
31    </complexType>
32
33    <element name = "laptop" type = "computer:portable"/>
34 </schema>
```

Outline

computer.xsd

(2 of 2)

The all element specifies a list of elements that must be included, in any order, in the document

The types defined in the last slide are used by these elements

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 14.15: laptop.xml                -->
4  <!-- Laptop components marked up as XML -->
5  <computer:laptop xmlns:computer = "http://www.deitel.com/computer"
6     manufacturer = "IBM">
7
8     <processor model = "Centrino">Intel</processor>
9     <monitor>17</monitor>
10    <CPUSpeed>2.4</CPUSpeed>
11    <RAM>256</RAM>
12 </computer:laptop>
```

XML Vocabularies

- Some XML vocabularies
  - MathML (Mathematical Markup Language)
  - Scalable Vector Graphics (SVG)
  - Wireless Markup Language (WML)
  - Extensible Business Reporting Language (XBRL)
  - Extensible User Interface Language (XUL)
  - Product Data Markup Language (PDML)
  - W3C XML Schema
  - Extensible Stylesheet Language (XSL)
- MathML markup describes mathematical expressions for display
  - Divided into two types of markup—content markup and presentation markup
  - Content MathML allows programmers to write mathematical notation specific to different areas of mathematics
  - Presentation MathML is directed toward formatting and displaying mathematical notation
  - By convention, MathML files end with the `.mml` filename extension

XML Vocabularies (Cont.)

- MathML document root node is the `math` element
  - Default namespace is `http:// www.w3.org/1998/Math/MathML`
- `mn` element
  - marks up a number
- `mo` element
  - marks up an operator
- Entity reference `&InvisibleTimes;`
  - indicates a multiplication operation without explicit symbolic representation
- `msup` element
  - represents a superscript
  - has two children—the expression to be superscripted (i.e., the base) and the superscript (i.e., the exponent)
  - Correspondingly, the `msub` element represents a subscript
- To display variables, use identifier element `mi`

XML Vocabularies (Cont.)

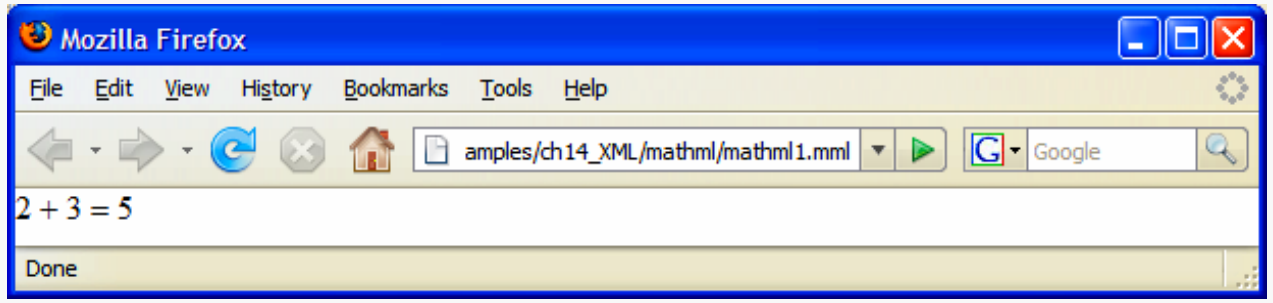
- `mfrac` element
  - displays a fraction
  - If either the numerator or the denominator contains more than one element, it must appear in an `mrow` element
- `mrow` element
  - groups elements that are positioned horizontally in an expression
- Entity reference `&int;`
  - represents the integral symbol
- `msubsup` element
  - specifies the subscript and superscript of a symbol
  - Requires three child elements—an operator, the subscript expression and the superscript expression
- `msqrt` element
  - represents a square-root expression
- Entity reference `&delta;`
  - represents a lowercase delta symbol

Outline

mathml1.mml

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
3     "http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
4
5  <!-- Fig. 14.16: mathml1.mml -->
6  <!-- MathML equation -->
7  <math xmlns="http://www.w3.org/1998/Math/
8     <mn>2</mn>
9     <mo>+</mo>
10    <mn>3</mn>
11    <mo>=</mo>
12    <mn>5</mn>
13 </math>
```

The `math` element contains number and operator elements that represent the equation `2 + 3 = 5`



Outline

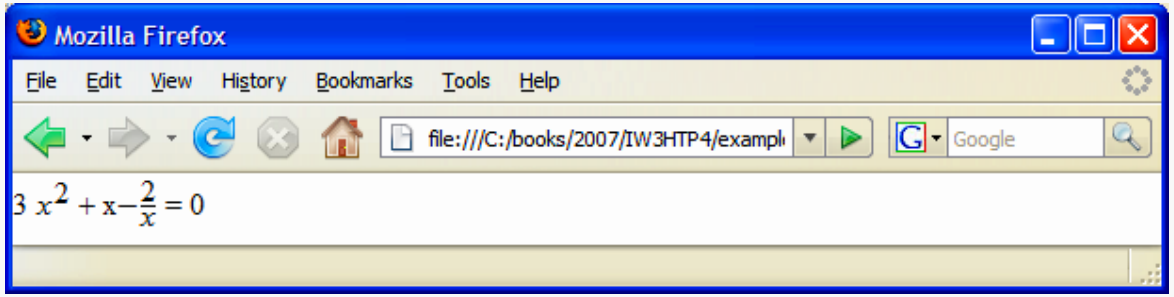
mathml2.html

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
3     "http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
4
5  <!-- Fig. 14.17: mathml2.html -->
6  <!-- MathML algebraic equation. -->
7  <math xmlns="http://www.w3.org/1998/Math/MathML">
8     <mn>3</mn>
9     <mo>&InvisibleTimes;</mo>
10    <msup>
11       <mi>x</mi>
12       <mn>2</mn>
13    </msup>
14    <mo>+</mo>
15    <mn>x</mn>
16    <mo>&minus;</mo>
17    <mfrac>
18       <mn>2</mn>
19       <mi>x</mi>
20    </mfrac>
21    <mo>=</mo>
22    <mn>0</mn>
23 </math>
```

`&InvisibleTimes;` **represents an implied multiplication**

The `msup` element contains two elements: a base and a superscript

The `mfrac` element contains two elements: a numerator and a denominator



Outline

mathml3.html

(1 of 2)

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
3     "http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
4
5  <!-- Fig. 14.18 mathml3.html -->
6  <!-- Calculus example using MathML -->
7  <math xmlns="http://www.w3.org/1998/Math/MathML">
8     <mrow>
9        <msubsup>
10          <mo>&int;</mo>
11          <mn>0</mn>
12          <mrow>
13             <mn>1</mn>
14             <mo>&minus;</mo>
15             <mi>y</mi>
16          </mrow>
17       </msubsup>
18       <msqrt>
19          <mn>4</mn>
20          <mo>&InvisibleTimes;</mo>
21          <msup>
22             <mi>x</mi>
23             <mn>2</mn>
24          </msup>
```

`&int;` **represents the integral symbol**

`mrow` **groups elements horizontally in an expression**

`msqrt` **puts its contents underneath a square root symbol**

Extensible Stylehsheet Language and XSL Transformations

- Convert XML into any text-based document
- XSL documents have the extension `.xsl`
- XPath
  - A string-based language of expressions used by XML and many of its related technologies for effectively and efficiently locating structures and data (such as specific elements and attributes) in XML documents
  - Used to locate parts of the source-tree document that match templates defined in an XSL style sheet. When a match occurs (i.e., a node matches a template), the matching template executes and adds its result to the result tree. When there are no more matches, XSLT has transformed the source tree into the result tree.
- XSLT does not analyze every node of the source tree
  - it selectively navigates the source tree using XPath's `select` and `match` attributes
- For XSLT to function, the source tree must be properly structured
  - Schemas, DTDs and validating parsers can validate document structure before using XPath and XSLTs
- XSL style sheets can be connected directly to an XML document by adding an `xml:stylesheet` processing instruction to the XML document

Extensible Stylehsheet Language and XSL Transformations (Cont.)

- Two tree structures are involved in transforming an XML document using XSLT
  - source tree (the document being transformed)
  - result tree (the result of the transformation)
- XPath character `/` (a forward slash)
  - Selects the document root
  - In XPath, a leading forward slash specifies that we are using absolute addressing
  - An XPath expression with no beginning forward slash uses relative addressing
- XSL element `value-of`
  - Retrieves an attribute's value
  - The `@` symbol specifies an attribute node
- XSL node-set function `name`
  - Retrieves the current node's element name
- XSL node-set function `text`
  - Retrieves the text between an element's start and end tags
- The XPath expression `//*`
  - Selects all the nodes in an XML document

```
1  <?xml version = "1.0"?>
2  <?xml-stylesheet type = "text/xsl" href = "sports.xsl"?>
3
4  <!-- Fig. 14.20: sports.xml -->
5  <!-- Sports Database -->
6
7  <sports>
8     <game id = "783">
9        <name>Cricket</name>
10
11       <paragraph>
12          More popular among commonwealth nations.
13       </paragraph>
14    </game>
15
16    <game id = "239">
17       <name>Baseball</name>
18
19       <paragraph>
20          More popular in America.
21       </paragraph>
22    </game>
23
```

The `xml-stylesheet` declaration points to an XSL style sheet for this document

Outline

`sports.xml`

(1 of 2)

```
24    <game id = "418">
25       <name>Soccer (Futbol)</name>
26
27       <paragraph>
28          Most popular sport in the world.
29       </paragraph>
30    </game>
31 </sports>
```

Outline

sports.xml

(2 of 2)

Software Engineering Observation

XSL enables document authors to separate data presentation (specified in XSL documents) from data description (specified in XML documents).

Common Programming Error

You will sometimes see the XML processing instruction `<?xml-stylesheet?>` written as `<?xml:stylesheet?>` with a colon rather than a dash. The version with a colon results in an XML parsing error in Firefox.

Outline

sports.xsl

(1 of 2)

```
1  <?xml version = "1.0"?>
2  <!-- Fig. 14.21: sports.xsl -->
3  <!-- A simple XSLT transformation -->
4
5  <!-- reference XSL style sheet URI -->
6  <xsl-stylesheet version = "1.0"
7     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
8
9     <xsl:output method = "html" omit-xml-declaration = "no"
10       doctype-system =
11          "http://www.w3c.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
12       doctype-public = "-//W3C//DTD XHTML 1.0 Strict//EN"/>
13
14    <xsl:template match = "/"> <!-- match root element -->
15
16    <html xmlns = "http://www.w3.org/1999/xhtml">
17       <head>
18          <title>Sports</title>
19       </head>
20
21       <body>
22          <table border = "1" bgcolor = "wheat">
23             <thead>
24                <tr>
25                   <th>ID</th>
26                   <th>Sport</th>
27                   <th>Information</th>
28                </tr>
29             </thead>
30
```

Use `xsl-output` to write a doctype.

```
31            <!-- insert each name and paragraph element value -->
32            <!-- into a table row. -->
33            <xsl:for-each select = "/sports/game">
34               <tr>
35                  <td><xsl:value-of select = "@id"/></td>
36                  <td><xsl:value-of select = "name"/></td>
37                  <td><xsl:value-of select = "paragraph"/></td>
38               </tr>
39            </xsl:for-each>
40         </table>
41      </body>
42   </html>
43
44   </xsl:template>
45 </xsl:stylesheet>
```

Write the following HTML for each `game` element in the `sports` element that is contained in the root element

Write the value of the `game`'s `id` attribute in a table cell

Write the value of the `game`'s `name` child element in a table cell

Write the value of the `game`'s `paragraph` child element in a table cell

`sports.xsl`

(2 of 2)

Outline

sorting.xml

```
1  <?xml version = "1.0"?>
2  <?xml-stylesheet type = "text/xsl" href = "sorting.xsl"?>
3
4  <!-- Fig. 14.22: sorting.xml -->
5  <!-- XML document containing book information -->
6  <book isbn = "999-99999-9-X">
7     <title>Deitel&apos;s XML Primer</title>
8
9     <author>
10       <firstName>Jane</firstName>
11       <lastName>Blue</lastName>
12    </author>
13
14    <chapters>
15       <frontMatter>
16          <preface pages = "2" />
17          <contents pages = "5" />
18          <illustrations pages = "4" />
19       </frontMatter>
20
21       <chapter number = "3" pages = "44">Advanced XML</chapter>
22       <chapter number = "2" pages = "35">Intermediate XML</chapter>
23       <appendix number = "B" pages = "26">Parsers and Tools</appendix>
24       <appendix number = "A" pages = "7">Entities</appendix>
25       <chapter number = "1" pages = "28">XML Fundamentals</chapter>
26    </chapters>
27
28    <media type = "CD" />
29 </book>
```

The chapters are out of order here, but our XSL transformation will sort the data before displaying it

Outline

sorting.xsl

(1 of 4)

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 14.23: sorting.xsl -->
4  <!-- Transformation of book information into XHTML -->
5  <xsl:stylesheet version = "1.0"
6     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
7
8     <!-- write XML declaration and DOCTYPE DTD information -->
9     <xsl:output method = "html" omit-xml-declaration = "no"
10       doctype-system = "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"
11       doctype-public = "-//W3C//DTD XHTML 1.1//EN"/>
12
13    <!-- match document root -->
14    <xsl:template match = "/">
15       <html xmlns = "http://www.w3.org/1999/xhtml">
16          <xsl:apply-templates/>
17       </html>
18    </xsl:template>
19
20    <!-- match book -->
21    <xsl:template match = "book">
22       <head>
23          <title>ISBN <xsl:value-of select = "@isbn"/> -
24             <xsl:value-of select = "title"/></title>
25       </head>
26
```

Apply the templates in this document to the document root's child nodes

Set the XHTML document's title to show the book's ISBN number and title

```
27        <body>
28           <h1 style = "color: blue"><xsl:value-of select = "title"/></h1>
29           <h2 style = "color: blue">by
30              <xsl:value-of select = "author/lastName"/>,
31              <xsl:value-of select = "author/firstName"/></h2>
32
33           <table style = "border-style: groove; background-color: wheat">
34
35              <xsl:for-each select = "chapters/frontMatter/*">
36                 <tr>
37                    <td style = "text-align: right">
38                       <xsl:value-of select = "name()"/>
39                    </td>
40
41                    <td>
42                       ( <xsl:value-of select = "@pages"/> pages )
43                    </td>
44                 </tr>
45              </xsl:for-each>
46
47              <xsl:for-each select = "chapters/chapter">
48                 <xsl:sort select = "@number" data-type = "number"
49                      order = "ascending"/>
50                 <tr>
51                    <td style = "text-align: right">
52                       Chapter <xsl:value-of select = "@number"/>
53                    </td>
54
```

Outline

sorting.xsl

(2 of 4)

Output the enclosed XHTML for each element in the `frontmatter` element, which is contained in the `chapters` element

Sort the chapter elements by the `number` contained in their `number` attribute in ascending order

```
55                   <td>
56                      <xsl:value-of select = "text()"/>
57                      ( <xsl:value-of select = "@pages"/> pages )
58                   </td>
59                </tr>
60             </xsl:for-each>
61
62             <xsl:for-each select = "chapters/appendix">
63                <xsl:sort select = "@number" data-type = "text"
64                   order = "ascending"/>
65                <tr>
66                   <td style = "text-align: right">
67                      Appendix <xsl:value-of select = "@number"/>
68                   </td>
69
70                   <td>
71                      <xsl:value-of select = "text()"/>
72                      ( <xsl:value-of select = "@pages"/> pages )
73                   </td>
74                </tr>
75             </xsl:for-each>
76          </table>
77
```

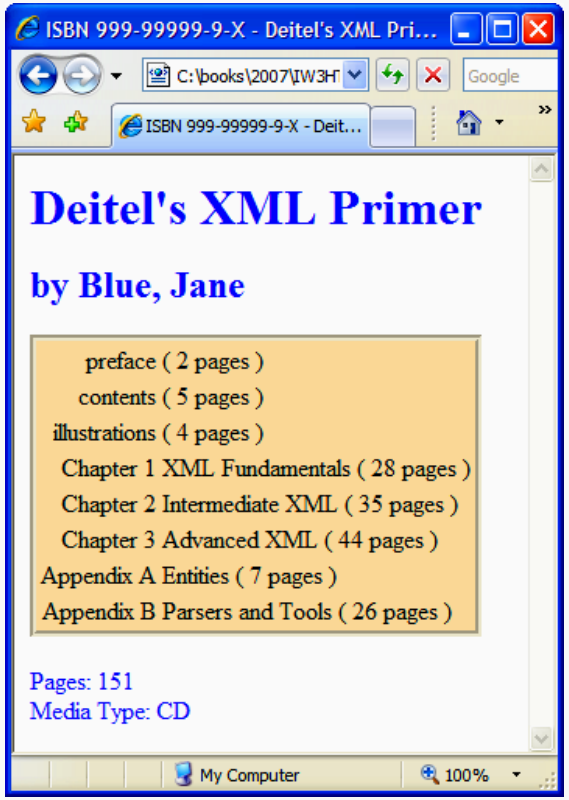
Outline

sorting.xsl

(3 of 4)

Sort the appendices alphabetically by their `number` attributes in ascending order

```
78          <br /><p style = "color: blue">Pages:
79             <xsl:variable name = "pagecount"
80                select = "sum(chapters//*/@pages)"/>
81             <xsl:value-of select = "$pagecount"/>
82          <br />Media Type: <xsl:value-of select = "media/@type"/></p>
83       </body>
84    </xsl:template>
85 </xsl:stylesheet>
```



Outline

sorting.xsl

(4 of 4)

Document Object Model

- Retrieving data from an XML document using traditional sequential file processing techniques is neither practical nor efficient
- Some XML parsers store document data as tree structures in memory
  - This hierarchical tree structure is called a Document Object Model (DOM) tree, and an XML parser that creates this type of structure is known as a DOM parser
  - Each element name is represented by a node
  - A node that contains other nodes is called a parent node
  - A parent node can have many children, but a child node can have only one parent node
  - Nodes that are peers are called sibling nodes
  - A node's descendant nodes include its children, its children's children and so on
  - A node's ancestor nodes include its parent, its parent's parent and so on

Document Object Model (Cont.)

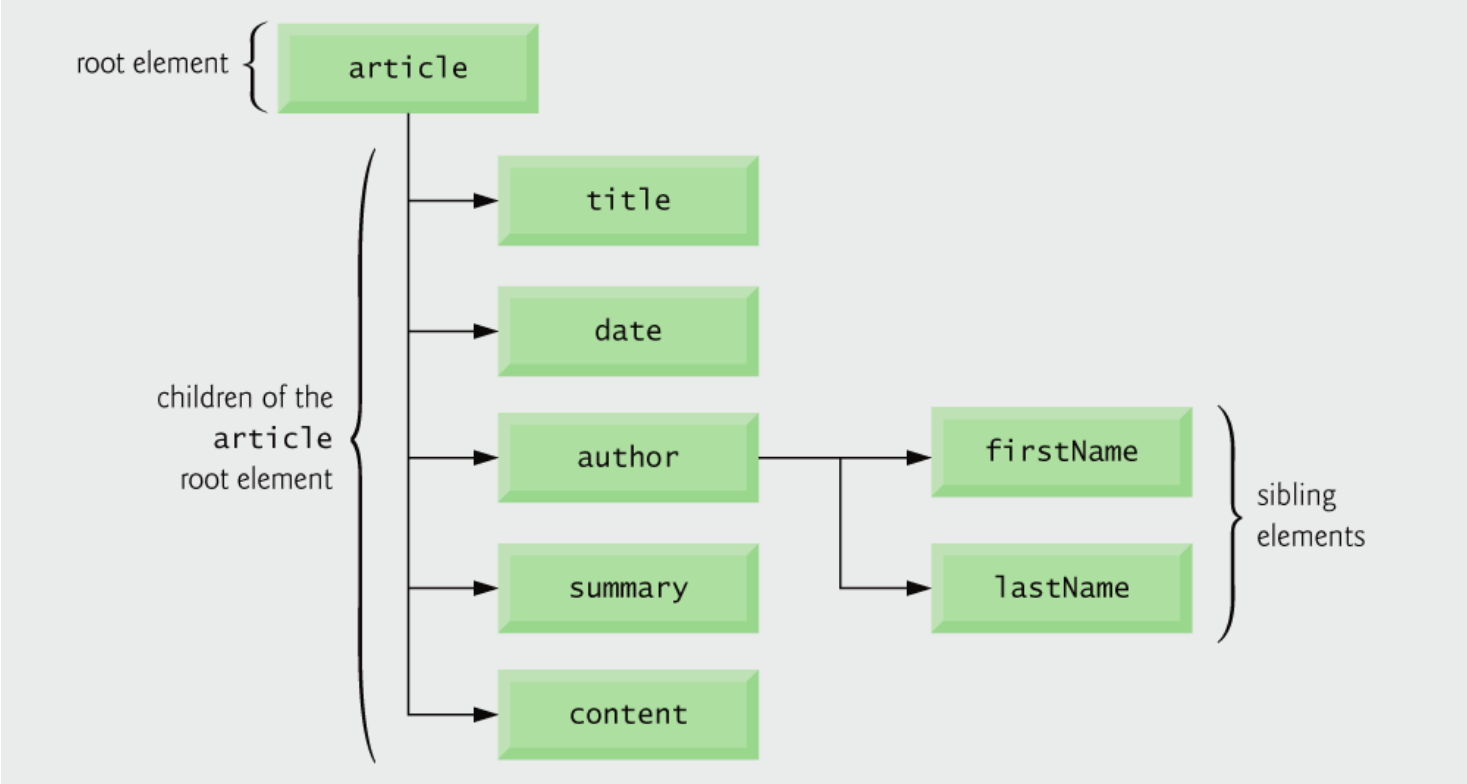
- Many of the XML DOM capabilities are similar or identical to those of the XHTML DOM
- The DOM tree has a single root node, which contains all the other nodes in the document
- `window.ActiveXObject`
  - If this object exists, the browser is Internet Explorer
  - Loads Microsoft's MSXML parser is used to manipulate XML documents in Internet Explorer
- MSXML `load` method
  - loads an XML document
- `childNodes` property of a document
  - contains a list of the XML document's top-level nodes
- If the browser is Firefox 2, then the `document` object's `implementation` property and the `implementation` property's `createDocument` method will exist
- Firefox loads each XML document asynchronously
  - You must use the XML document's `onload` property to specify a function to call when the document finishes loading to ensure that you can access the document's contents
- `nodeType` property of a node
  - contains the type of the node

Document Object Model (Cont.)

- Nonbreaking spaces (` `)
  - spaces that the browser is not allowed to collapse or that can be used to keep words together.
- `nodeName` property of a node
  - Obtain the name of an element
- `childNodes` list of a node
  - Nonzero if the currrent node has children
- `nodeValue` property
  - Returns the value of an element
- `firstChild` property of a node
  - Refers to the first child of a given node
- `lastChild` property of a node
  - refers to the last child of a given node
- `nextSibling` property of a node
  - refers to the next sibling in a list of children of a particular node.
- `previousSibling` property of a node
  - refers to the current node's previous sibling
- `parentNode` property of a node
  - refers to the current node's parent node

Document Object Model (Cont.)

- Use XPath expressions to specify search criteria
  - In IE7, the XML document object's `selectNodes` method receives an XPath expression as an argument and returns a collection of elements that match the expression
  - Firefox 2 searches for XPath matches using the XML document object's `evaluate` method, which receives five arguments
    - the XPath expression
    - the document to apply the expression to
    - a namespace resolver
    - a result type
    - an `XPathResult` object into which to place the results
    - If the last argument is `null`, the function simply returns a new `XPathResult` object containing the matches
    - The namespace resolver argument can be `null` if you are not using XML namespace prefixes in the XPath processing



Tree structure for the document `article.xml`

```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 14.26: XMLDOMTraversal.html -->
6  <!-- Traversing an XML document using the XML DOM. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9     <title>Traversing an XML document using the XML DOM</title>
10    <style type = "text/css">
11       .highlighted { background-color: yellow }
12       #outputDiv { font: 10pt "Lucida Console", monospace; }
13    </style>
14    <script type="text/javascript">
15    <!--
16    var doc; // variable to reference the XML document
17    var outputHTML = ""; // stores text to output in outputDiv
18    var idCounter = 1; // used to create div IDs
19    var depth = -1; // tree depth is -1 to start
20    var current = null; // represents the current node for traversals
21    var previous = null; // represent prior node in traversals
22
23    // load XML document based on whether the browser is IE7 or Firefox 2
24    function loadXMLDocument( url )
25    {
26       if ( window.ActiveXObject ) // IE7
27       {
28          // create IE7-specific XML document object
29          doc = new ActiveXObject( "Msxml2.DOMDocument.6.0" );
30          doc.async = false; // specifies synchronous loading of XML doc
```

Load the document using the IE-specific `ActiveXObject`

Outline

XMLDOMTraversal.html

(1 of 13)

```
31          doc.load( url ); // load the XML document specified by url
32          buildHTML( doc.childNodes ); // display the nodes
33          displayDoc();
34       } // end if
35       else if ( document.implementation &&
36          document.implementation.createDocument ) // other browsers
37       {
38          // create XML document object
39          doc = document.implementation.createDocument( "", "", null );
40          doc.load( url ); // load the XML document specified by url
41          doc.onload = function() // function to execute when doc loads
42          {
43             buildHTML( doc.childNodes ); // called by XML doc onload event
44             displayDoc(); // display the HTML
45          } // end XML document's onload event handler
46       } // end else
47       else // not supported
48          alert( 'This script is not supported by your browser' );
49    } // end function loadXMLDocument
50
51    // traverse xmlDocument and build XHTML representation of its content
52    function buildHTML( childList )
53    {
54       ++depth; // increase tab depth
55
56       // display each node's content
```

Outline

XMLDOMTraversal.html

(2 of 13)

Load the document using other browsers

In either case, call `buildHTML` and `displayDoc` methods once the document has loaded

Outline

```
57       for ( var i = 0; i < childList.length; i++ )
58       {
59          switch ( childList[ i ].nodeType )
60          {
61             case 1: // Node.ELEMENT_NODE; value used for portability
62                outputHTML += "<div id=\"id" + idCounter + "\">";
63                spaceOutput( depth ); // insert spaces
64                outputHTML += childList[ i ].nodeName; // show node's name
65                ++idCounter; // increment the id counter
66
67                // if current node has children, call buildHTML recursively
68                if ( childList[ i ].childNodes.length != 0 )
69                   buildHTML( childList[ i ].childNodes );
70
71                outputHTML += "</div>";
72                break;
73             case 3: // Node.TEXT_NODE; value used for portability
74             case 8: // Node.COMMENT_NODE; value used for portability
75                // if nodeValue is not 3 or 6 spaces (Firefox issue),
76                // include nodeValue in HTML
77                if ( childList[ i ].nodeValue.indexOf( "    " ) == -1 &&
78                   childList[ i ].nodeValue.indexOf( "      " ) == -1 )
79                {
80                   outputHTML += "<div id=\"id" + idCounter + "\">";
81                   spaceOutput( depth ); // insert spaces
82                   outputHTML += childList[ i ].nodeValue + "</div>";
83                   ++idCounter; // increment the id counter
84                } // end if
85          } // end switch
86       } // end for
```

Loop through children and decide what to do depending on `nodeType`

.html

(3 of 13)

If the node is an element, display its name

If the node has children, call `buildHTML` **recursively to handle the children**

If a text node or comment node is not simply indentation white space, add its value to the `outputHTML`

```
87
88          --depth; // decrease tab depth
89       } // end function buildHTML
90
91       // display the XML document and highlight the first child
92       function displayDoc()
93       {
94          document.getElementById( "outputDiv" ).innerHTML = outputHTML;
95          current = document.getElementById( 'id1' );
96          setCurrentNodeStyle( current.id, true );
97       } // end function displayDoc
98
99       // insert non-breaking spaces for indentation
100      function spaceOutput( number )
101      {
102         for ( var i = 0; i < number; i++ )
103         {
104            outputHTML += "   ";
105         } // end for
106      } // end function spaceOutput
107
108      // highlight first child of current node
109      function processFirstChild()
110      {
111         if ( current.childNodes.length == 1 && // only one child
112            current.firstChild.nodeType == 3 )  // and it's a text node
113         {
114            alert( "There is no child node" );
115         } // end if
```

Insert three spaces for each level of indentation

Outline

XMLDOMTraversal.html

```
116      else if ( current.childNodes.length > 1 )
117      {
118         previous = current; // save currently highlighted node
119
120         if ( current.firstChild.nodeType != 3 ) // if not text node
121            current = current.firstChild; // get new current node
122         else // if text node, use firstChild's nextSibling instead
123            current = current.firstChild.nextSibling; // get first sibling
124
125         setCurrentNodeStyle( previous.id, false ); // remove highlight
126         setCurrentNodeStyle( current.id, true ); // add highlight
127      } // end if
128      else
129         alert( "There is no child node" );
130   } // end function processFirstChild
131
132   // highlight next sibling of current node
133   function processNextSibling()
134   {
135      if ( current.id != "outputDiv" && current.nextSibling )
136      {
137         previous = current; // save currently highlighted node
138         current = current.nextSibling; // get new current node
139         setCurrentNodeStyle( previous.id, false ); // remove highlight
140         setCurrentNodeStyle( current.id, true ); // add highlight
141      } // end if
142      else
143         alert( "There is no next sibling" );
144   } // end function processNextSibling
145
```

Outline

XMLDOMTraversal.html

(5 of 13)

Check that the first child of the current node exists, then select and highlight it

Select the next sibling of the current node, if it exists

```
146  // highlight previous sibling of current node if it is not a text node
147  function processPreviousSibling()
148  {
149     if ( current.id != "outputDiv" && current.previousSibling &&
150        current.previousSibling.nodeType != 3 )
151     {
152        previous = current; // save currently highlighted node
153        current = current.previousSibling; // get new current node
154        setCurrentNodeStyle( previous.id, false ); // remove highlight
155        setCurrentNodeStyle( current.id, true ); // add highlight
156     } // end if
157     else
158        alert( "There is no previous sibling" );
159  } // end function processPreviousSibling
160
161  // highlight last child of current node
162  function processLastChild()
163  {
164     if ( current.childNodes.length == 1 &&
165        current.lastChild.nodeType == 3 )
166     {
167        alert( "There is no child node" );
168     } // end if
169     else if ( current.childNodes.length != 0 )
170     {
171        previous = current; // save currently highlighted node
172        current = current.lastChild; // get new current node
173        setCurrentNodeStyle( previous.id, false ); // remove highlight
174        setCurrentNodeStyle( current.id, true ); // add highlight
175     } // end if
```

Outline

XMLDOMTraversal.html

(6 of 13)

Select the previous sibling of the current node, if it exists

Check that the last child of the current node exists, then select and highlight it

Outline

XMLDOMTraversal.html

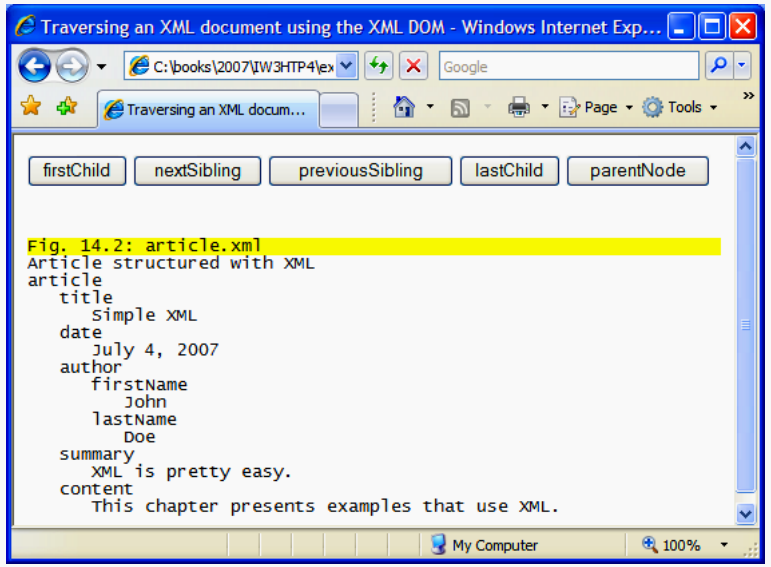
(7 of 13)

```
176       else
177          alert( "There is no child node" );
178    } // end function processLastChild
179
180    // highlight parent of current node
181    function processParentNode()
182    {
183       if ( current.parentNode.id != "body" )
184       {
185          previous = current; // save currently highlighted node
186          current = current.parentNode; // get new current node
187          setCurrentNodeStyle( previous.id, false ); // remove highlight
188          setCurrentNodeStyle( current.id, true ); // add highlight
189       } // end if
190       else
191          alert( "There is no parent node" );
192    } // end function processParentNode
193
194    // set style of node with specified id
195    function setCurrentNodeStyle( id, highlight )
196    {
197       document.getElementById( id ).className =
198          ( highlight ? "highlighted" : "" );
199    } // end function setCurrentNodeStyle
200    // -->
201    </script>
202 </head>
```

Select the parent of the current node

Method to highlight or remove highlighting from elements

```
<body id = "body" onload = "loadXMLDocument( 'article.xml' );">
   <form action = "" onsubmit = "return false;">
      <input type = "submit" value = "firstChild"
         onclick = "processFirstChild()"/>
      <input type = "submit" value = "nextSibling"
         onclick = "processNextSibling()"/>
      <input type = "submit" value = "previousSibling"
         onclick = "processPreviousSibling()"/>
      <input type = "submit" value = "lastChild"
         onclick = "processLastChild()"/>
      <input type = "submit" value = "parentNode"
         onclick = "processParentNode()"/>
   </form><br/>
   <div id = "outputDiv"></div>
</body>
</html>
```

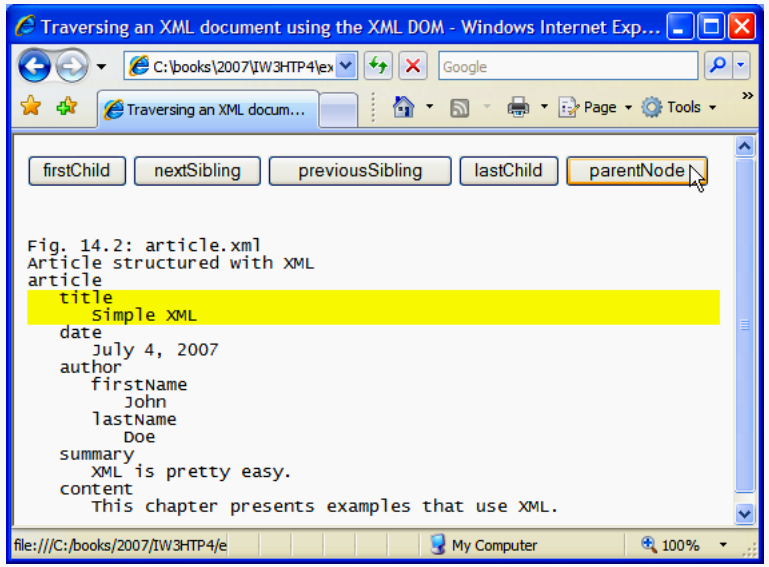


a) The comment node at the beginning of `article.xml` is highlighted when the XML document first loads.

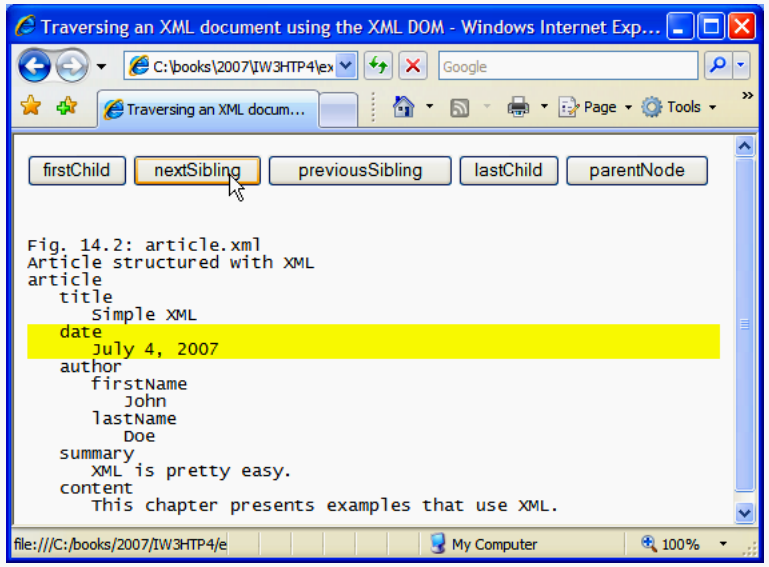
Outline

XMLDOMTraversal.html

(8 of 13)



f) User clicked the **parentNode** button to highlight the text node's parent `title` node.

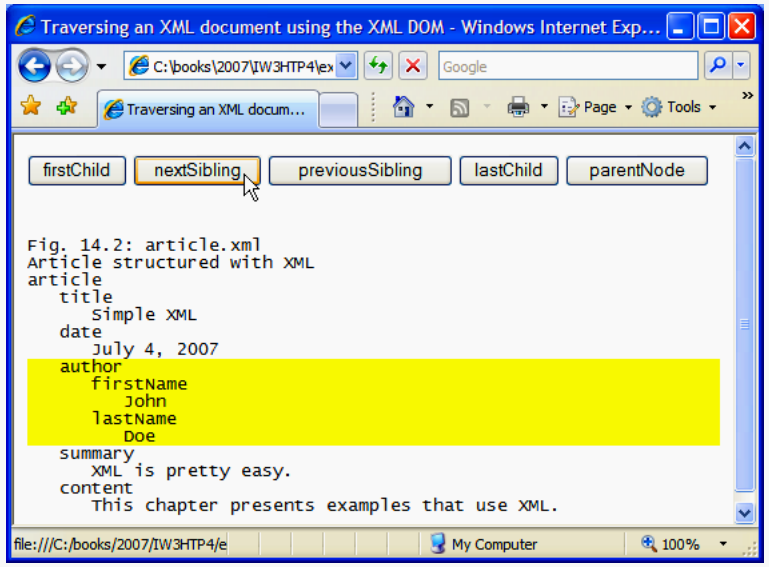


g) User clicked the **nextSibling** button to highlight the `title` node's `date` sibling node.

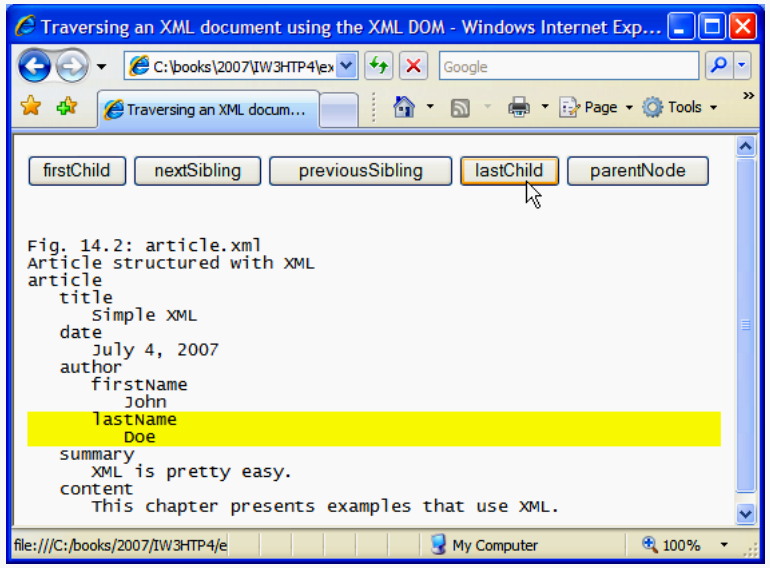
Outline

`XMLDOMTraversal.html`

(11 of 13)



h) User clicked the **nextSibling** button to highlight the `date` node's `author` sibling node.

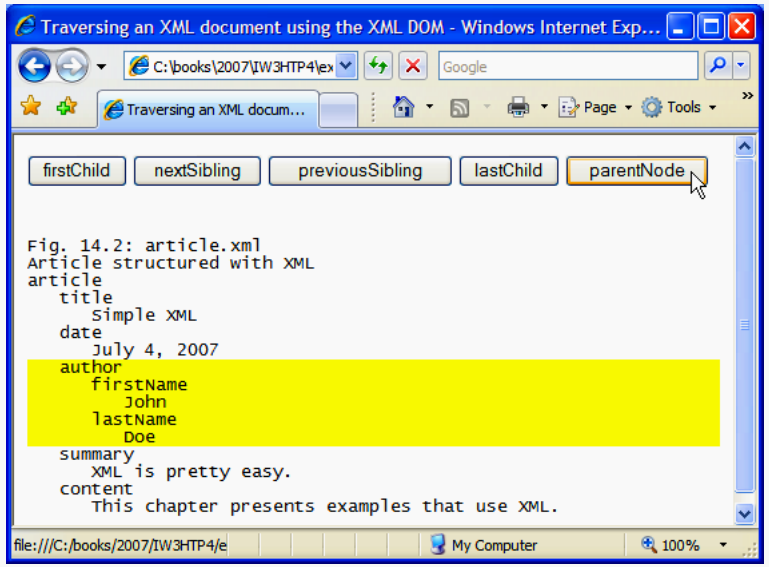


i) User clicked the **lastChild** button to highlight the `author` node's last child node (`lastName`).

Outline

`XMLDOMTraversal.html`

(12 of 13)



j) User clicked the **parentNode** button to highlight the `lastName` node's `author` parent node.

Outline

`XMLDOMTraversal.html`

(13 of 13)

Common Programming Error

Attempting to process the contents of a dynamically loaded XML document in Firefox before the document's `onload` event fires is a logic error. The document's contents are not available until the `onload` event fires.

Portability Tip

Firefox’s XML parser does not ignore white space used for indentation in XML documents. Instead, it creates text nodes containing the white-space characters.

XML Parser

- **A program that analyses the grammatical structure of an input, with respect to a given formal grammar**
- The parser determines how a sentence can be constructed from the grammar of the language by describing the atomic elements of the input and the relationship among them

SAX & DOM

- **SAX (Simple API for XML)**
  - event-driven parsing
  - “serial access” protocol
  - Read only API

- **DOM (Document Object Model)**
  - convert XML into a tree of objects
  - “random access” protocol
  - Can update XML document (insert/delete nodes)

SAX

- SAX = Simple API for XML
- XML is read sequentially
- When a *parsing event* happens, the parser invokes the corresponding method of the corresponding handler
- The handlers are programmer's implementation of standard Java API (i.e., interfaces and classes)
- Similar to an I/O-Stream, goes in one direction

Sample Data

Orders Data in XML:
several orders, each with several items
each item has a part number and a quantity

```
<orders>
<order>
 <onum>1020</onum>
 <takenBy>1000</takenBy>
 <customer>1111</customer>
 <recDate>10-DEC 94</recDate>
 <items>
   <item>
     <pnum>10506</pnum>
     <quantity>1</quantity>
   </item>
   <item>
     <pnum>10507</pnum>
     <quantity>1</quantity>
   </item>
   <item>
     <pnum>10508</pnum>
     <quantity>2</quantity>
   </item>
   <item>
     <pnum>10509</pnum>
     <quantity>3</quantity>
   </item>
 </items>
</order>
...
</orders>
```

Sample Data

Orders Data in XML:
several orders, each with several items
each item has a part number and a quantity

Parsing Event

startDocument

```
<orders>
  <order>
    <onum>1020</onum>
    <takenBy>1000</takenBy>
    <customer>1111</customer>
    <recDate>10-DEC 94</recDate>
    <items>
      <item>
        <pnum>10506</pnum>
        <quantity>1</quantity>
      </item>
      <item>
        <pnum>10507</pnum>
        <quantity>1</quantity>
      </item>
      <item>
        <pnum>10508</pnum>
        <quantity>2</quantity>
      </item>
      <item>
        <pnum>10509</pnum>
        <quantity>3</quantity>
      </item>
    </items>
  </order>
  ...
</orders>
```

endDocument

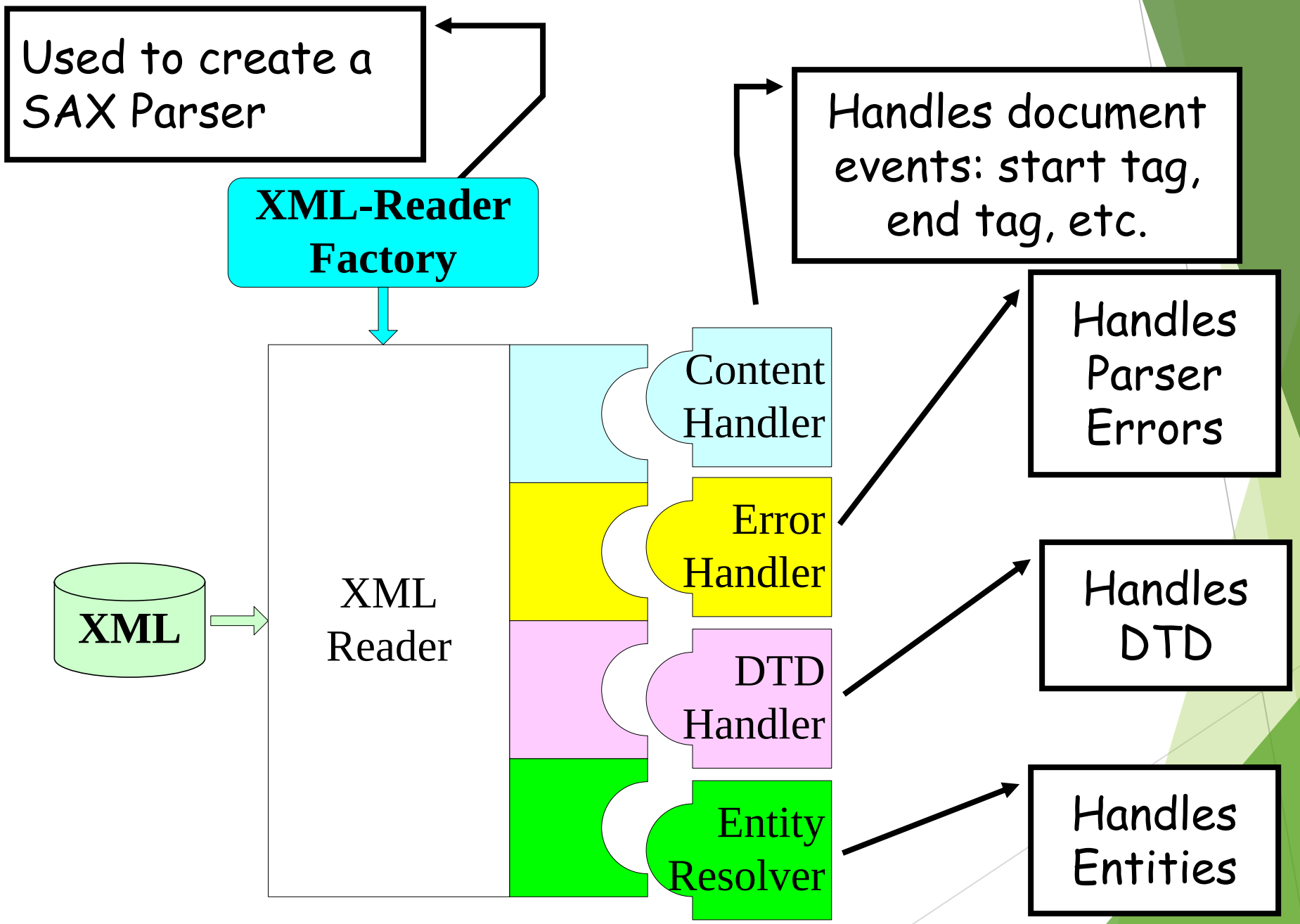
Sample Data

Orders Data in XML:
several orders, each with several items
each item has a part number and a quantity

```
<orders>
<order>
 <onum>1020</onum>
 <takenBy>1000</takenBy>
 <customer>1111</customer>
 <recDate>10-DEC-94</recDate>
 <items>
  <item>
   <pnum>10506</pnum>
   <quantity>1</quantity>
  </item>
  <item>
   <pnum>10507</pnum>
   <quantity>1</quantity>
  </item>
  <item>
   <pnum>10508</pnum>
   <quantity>2</quantity>
  </item>
  <item>
   <pnum>10509</pnum>
   <quantity>3</quantity>
  </item>
 </items>
</order>
...
</orders>
```

startElement

endElement


Used to create a SAX Parser
XML-Reader Factory
Handles document events: start tag, end tag, etc.
Content Handler
Handles Parser Errors
Error Handler
XML
XML Reader
Handles DTD
DTD Handler
Entity Resolver
Handles Entities

SAX API

Two important classes in the SAX API: `SAXParser` and `HandlerBase`.

A new `SAXParser` object is created by:

```
public SAXParser()
```

Register a SAX handler with the parser object to receive notifications of various parser events by:

```
public void setDocumentHandler(DocumentHandler h)
```

A similar method is available to register an error handler:

```
public void setErrorHandler(ErrorHandler h)
```

SAX API - Continued

- The `HandlerBase` class is a default base class for all handlers.
- It implements the default behavior for the various handlers.
- Application writers extend this class by rewriting the following event handler methods:

```
public void startDocument() throws SAXException
public void endDocument() throws SAXException
public void startElement() throws SAXException
public void endElement() throws SAXException
public void characters() throws SAXException
public void warning() throws SAXException
public void error() throws SAXException
```

Creating a SAX Parser

```
import org.xml.sax.*;
import oracle.xml.parser.v2.SAXParser;
public class SampleApp extends HandlerBase {
  // Global variables declared here
  static public void main(String [] args){
    Parser parser = new SAXParser();
    SampleApp app = new SampleApp();
    parser.setDocumentHandler(app);
    parser.setErrorHandler(app);
    try {
      parser.parse(createURL(args[0]).toString());
    } catch (SAXParseException e) {
        System.out.println(e.getMessage());
      }
  }
}
```

SAX API – Sample Application

Write a SAX Parser that reads the `orders.xml` file and extracts the various data items and creates SQL insert statements to insert the data into a relational database table.

```
//Global Variables
Vector itemNum = new Vector();
int numberOfRows, numberOfItems;
String elementEncountered, orderNumber, takenBy,
       customer, receivedDate, partNumber, quantity;

//elementEncountered holds most recent element name

public void startDocument() throws SAXException {
  //Print SQL comment, initialize variable
  System.out.println("--Start of SQL insert Statements");
  itemNum.setSize(1);
  numberOfRows = 0;
}//end startDocument
```

SAX API – Sample Application continued

```
public void startElement(String name,
                         AttributeList atts) throws SAXException {
  elementEncountered = name;
  if (name.equalsIgnoreCase("items")) {
    numberOfItems = 0;
  }//end if statement
}//end startElement

public void characters(char[] cbuf, int start, int len) {
  if (elementEncountered.equals("orderNumber")
    orderNumber = new String(cbuf,start,len);
  else if(elementEncountered.equals("takenBy")) {
    takenBy = new String(cbuf,start,len);
  ...
}//end characters
```

SAX API – Sample Application continued

```
public void endElement(String name) throws SAXException{
  if (name.equalsIgnoreCase("item")) {
    numberOfItems++;
    if (numberOfItems == 1) { // first item; create orders row
      System.out.println(
        "insert into orders values('"+
           orderNumber +"','"+ customer +"','"+
           takenBy +"','"+ receivedDate +"','null');");
      }
      System.out.println("insert into odetails values('"+
          orderNumber +"','"+ partNumber +"','"+
          quantity +"');");
  }//end if statement
  if (name.equalsIgnoreCase("items")) {
    System.out.println("--------");
  }
}//end endElement
```

SAX API – Sample Application continued

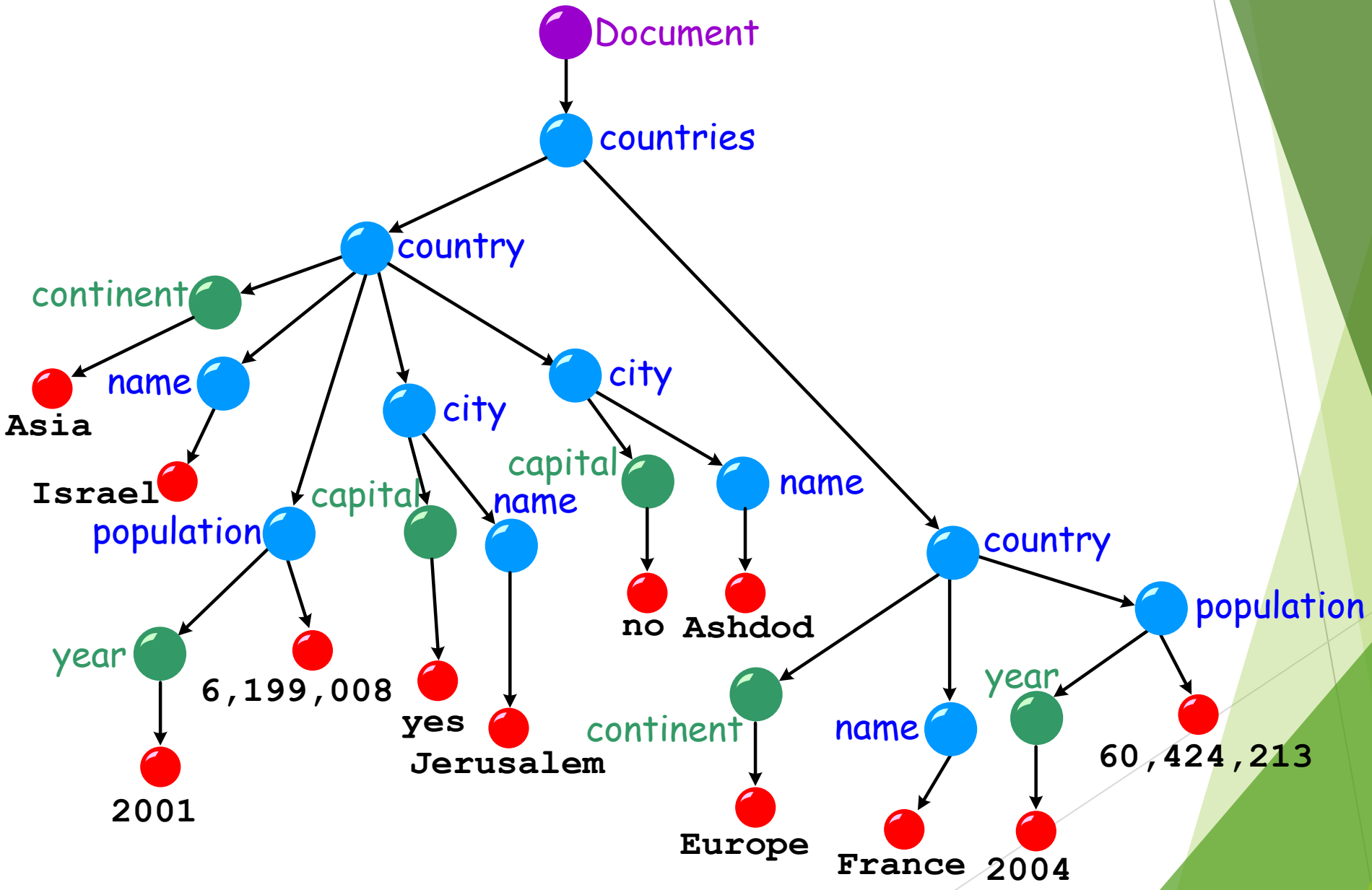
```
public void endDocument(){
  System.out.println("End of SQL insert statements.");
}//end endDocument

public void warning(SAXParseException e)
              throws SAXException {
  System.out.println("Warning:"+ e.getMessage());
}

public void error(SAXParseException e)
              throws SAXException{
  throw new SAXException(e.getMessage());
}
```

DOM Parser

- DOM = Document Object Model
- Parser creates a tree object out of the document
- User accesses data by traversing the tree
  - The tree and its traversal conform to a W3C standard
- The API allows for constructing, accessing and manipulating the structure and content of XML documents

```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name>Israel</name>
    <population year="2001">6,199,008</population>
    <city capital="yes"><name>Jerusalem</name></city>
    <city><name>Ashdod</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2004">60,424,213</population>
  </country>
</countries>
```

The DOM Tree


Document
countries
country
continent
Asia
name
Israel
city
population
year
2001
6,199,008
capital
yes
city
name
Jerusalem
capital
no
city
name
Ashdod
country
continent
Europe
name
France
year
2004
population
60,424,213

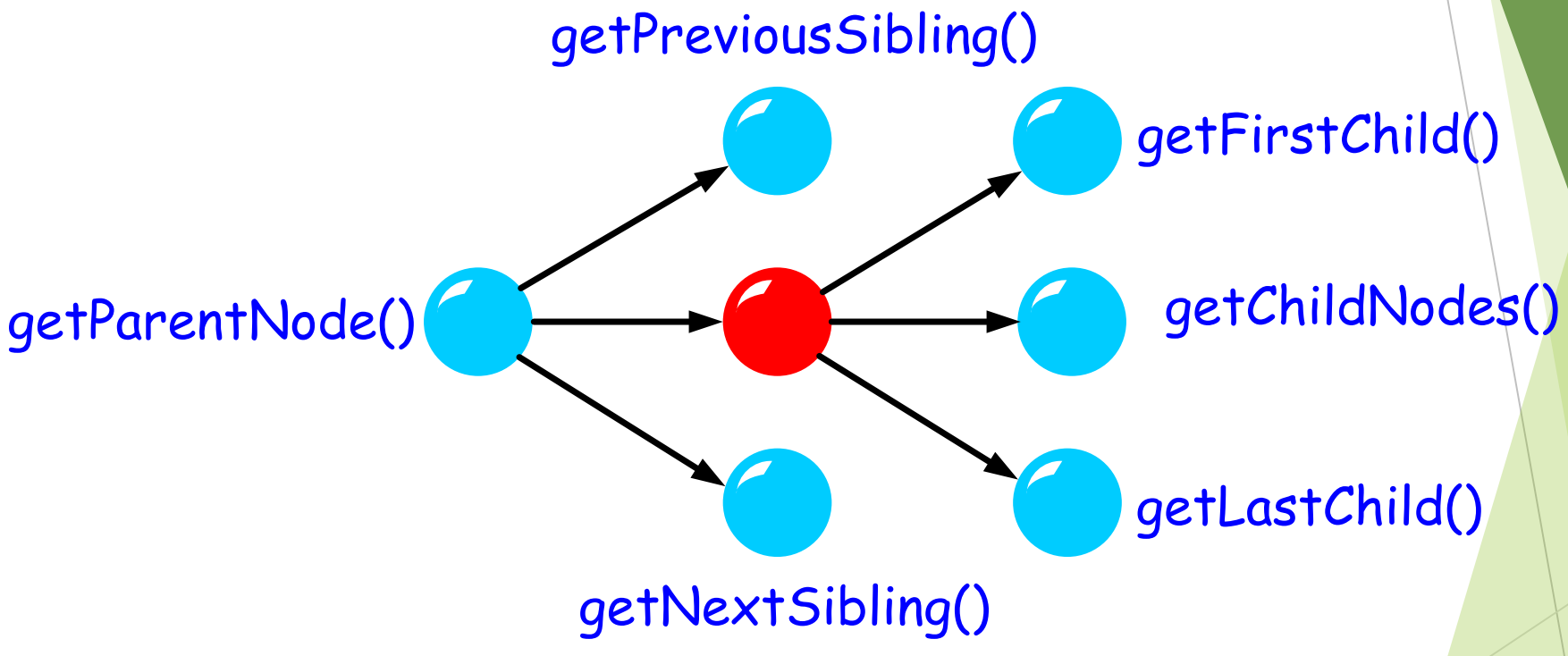
The Node Interface

- The nodes of the DOM tree include
  - a special root (denoted *document*)
  - element nodes
  - text nodes and CDATA sections
  - attributes
  - comments
  - and more ...
- Every node in the DOM tree implements the **Node** interface

Node Navigation

- Every node has a specific location in tree
- **Node** interface specifies methods for tree navigation
  - `Node getFirstChild();`
  - `Node getLastChild();`
  - `Node getNextSibling();`
  - `Node getPreviousSibling();`
  - `Node getParentNode();`
  - `NodeList getChildNodes();`
  - `NamedNodeMap getAttributes()`

Node Navigation (cont)



DOM Parsing Example

Consider the following XML data file describing geographical information about states in U.S.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE geography SYSTEM "geo.dtd">
<geography>
  <state id="georgia">
    <scode>GA</scode>
    <sname>Georgia</sname>
    <capital idref="atlanta"/>
    <citiesin idref="atlanta"/>
    <citiesin idref="columbus"/>
    <citiesin idref="savannah"/>
    <citiesin idref="macon"/>
    <nickname>Peach State</nickname>
    <population>6478216</population>
  </state>
  ...
  <city id="atlanta">
    <ccode>ATL</ccode>
    <cname>Atlanta</cname>
    <stateof idref="georgia"/>
  </city>
  ...
</geography>
```

Geography XML Data DTD

```
<!ELEMENT geography (state|city)*>
<!ELEMENT state (scode, sname, capital,
                 citiesin*, nickname, population)>
   <!ATTLIST state id ID #REQUIRED>
<!ELEMENT scode (#PCDATA)>
<!ELEMENT sname (#PCDATA)>
<!ELEMENT capital EMPTY>
   <!ATTLIST capital idref IDREF #REQUIRED>
<!ELEMENT citiesin EMPTY>
   <!ATTLIST citiesin idref IDREF #REQUIRED>
<!ELEMENT nickname (#PCDATA)>
<!ELEMENT population (#PCDATA)>
<!ELEMENT city (ccode, cname, stateof)>
   <!ATTLIST city id ID #IMPLIED>
<!ELEMENT ccode (#PCDATA)>
<!ELEMENT cname (#PCDATA)>
<!ELEMENT stateof EMPTY>
   <!ATTLIST stateof idref IDREF #REQUIRED>
```

Geography Data: SQL Schema (Oracle Objects)

```
create type city_type as object (
  ccode      varchar2(15),
  cname      varchar2(50)
);

create type cities_in_table as table of city_type;

create table state (
  scode      varchar2(15),
  sname      varchar2(50),
  nickname   varchar2(100),
  population number(30),
  capital    city_type,
  cities_in  cities_in_table,
  primary key (scode))
nested table cities_in store as cities_tab;
```

Create DOM Parser Object

```
import org.w3c.dom.*;
import org.w3c.dom.Node;
import oracle.xml.parser.v2.*;

public class CreateGeoData {
  static public void main(String[] argv)
      throws SQLException {
  // Get an instance of the parser
  DOMParser parser = new DOMParser();

  // Set various parser options: validation on,
  // warnings shown, error stream set to stderr.
  parser.setErrorStream(System.err);
  parser.setValidationMode(true);
  parser.showWarnings(true);

  // Parse the document.
  parser.parse(url);

  // Obtain the document.
  XMLDocument doc = parser.getDocument();
```

Traverse DOM Tree

```
NodeList sl = doc.getElementsByTagName("state");
NodeList cl = doc.getElementsByTagName("city");


XMLNode e = (XMLNode) sl.item(j);
scode = e.valueOf("scode");
sname = e.valueOf("sname");
nickname = e.valueOf("nickname");
population = Long.parseLong(e.valueOf("population"));

XMLNode child = (XMLNode) e.getFirstChild();
while (child != null) {
  if (child.getNodeName().equals("capital"))
     break;
  child = (XMLNode) child.getNextSibling();
}

NamedNodeMap nnm = child.getAttributes();
XMLNode n = (XMLNode) nnm.item(0);
```

Node Manipulation

- Children of a node in a DOM tree can be manipulated - added, edited, deleted, moved, copied, etc.
- To constructs new nodes, use the methods of Document
  - createElement, createAttribute, createTextNode, createCDATASection etc.
- To manipulate a node, use the methods of Node
  - appendChild, insertBefore, removeChild, replaceChild, setNodeValue, cloneNode(boolean deep) etc.

PHP SimpleXML

Example.php

```
<?php
$xmlstr = <<<XML
<?xml version='1.0' standalone='yes'?>
<movies>
 <movie>
  <title>PHP: Behind the Parser</title>
  <characters>
   <character>
    <name>Ms. Coder</name>
    <actor>Onlivia Actora</actor>
   </character>
   <character>
    <name>Mr. Coder</name>
    <actor>El Act&#211;r</actor>
   </character>
  </characters>
  <plot>
   So, this language. It's like, a programming language. Or is it a
   scripting language? All is revealed in this thrilling horror spoof
   of a documentary.
  </plot>
  <great-lines>
   <line>PHP solves all my web problems</line>
  </great-lines>
  <rating type="thumbs">7</rating>
  <rating type="stars">5</rating>
 </movie>
</movies>
XML;
?>
```

PHP SimpleXML

```php
<?php
include 'example.php';

$movies = new SimpleXMLElement($xmlstr);

echo $movies->movie[0]->plot;
?>
```

```
So, this language. It's like, a programming language. Or is it a
scripting language? All is revealed in this thrilling horror spoof
of a documentary.
```

PHP SimpleXML

```php
<?php
include 'example.php';

$movies = new SimpleXMLElement($xmlstr);

echo $movies->movie->{'great-lines'}->line;
?>
```

```
PHP solves all my web problems
```

```php
<?php
include 'example.php';

$movies = new SimpleXMLElement($xmlstr);

/* For each <character> node, we echo a separate <name>. */
foreach ($movies->movie->characters->character as $character) {
   echo $character->name, ' played by ', $character->actor, PHP_EOL;
}

?>
```

```
Ms. Coder played by Onlivia Actora
Mr. Coder played by El ActÓr
```

PHP SimpleXML

```php
<?php
include 'example.php';

$movies = new SimpleXMLElement($xmlstr);

/* Access the <rating> nodes of the first movie.
 * Output the rating scale, too. */
foreach ($movies->movie[0]->rating as $rating) {
    switch((string) $rating['type']) { // Get attributes as element indices
    case 'thumbs':
        echo $rating, ' thumbs up';
        break;
    case 'stars':
        echo $rating, ' stars';
        break;
    }
}
?>
```

```
7 thumbs up5 stars
```

Using XPATH

```php
<?php
include 'example.php';

$movies = new SimpleXMLElement($xmlstr);

foreach ($movies->xpath('//character') as $character) {
    echo $character->name, 'played by ', $character->actor, PHP_EOL;
}
?>
```

```
Ms. Coder played by Onlivia Actora
Mr. Coder played by El ActÓr
```

Setting Values

```php
<?php
include 'example.php';
$movies = new SimpleXMLElement($xmlstr);

$movies->movie[0]->characters->character[0]->name = 'Miss Coder';

echo $movies->asXML();
?>
```

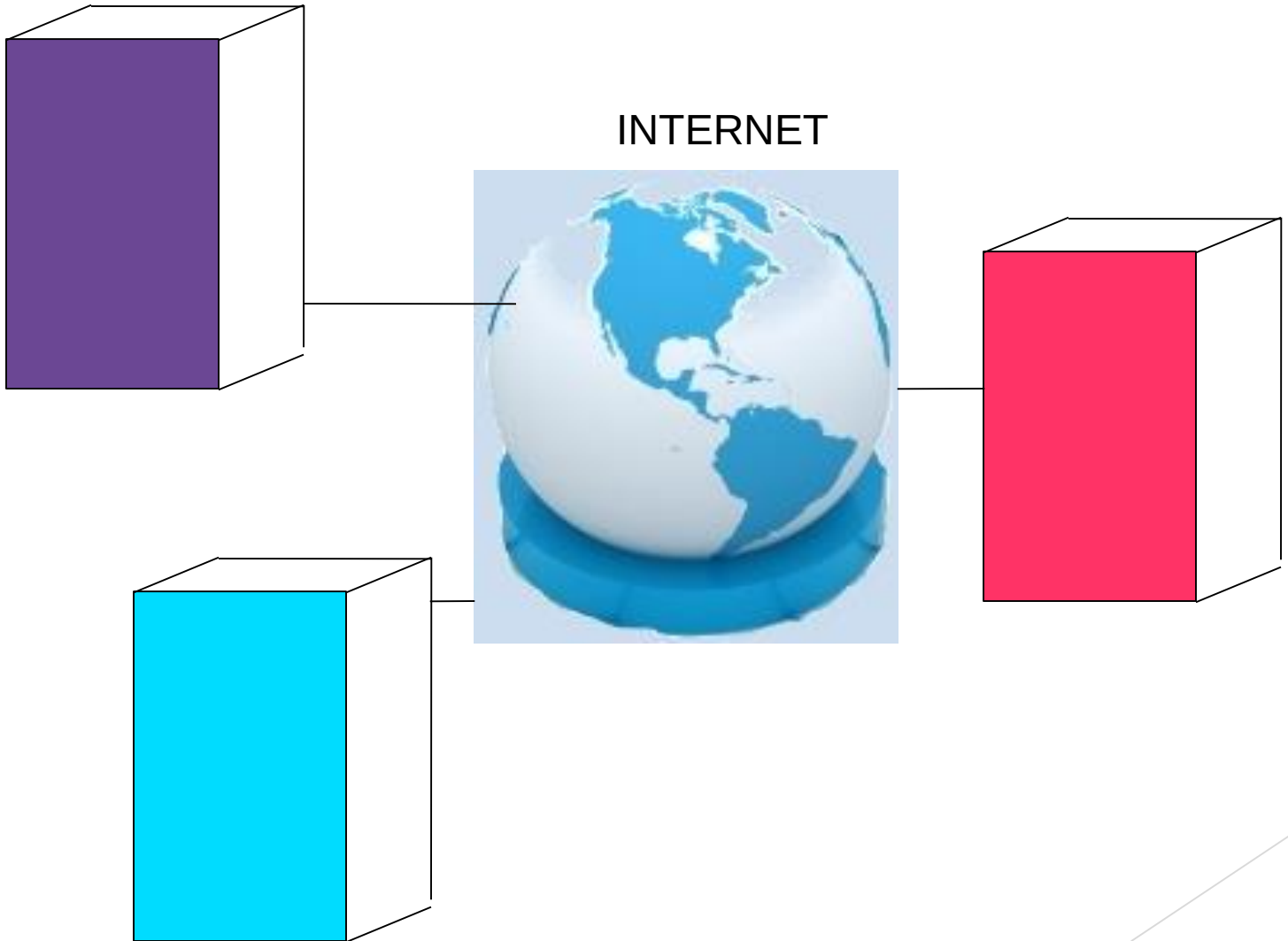
```xml
<?xml version="1.0" standalone="yes"?>
<movies>
 <movie>
  <title>PHP: Behind the Parser</title>
  <characters>
   <character>
    <name>Miss Coder</name>
    <actor>Onlivia Actora</actor>
   </character>
   <character>
    <name>Mr. Coder</name>
    <actor>El Act&#xD3;r</actor>
   </character>
  </characters>
  <plot>
   So, this language. It's like, a programming language. Or is it a
   scripting language? All is revealed in this thrilling horror spoof
   of a documentary.
  </plot>
  <great-lines>
   <line>PHP solves all my web problems</line>
  </great-lines>
  <rating type="thumbs">7</rating>
  <rating type="stars">5</rating>
 </movie>
</movies>
```

Adding Contents

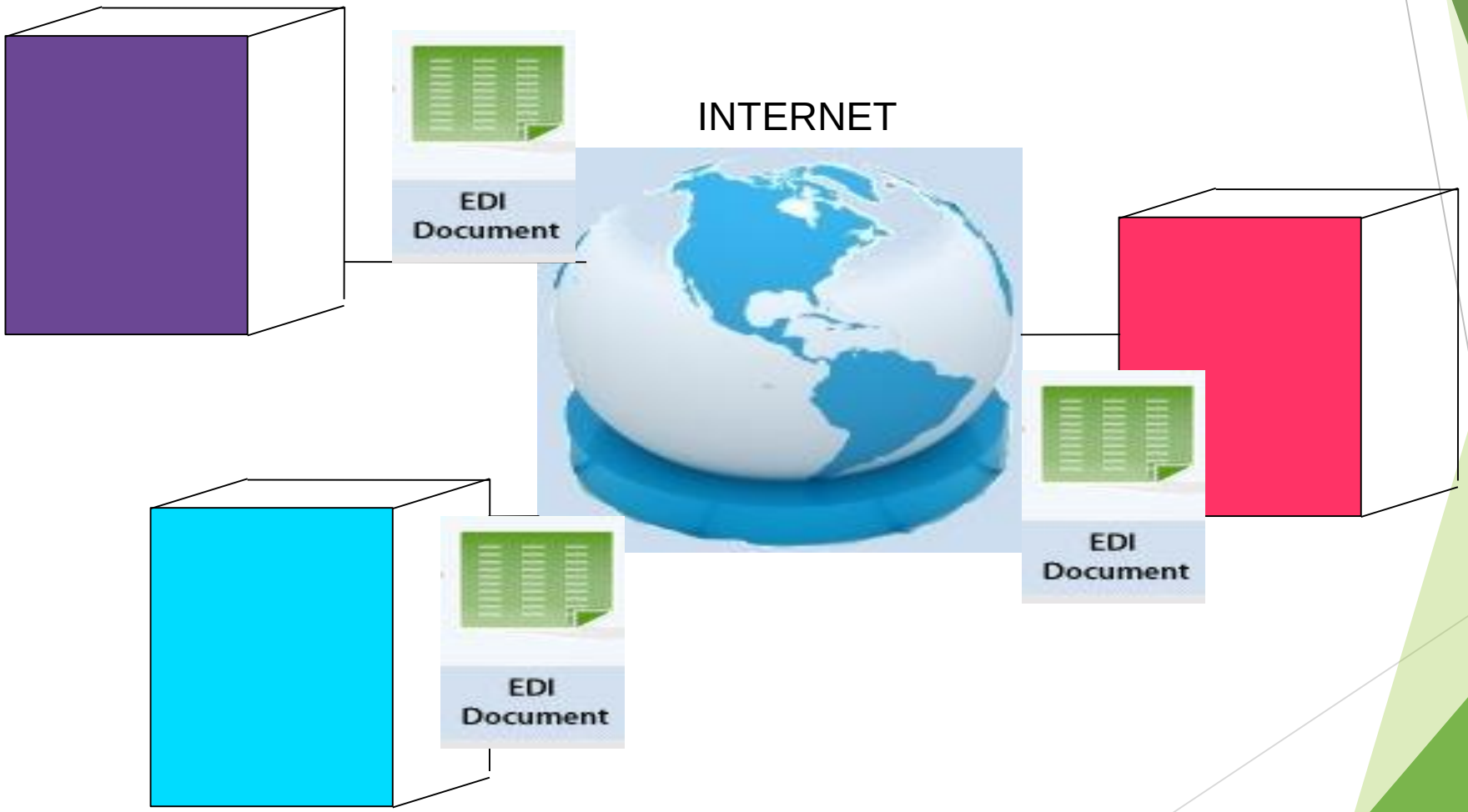
```php
<?php
include 'example.php';
$movies = new SimpleXMLElement($xmlstr);

$character = $movies->movie[0]->characters->addChild('character');
$character->addChild('name', 'Mr. Parser');
$character->addChild('actor', 'John Doe');

$rating = $movies->movie[0]->addChild('rating', 'PG');
$rating->addAttribute('type', 'mpaa');

echo $movies->asXML();
?>
```

```xml
<?xml version="1.0" standalone="yes"?>
<movies>
 <movie>
  <title>PHP: Behind the Parser</title>
  <characters>
   <character>
    <name>Ms. Coder</name>
    <actor>Onlivia Actora</actor>
   </character>
   <character>
    <name>Mr. Coder</name>
    <actor>El Act&#xD3;r</actor>
   </character>
  <character><name>Mr. Parser</name><actor>John Doe</actor></character></characters>
  <plot>
   So, this language. It's like, a programming language. Or is it a
   scripting language? All is revealed in this thrilling horror spoof
   of a documentary.
  </plot>
  <great-lines>
   <line>PHP solves all my web problems</line>
  </great-lines>
  <rating type="thumbs">7</rating>
  <rating type="stars">5</rating>
 <rating type="mpaa">PG</rating></movie>
</movies>
```

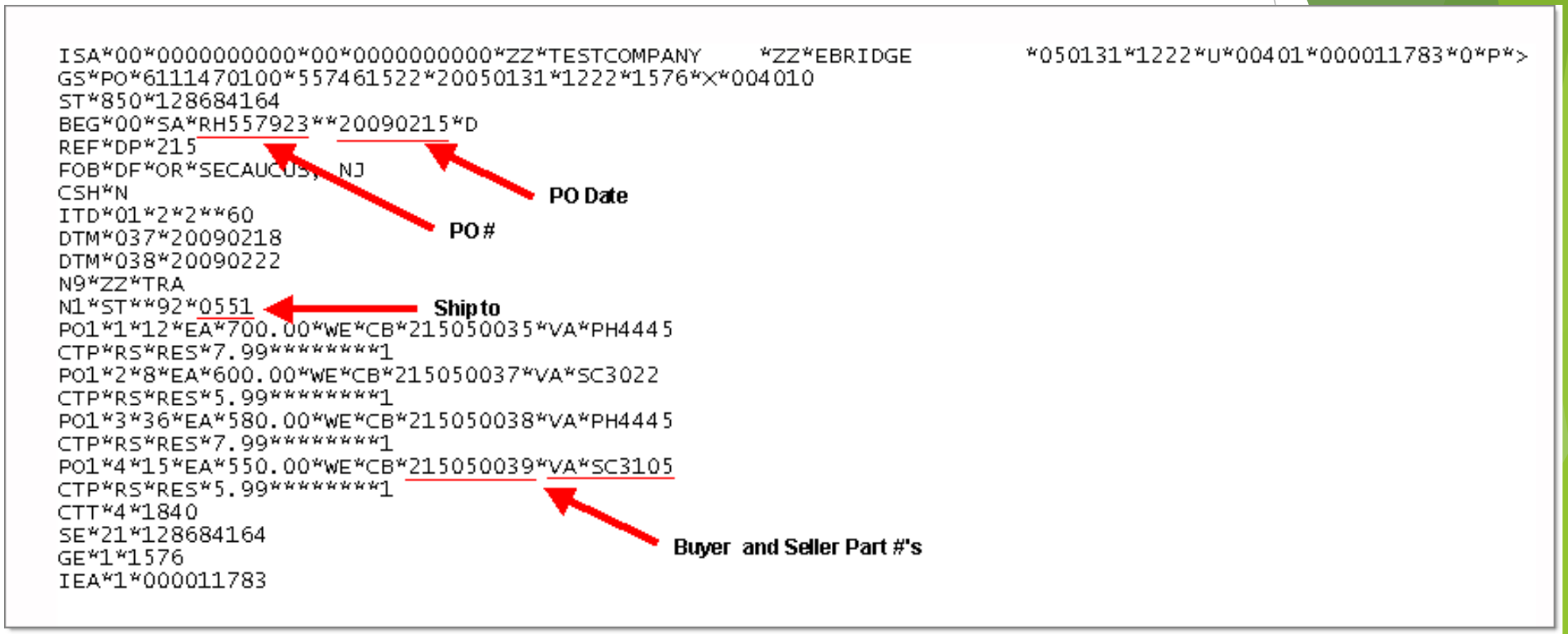
Communicating Machines



Communicationg Machines



Electronic Data Interchage (EDI)



Using EDI

- Much less labor time is required
- Fewer errors occur because computer systems process the documents rather than processing by hand
- Business transactions flow faster.

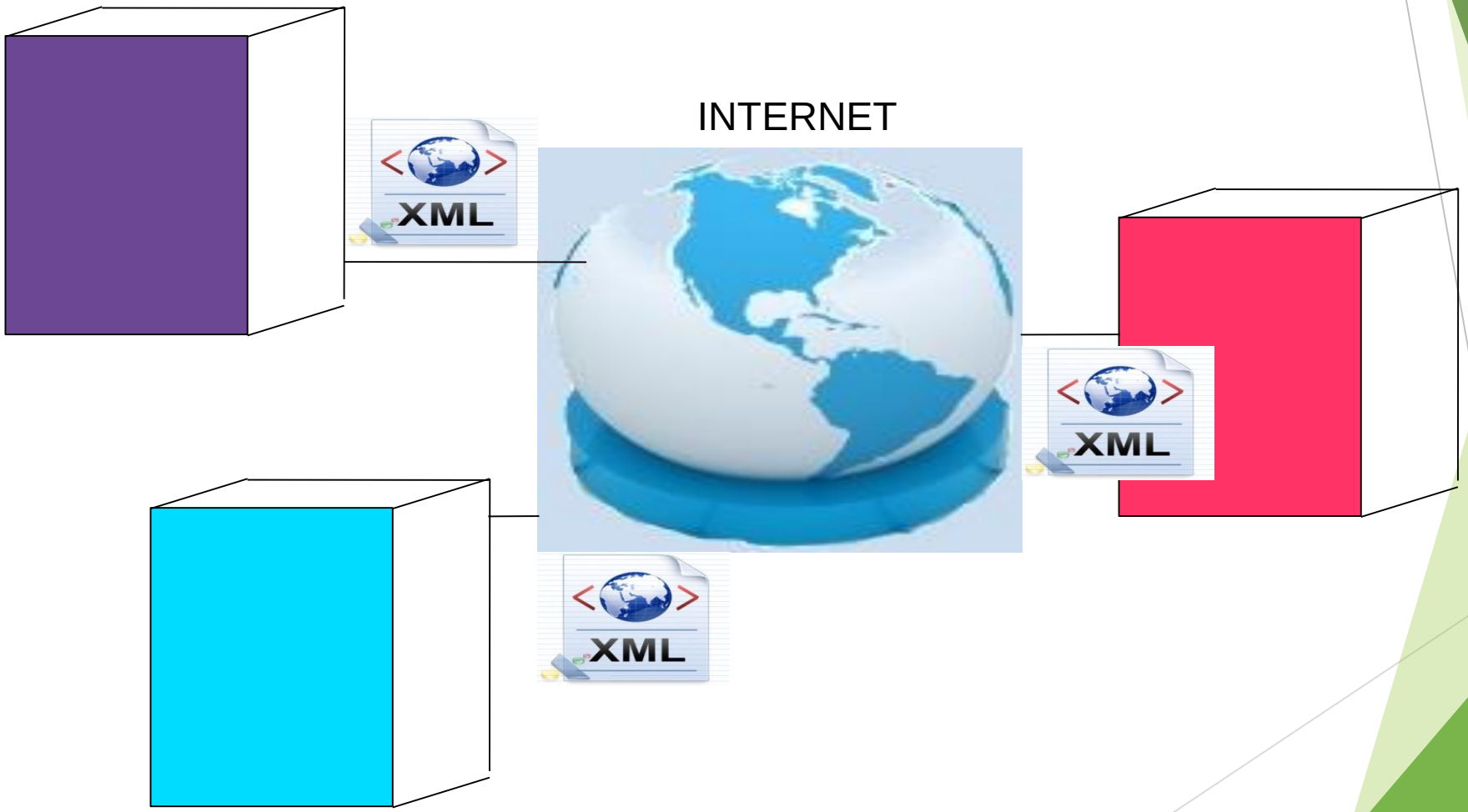
Using EDI

- EDI adapter software is too expensive for most organizations
- The EDI system is highly static and every business process has to be meticulously negotiated between partners
- Since there is no common registry or discovery mechanism, partners have to retain information on institution codes, product codes, up-to-date catalogs etc. associated with everybody they do business with

What we Expect?

- The interchange has to be hardware independent
- The message syntax has to be unambiguous, although not self-describing
- It has to reduce labor intensive tasks of exchanging data such as data reentry
- Should allow the sender to control the exchange,
- including knowing if and when a recipient has received the message

Communicationg Machines



XML

```
<Books>
    <Book ISBN="0553212419">
        <title>Sherlock Holmes: Complete Novels...
        <author>Sir Arthur Conan Doyle</author>
    </Book>
    <Book ISBN="0743273567">
        <title>The Great Gatsby</title>
        <author>F. Scott Fitzgerald</author>
    </Book>
    <Book ISBN="0684826976">
        <title>Undaunted Courage</title>
        <author>Stephen E. Ambrose</author>
    </Book>
    <Book ISBN="0743203178">
        <title>Nothing Like It In the World</title>
        <author>Stephen E. Ambrose</author>
    </Book>
</Books>
```

XML

- XML data format is used to represent information or knowledge
- Standard for exchanging information in the web
- It is a simultaneously human- and machine-readable format.
- It supports Unicode, allowing almost any information in any written human language to be communicated.
- It can represent the most general computer science data structures: records, lists and trees.

XML

- Its self-documenting format describes structure and field names as well as specific values.
- The strict syntax and parsing requirements make the necessary parsing algorithms extremely simple, efficient, and consistent.
- XML is heavily used as a format for document storage and processing, both online and offline.
- It is based on international standards.
- The hierarchical structure is suitable for most (but not all) types of documents.

XML Pros

- It allows validation using schema languages which makes effective unit-testing, firewalls, acceptance testing, contractual specification and software construction easier.
- It manifests as plain text files, which are less restrictive than other proprietary document formats.
- It is platform-independent, thus relatively immune to changes in technology.
- Forward and backward compatibility are relatively easy to maintain despite changes in DTD or Schema.

XML Cons

- XML syntax is redundant or large relative to binary representations of similar data.
- The redundancy may affect application efficiency through higher storage, transmission and processing costs.
- XML syntax is too verbose relative to other alternative ‘text-based’ data transmission formats.
- No intrinsic data type support: XML provides no specific notion of “integer”, “string”, “boolean”, “date”, and so on.

XML Cons

- The hierarchical model for representation is limited in comparison to the relational model or an object oriented graph.
- XML namespaces are problematic to use and namespace support can be difficult to correctly implement in an XML parser.
- XML is commonly depicted as “self-documenting” but this depiction ignores critical ambiguities.

JSON

```
{"books":
   { "book": [
        {
        "isbn" : "0553212419",
        "title" : "Sherlock Holmes: Complete Novels",
        "author" : ”Sir Atrthur Conan Doyle"
        },
       {
        "isbn" : "0743273567",
        "title" : "The Great Gatsby",
        "author" : ”F. Scott Fitzgerald"
        },
       {
        "isbn" : "0684826976",
        "title" : "Undaunted Courage",
        "author" : ”Stephen E. Ambrose"
       },
       {
        "isbn" : "0743203178",
        "title" : "Nothing Like It In the World",
        "author" : ”Stephen E. Ambrose"
```

JSON

- JavaScript Object Notation (Douglas Crockford, http://www.crockford.com)
- JSON data format is ideally suited for consumption by a JavaScript program
- AJAX Application uses Javascript
- Ajax is limited to fetching data from the same domain (website) that the Ajax application came from if the data is formatted as XML
- If the data is formatted as JSON then Ajax can travel across domains

JSON

```
{"person": {
      "name":     "John Doe",
      "website": "http://www.example.com/",
      "email":    "jdoe@example.com"
    }}
```

JSON DATA

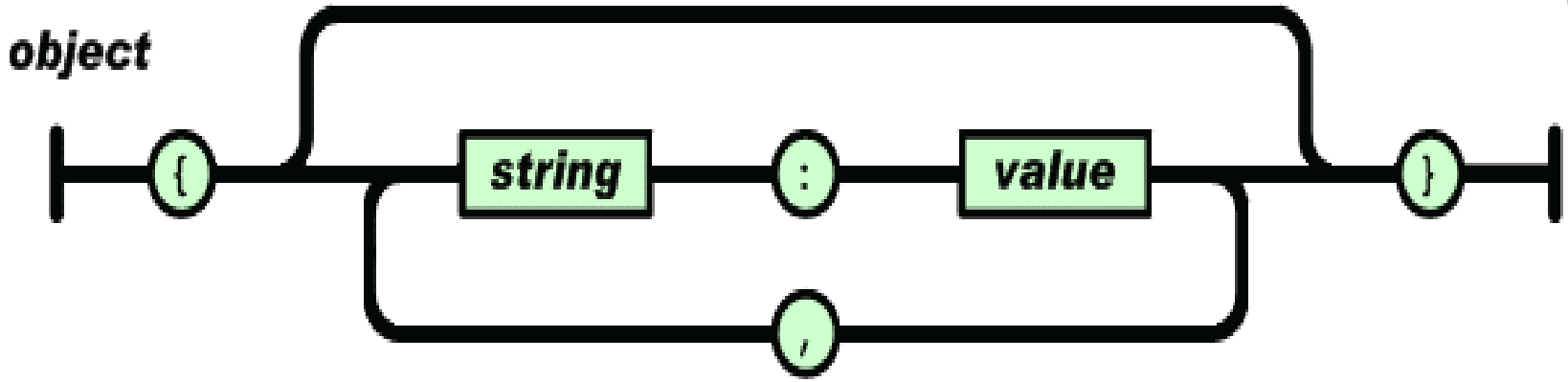
- If variable **dt** holds the JSON data:

dt.person.name returns "John Doe"

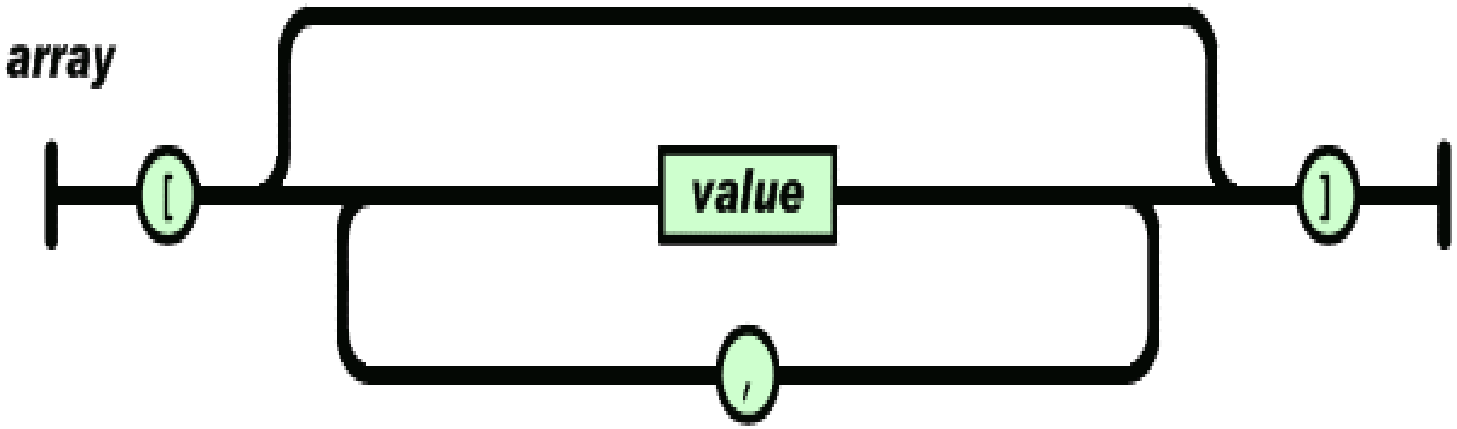
dt.person.website returns "http://www.example.com"

dt.person.email returns "jdoe@example.com"

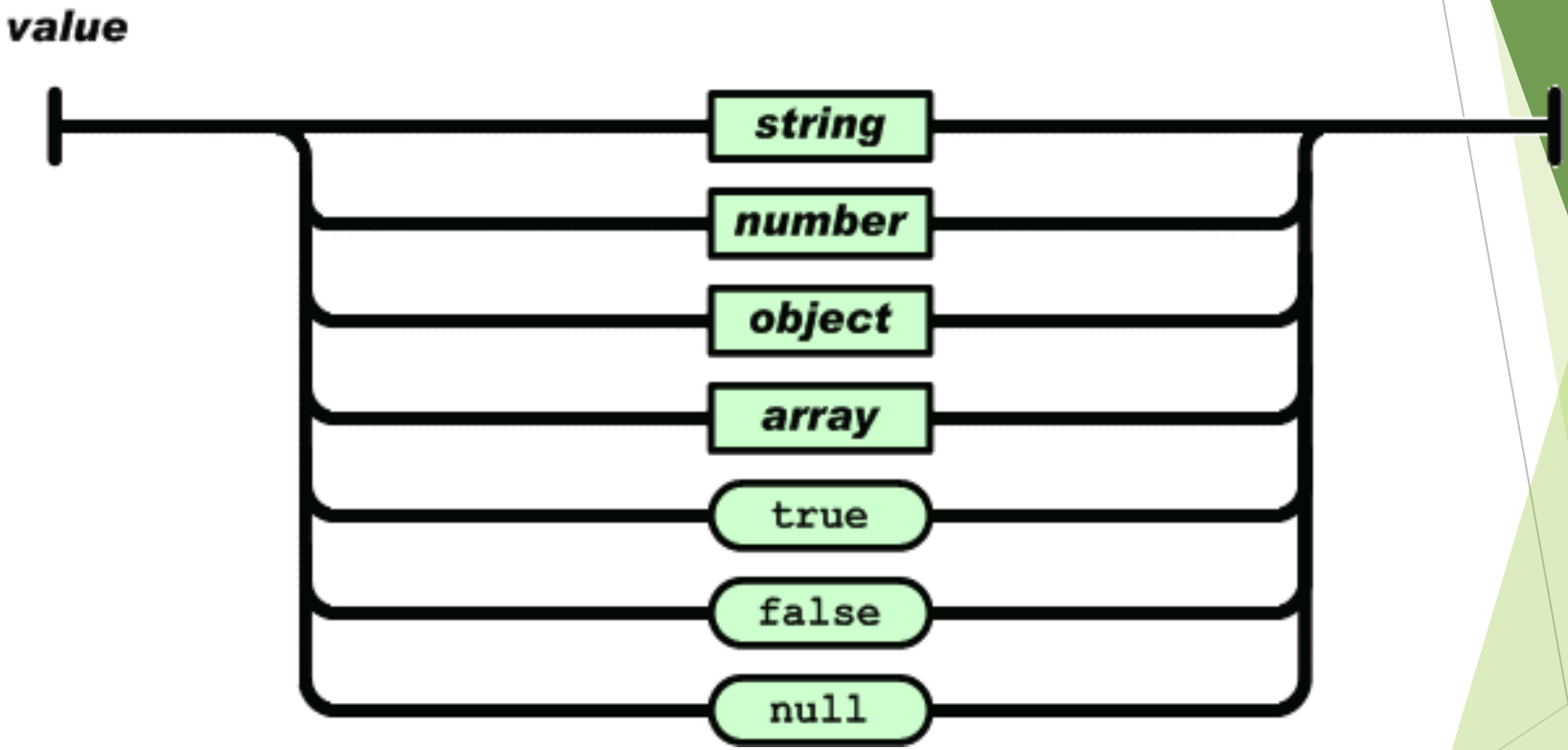
JSON Format



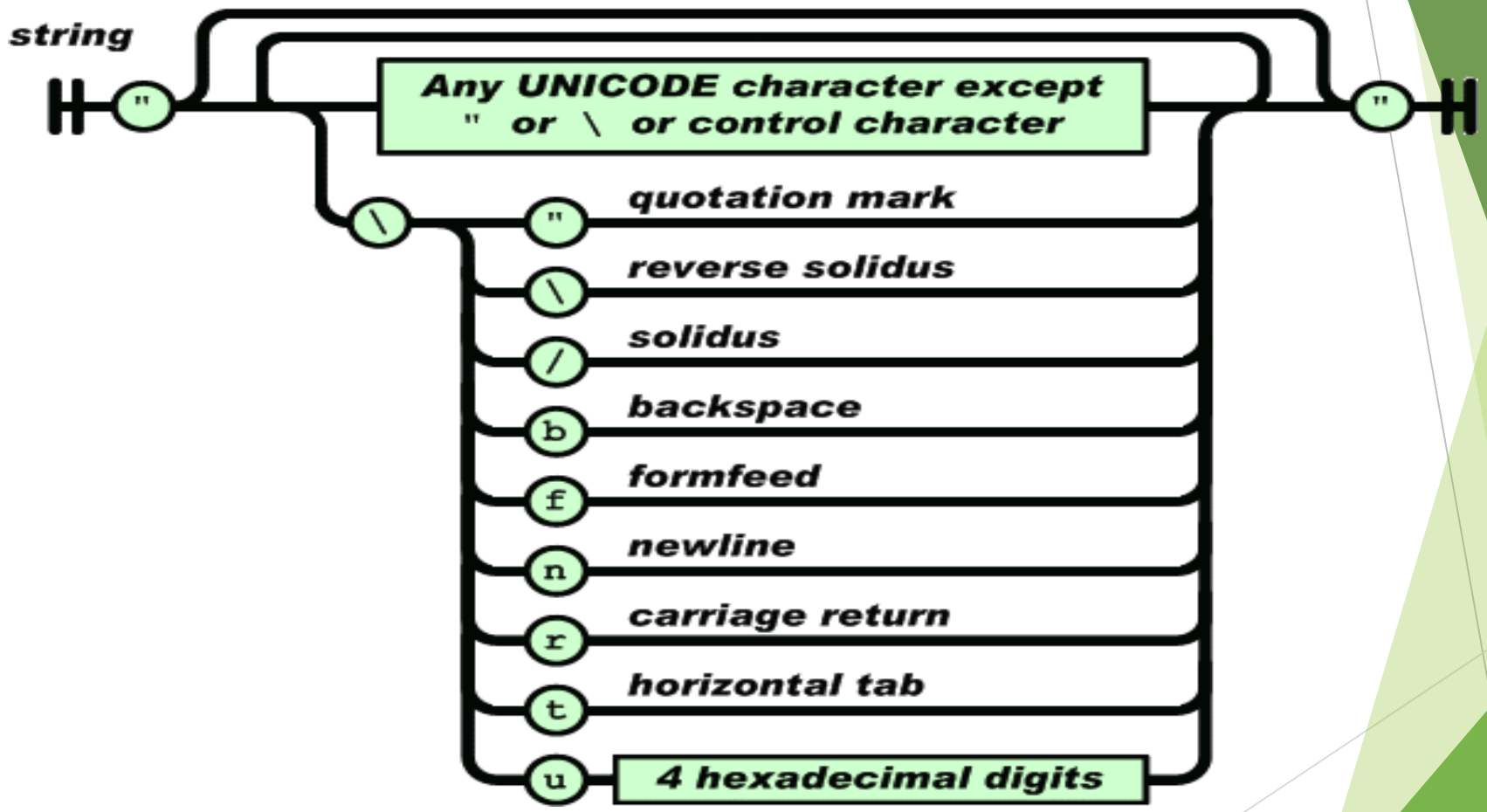
JSON Format



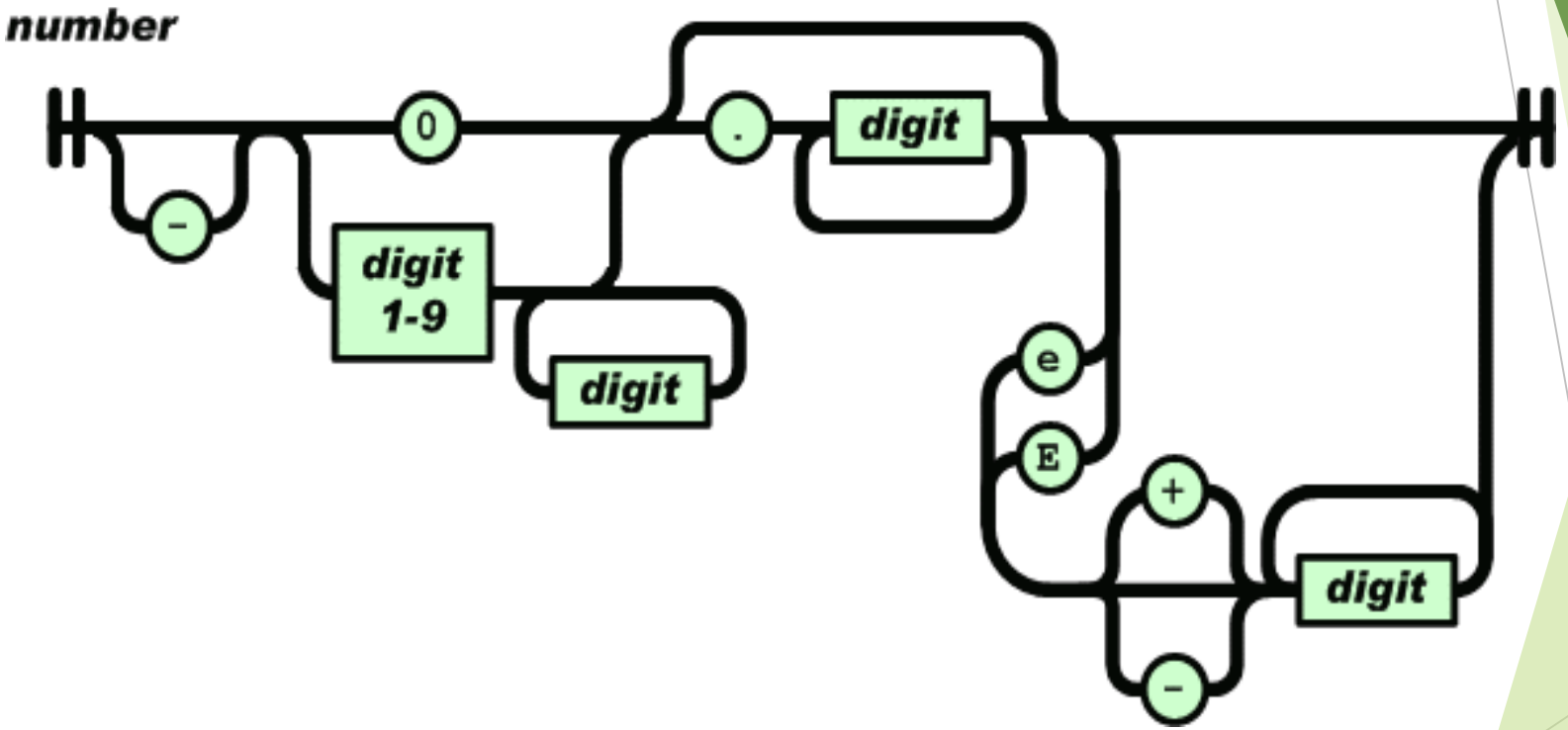
JSON Format



JSON Format


string
"
Any UNICODE character except
" or \ or control character
"
\
"
quotation mark
\
reverse solidus
/
solidus
b
backspace
f
formfeed
n
newline
r
carriage return
t
horizontal tab
u
4 hexadecimal digits

JSON Format



Example

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
      "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
      ]
```

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

JSON in Java Script

```
<html>
<head>
<script type='text/javascript'>
var myFirstJSON = { "person": {
                    "firstName" : "John",
                    "lastName"  : "Doe",
                    "age"       : 23 }};

document.writeln(myFirstJSON.person.firstName);  // Outputs John
document.writeln(myFirstJSON.person.lastName);   // Outputs Doe
document.writeln(myFirstJSON.person.age);        // Outputs 23
</script>
</head>
<body>
</body>
</html>
```

JSON dan AJAX

1. Assignment

```
var JSONFile = "someVar = { 'color' : 'blue' }";
// example of what is received from the server.

eval(JSONFile);
// Execute the javascript code contained in JSONFile.

document.writeln(someVar.color); // Outputs 'blue'
```

JSON dan AJAX

2. JSON via Callback

```
function processData(incommingJSON) {
   document.writeln(incommingJSON.color);
// Outputs 'blue'
}

// example of what is received from the server...
var JSONFile =
    "processData( { 'color' : 'blue' } )";

eval(JSONFile);
```

JSON dan AJAX

3. JSON via Parse

   - run the text against a regular expression which looks for non-JSON characters
   - compile the text into a JavaScript structure
   - (optional) recursively walk the new structure, passing each name/value pair to a filter function for possible transformation

http://telaga.cs.ui.ac.id/~wibowo/lecture/teks/SW/parse.txt

Retrieving JSON Data via AJAX

```
function processData(JSONData) {
   alert(JSONData.color);
}

var ajaxRequest = new ajaxObject('http://www.somedomain.com/getdata.php');
    ajaxRequest.callback = function (responseText) {
       eval(responseText);
    }
    ajaxRequest.update();

// In this example we assume the server sends back the following data file
// (which the ajax routine places in responseText)
//
// processData( { "color" : "green" } )
```

Rembering Java Script

JavaScript ≠ Java
    Developed by Netscape

Purpose: to Create Dynamic websites

Widely Used

How it works in HTML:

<SCRIPT> … </SCRIPT>

<!-- …

// -->

An Example of A Java Script Usage

```
<html>
<head>
<title>JavaScript Page</title>
<script type="text/javascript">
<!-- actual JavaScript follows below

        alert ("Welcome to the Test Site!");

// ending the script -->
</script>
</head>
<body> Content of the Page </body>
</html>
```

Element of Java Script

Variables

Arrays

Functions

Variables

```
<script language="JavaScript">

<!-- definition of variables
var num_car= 25;
var passenger_per_car= 3;

//calculation of total number of people
var total_passenger= num_car * passenger_per_car

Alert(total_passenger);

// end of script -->

</script>
```

Arrays

```
var score = new Array(3);

score[0] = 35
score[1] = 56
score[2] = 10
```

Alternative : var score = new Array(35,56,10);

```
sum=score[0]+score[1]+score[2];

alert(sum) ;
```

Function

```
<script type='text/javascript">
<!-- hide me

function announceTime( )    {

//get the date, the hour, minutes, and seconds

var the_date = new Date();
var the_hour = the_date.getHours();
var the_minute = the_date.getMinutes();
var the_second = the_date.getSeconds();

//put together the string and alert with it

var the_time = the_hour + ":" + the_minute + ":" + the_second;

alert("The time is now: " + the_time); }

// show me -->
</script> </head> <body> ... </body> </html>
```

JavaScript access to the elements of an HTML document.

An object hierarchy

Provides access to elements via:

ID (`ID` attribute of HTML tags)

Order of elements in document

Order of element in collection of similar elements.

```
<P ID=bestparagraph>This is the best paragraph in the
   document. It is the best because it doesn't contain any
   words that have the letter 'e'. If you think it is
   impossible, load me and find out!</P>

<SCRIPT>
b = document.getElementById("bestparagraph");
b.innerHTML="Hi world!";
</SCRIPT>

<P>Told you so</P>
```

innerHTML is an attribute of an object that corresponds to an HTML tag. It's value is the stuff between the start tag and the end tag.

If you assign an **`ID`** attribute to all your HTML tags, you can access the objects that correspond to those elements directly.

The JavaScript **`document`** object supports a method that allows you to get at the object that represents an HTML tag with a specific ID.

```
document.getElementById("foo");
```

You must use unique names!

DOM also supports *collections* of objects,

in Javascript these collections are represented as arrays of objects.

Every HTML element has a **`childNodes`** collection.

The document object has collections **`forms`**, **`images`**, and **`links`** in addition to **`childNodes`**.

images

```
<SCRIPT>
var txt="";
for (var i=0; i<document.images.length;i++) {
  image = document.images[i];
  name = image.getAttribute("src");
  txt = txt + name +"<BR>";
}
document.writeln("Here are the images found:<BR>\n");
document.write(txt);
</SCRIPT>
```

Add this to the bottom of any HTML document

- **`childNodes`**: just immediate descendants (so subelements are not in the collection).

Each member of the collection has it's own **`childNodes`** collection!

You can write a recursive function that can be used to display the structure of a document by looking through the **`childNodes`** collections.

Style properties can be accessed through the DOM.

and can be changed in response to user generated events.

```
document.body.style.backgroundColor="blue";

for (i=0;i<document.childNodes.length;i++) {
document.childNodes[i].style.fontFamily=
     "Courier";
}
```

Java Script and Ajax

AJAX = Asynchronous JavaScript and XML

AJAX is not a new programming language, but a new way to use existing standards.

AJAX is the art of exchanging data with a server, and update parts of a web page - without reloading the whole page

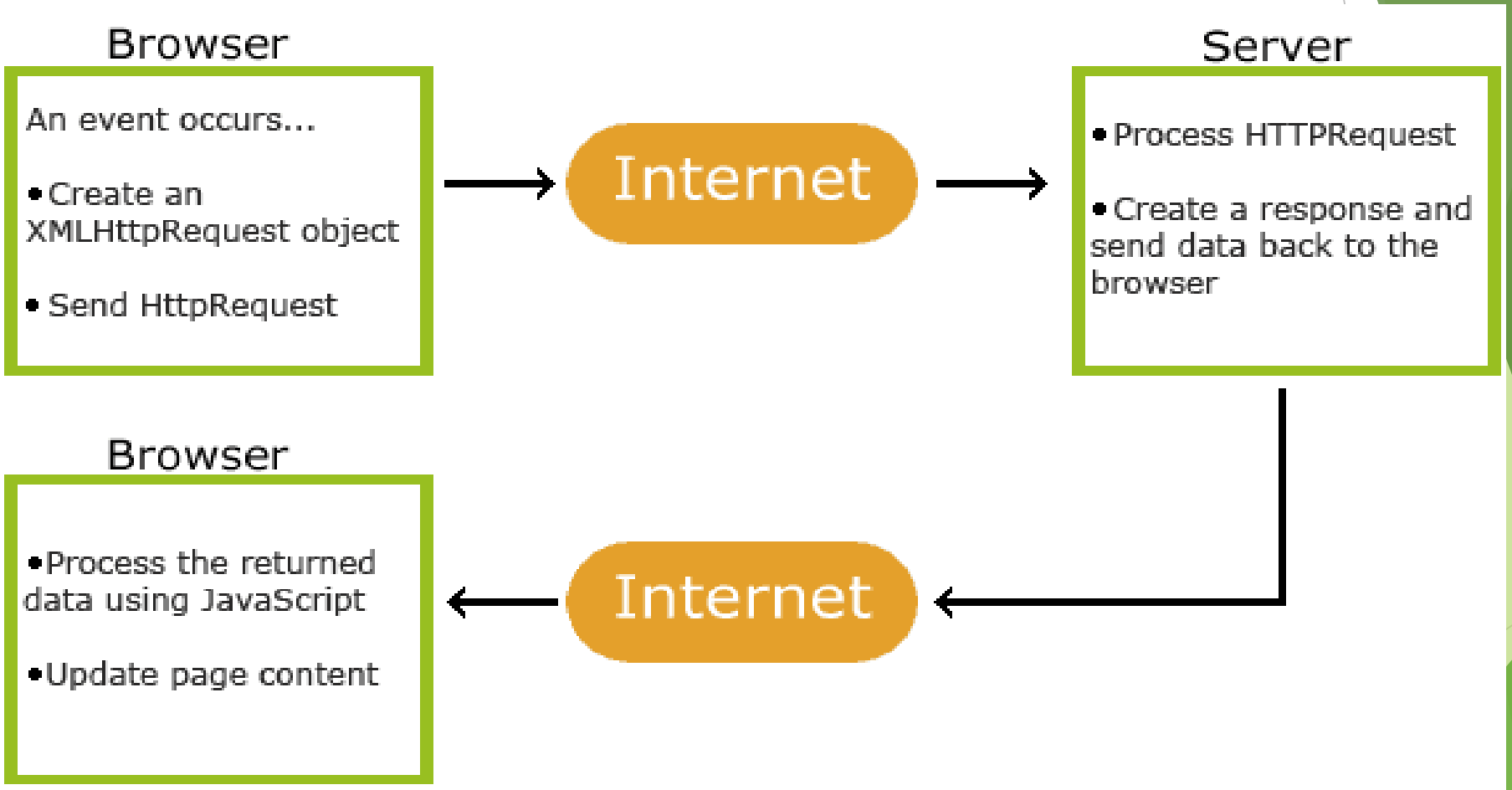
AJAX is a technique for creating fast and dynamic web pages.

Java Script and Ajax

AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Classic web pages, (which do not use AJAX) must reload the entire page if the content should change

How Ajax Works



AJAX is Based on Internet Standards

* XMLHttpRequest object (to exchange data asynchronously with a server)
* JavaScript/DOM (to display/interact with the information)
* CSS (to style the data)
* XML (often used as the format for transferring data)

AJAX applications are browser- and platform-independent