eXtensible Markup Language (XML)

What is XML?

- eXtensible Markup Language
- Subset of SGML (Standardized General Markup Language)
  - marks up content, not formatting, just like SGML
  - omits complex features
- Formatting is done with
  - Cascading Style Sheets (CSS)
  - eXtensible Style Language (XSL)
- Relationship to HTML
  - can express the rules for HTML in an
    XML Document Type Definition (DTD)
  - HTML documents can be XML documents
    - have a single root element (`<html>`)
    - close all tags
    - enclose attribute values in quotes

Some Benefits of Using XML

- XML documents are self-describing
  - not just a collection of data values
- Can use standard XML tools (& some SGML tools) to create, modify, process and view
  - XML editors
  - SAX and DOM programming interfaces
  - XML-aware browsers
- Can use a DTD to constrain structure and allowed values
- Can use XSL to read existing XML documents and create new ones with a different structure
  - when needs change
  - when other applications expect the data to be structured differently
- Errors in data don’t prevent use of other document data

as long as it is still well-formed

Some Design Goals For XML

- Compatible with SGML
  - take advantage of existing expertise and some existing tools
- Optional elements kept to a minimum
  - comparison of spec. sizes: SGML ~500 pages, XML ~30 pages
- Human-readable, like HTML
- Easy to create documents
- Terseness of markup is of little importance
  - element and attribute names take up a lot of space
    but compresses well (~90%)
- Support a variety of applications, not just web browsers
- Easy to write applications that process XML documents
  - can use SAX and DOM interfaces

Custom Markup Languages

- XML is used to create markup languages for specific applications
- Examples
  - – Chemical Markup Language (CML)
    - for displaying molecule descriptions and trees and model diagrams
  - – FDX - for footwear industry data
  - – FpML and FinXML - for securities trading data
  - – HL7 - for health care data
  - – MathML - for mathematical equations
  - – Open Financial eXchange (OFX)
    - for personal financial data like that stored in Quicken or Money
    - also supports consumer and small business banking and bill payment
  - – Open Software Distribution (OSD)
    - for software distribution and updates

Categorizing The Pieces

- Basics: XML
- Validating: DTD and XML Schema
- Links: XLink and XPointer
- Formatting: CSS and XSL
- Programming: SAX and DOM

XML Elements (or tags)

- Must be surrounded by < and > characters
- Must be terminated in one of two ways

```
<city/>
<city>content goes here</city>
```

- Can include attributes

```
<city name="St. Louis" state="Missouri"/>
```

- Can contain child elements

```
<city>
  <name>St. Louis</name>
  <state>Missouri</state>
</city>
```

- Child elements must be properly nested
  - proper: `<a> <b> </b> </a>`
  - improper: `<a> <b> </a> </b>`

Example XML Document

```
<?xml version="1.0"?>
<!DOCTYPE musicCollection SYSTEM "musicCollection.dtd">

<musicCollection>
  <owner>Mark Volkmann</owner>
  <artist type="group">
    <name>Cranberries</name>
    <cd category="pop">
      <title>Everybody Else Is Doing It, So Why Can't We?</title>
      <track>
        <name>I Still Do</name>
        <time>3:16</time>
      </track>
      <track>
        <name>Dreams</name>
        <time>4:32</time>
      </track>
    </cd>
  </artist>
  <!-- place more artists here -->
</musicCollection>
```

XML declaration

document type declaration

prolog section

root element

an XML comment

Well-formed & Valid XML Documents

- Well-formed documents
  - all elements are terminated
  - one element, the root element, contains all the others
  - elements do not overlap
    - child elements must be closed before their parent is closed
  - attribute values are enclosed in quotes
  - < is only used to start elements
  - also a few rules about entities
    - entities are shorthand names for replacement text
    - entities can also refer to non-XML data
- Valid documents
  - well-formed
  - conform to a DTD

Document Type Definitions (DTD)

- A DTD defines
  - allowed elements and their attributes
  - valid element nesting relationships
  - sequence in which elements must appear within their parent element
  - number of times elements can occur within their parent element
  - and more
- Only needed for “valid” documents,
  not for merely “well-formed” documents
- DTDs for many application domains already exist
- Four DTD types
  - `ELEMENT`, `ATTLIST`, `ENTITY` and `NOTATION`

identifies the application to be used for processing data in a given format

Element Definitions

- Four types of element definitions
  - no content - `EMPTY`
  - only text content, no child elements - `(#PCDATA)`
  - only child elements, no text content - requires a content model
  - any content - `ANY`

Parsed Character Data

- Element content models
  - typically a sequence of child elements separated by commas
  - special characters indicate the number of occurrences allowed
    - `?` means 0 or 1, `*` means 0 or more, `+` means 1 or more, default is 1
  - example: **`(a*,b+,c?,(d|e))`**
    - specifies 0 or more `a` elements, followed by one or more `b` elements, followed by 0 or 1 `c` elements, followed by one `d` or `e` element

Attributes

- Ten types of attribute definitions
  - enumerated - list of allowed values; example: `(red|green|blue)`
  - `CDATA` - any characters
  - `NMTOKEN` - only characters allowed in “name tokens”
    - letters, digits, underscore, hyphen, period and colon; begin with a letter
  - `NMTOKENS` - a white space-separated list of `NMTOKEN` values
  - `ID` - unique identifier for the element within the XML document
  - `IDREF` - refers to another element within the same XML document
  - `IDREFS` - a white space-separated list of `IDREF` values
  - `ENTITY` - refers to an unparsed external entity such as an image file
  - `ENTITIES` - a white space-separated list of `ENTITY` values
  - `NOTATION` - identifies the data format of an unparsed external entity
- Attributes can be required (`#REQUIRED`), optional (`#IMPLIED`) or have default values

Example DTD

```
<!ELEMENT musicCollection (owner, artist*)>
<!ELEMENT artist (name, cd*)>
<!ELEMENT cd (title, track*)>
<!ELEMENT track (name, time?)>

<!ELEMENT name (#PCDATA)>
<!ELEMENT owner (#PCDATA)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT title (#PCDATA)>

<!ATTLIST artist type (group|solo) #REQUIRED>

<!ATTLIST cd
 category (classical|country|jazz|pop|rock|other) #REQUIRED
 import (true|false) "false"
 year CDATA #IMPLIED
>

<!ATTLIST track sampleFile ENTITY #IMPLIED>

<!NOTATION mp3 SYSTEM "file:///C:\utils\mp3Player.exe">
<!NOTATION wav SYSTEM "file:///C:\utils\wavPlayer.exe">
```

XML Schema

- W^{3}C proposed schema language alternative to DTDs
  - a superset of the DTD capabilities
  - combines the best features of other similar proposals
    - XML-Data, XDR, DCD, SOX, DDML
  - uses XML syntax instead of a unique syntax like DTDs
    - can edit, validate, and transform using standard XML tools
- Created by the W^{3}C XML Schema Working Group
  - see XML Schema Part 1: Structures W^{3}C Working Draft 6-May-1999 and XML Schema Part 2: Datatypes W^{3}C Working Draft 6-May-1999

XML Schema (Cont'd)

- Some XML Schema improvements over DTDs
  - built-in datatypes
    - `string`, `boolean`, `number`, `dateTime`, `binary` (ex. image data), `uri`
    - `integer`, `decimal`, `real`, `date`, `time`, `timePeriod`
  - user-generated datatypes (built-in datatypes with "constraining facets")
    - `string` supports `length` (for fixed-length), `maxlength` (for variable-length), and COBOL-like pictures or Perl-like regular expressions to constrain format
    - `number` supports `minInclusive`, `minExclusive`, `maxInclusive` and `maxExclusive`
      - `integer`, `decimal` and `real` are all `number`s
  - limits on number of occurrences of child elements
    - `minOccur` can be set to a number (0 or more)
    - `maxOccur` can be set to a number (1 or more) or `"*"` for unlimited

XLink

- Provides greater linking capabilities than HTML
- Any element can serve as a link
  - no special element, just special attributes
- Features
  - simple links
    - unidirectional link to a single resource (like HTML links)
  - extended links
    - bi-directional link to one or more resources
    - can be defined in a separate file to allow multiple files to share the link
  - activate with “`user`” (user click) or “`auto`” (when page is loaded)
  - show with “`replace`” (replace current page),
    “`new`” (display in a new window) or
    “`embed`” (embed into current page)
  - custom behaviors when link is activated (ex. sounds and transition effects)

XLink (Cont’d)

- Extended XLink example

```
<playerDataExtendedLink xlink:form="extended"
    inline="true"
    <playerDataLink xlink:form="locator"
        href="http:www.nhl.com/players/Gretzky/teams.xml"
        title="Teams Played For"/>
    <playerDataLink xlink:form="locator"
        href="http:www.nhl.com/players/Gretzky/goals.xml"
        title="Goal Stats"/>
    <playerDataLink xlink:form="locator"
        href="http:www.nhl.com/players/Gretzky/assists.xml"
        title="Assist Stats”/>
  Gretzky Data
</playerDataExtendedLink>
```

link appears where it occurs in the document

link text

- Applications which support XLink could implement extended links with a popup menu
- Still a working draft (WD-xlink-19980303)

XPointer

- Retrieves a subset of a target document without requiring special tags in that document
  - can search for certain kinds of elements and certain occurrences of elements in the target document
  - can use multiple XPointers to search the result of a previous search
- Must use with XLink to locate the target document
- Example
  - to get win/loss record of the coach of the second team Gretzky played for

```
href="http:www.nhl.com/players/Gretzky.xml
  #child(2, team).child(1, coach).child(1, record)"
```

- Still a working draft (WD-xptr-19980303)

Cascading Style Sheets (CSS)

- Separates formatting from content
  - beneficial because different style sheets can be used to customize content for
    - different media types such as browser, print, and slide presentation
    - different types of users
- Why called “cascading”?
  - formatting rules can come from several places
  - there is a cascading order for choosing between conflicting rules
- Can be used to format both HTML and XML
- Two specs.
  - CSS1 (1996) - well established
  - CSS2 (1998) - most applications don’t fully implement this yet

CSS Rules

- CSS style sheets are composed of rules
- Each rule contains selectors and properties
  - syntax
    - `selectorList { propertyList }`
  - selectorList
    - comma-separated list of tag or element names to be styled (and more)
  - propertyList
    - semicolon-separated list of style properties to be applied
- Properties are applied to all elements that match the corresponding selector
  - and to their descendent elements, if the properties are inheritable and not overridden by a more specific rule

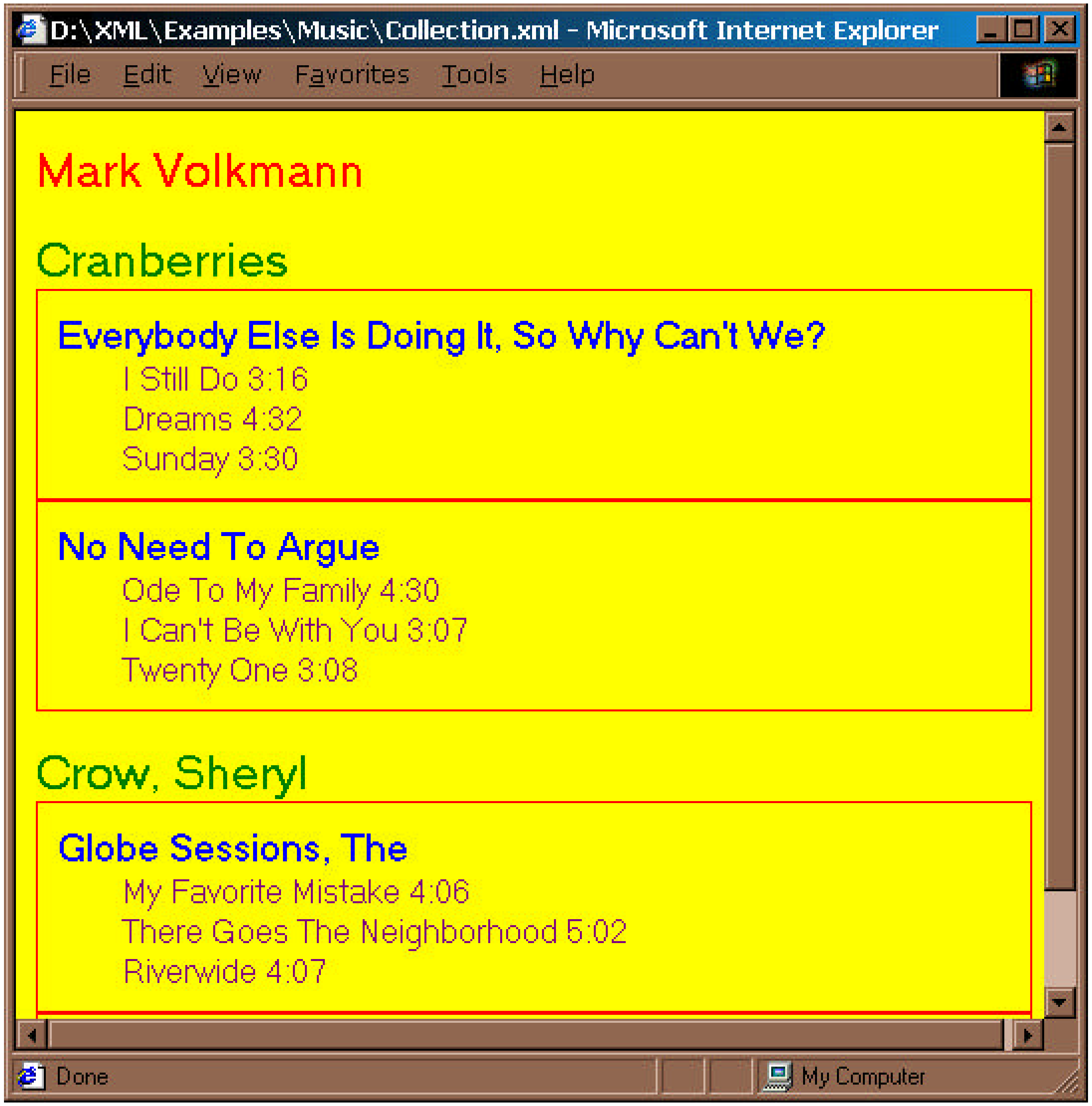
Example CSS Style Sheet

```
* { background:yellow; font-family:sans-serif }
owner { display:block; font-size:large; color:red }
artist { display:block; margin-top:1ex }
artist name { font-size:large; color:green; cursor:hand }
artist gender { display:none }
cd { display:block; border:solid red 1px; padding:10px; width:500px }
cd title { font-size:medium; color:blue }
track { display:block }
track name, track time { font-size:small; color:purple; text-indent:2em }
```

***** is the CSS2 “**universal selector**”. Assigning inheritable properties to it affects the entire document.

An “**em**” is the width of a lowercase ‘m’ in the current font. An “**ex**” is the height of a lowercase ‘x’ in the current font. It’s a good idea to specified sizes in terms of em’s and ex’s so that they are relative to the current font size.

Example Output



D:\XML\Examples\Music\Collection.xml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
My Computer

Mark Volkmann

Cranberries

Everybody Else Is Doing It, So Why Can't We?

I Still Do 3:16
Dreams 4:32
Sunday 3:30

No Need To Argue

Ode To My Family 4:30
I Can't Be With You 3:07
Twenty One 3:08

Crow, Sheryl

Globe Sessions, The

My Favorite Mistake 4:06
There Goes The Neighborhood 5:02
Riverwide 4:07

eXtensible Style Language (XSL)

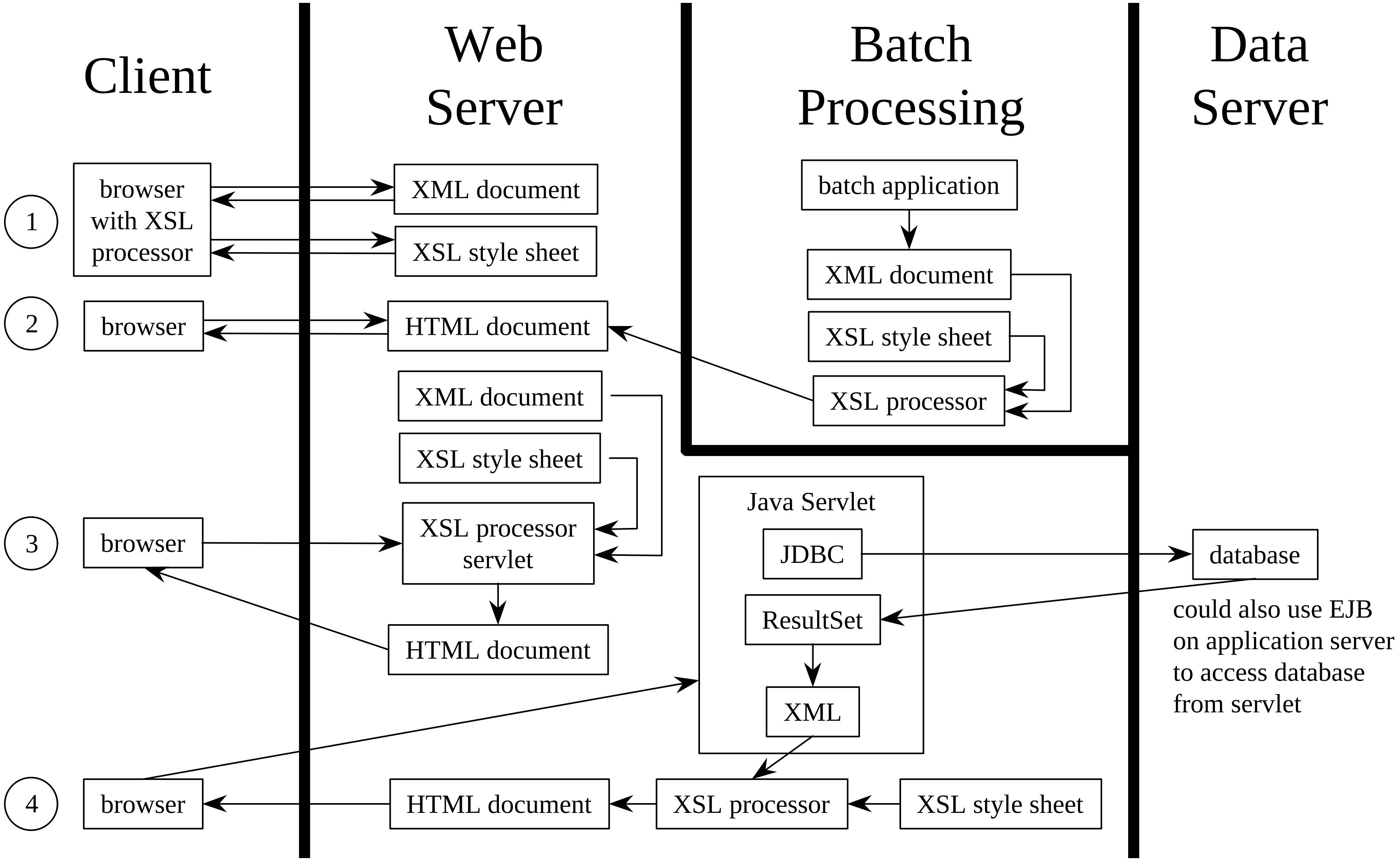
- Transforms and formats an XML document
  - output is text which is commonly new XML (may be valid HTML)
- Two parts
  - tree transformation (REC-**xslt**-19991116)
    - don't have to output all content
    - can output content multiple times (useful for a table of contents)
    - can output content in a different order (specific or sorted order)
    - can add new data and structure (parent/child relationships)
    - can create a new version of a document that conforms to a different DTD
    - XPath (REC-xpath-19991116) defines syntax for selecting nodes
      - a common-syntax for XPointer and XSL
  - formatting (WD-**xsl**-19990421)
    - a working draft
    - creates a result tree whose nodes are “XSL formatting objects”
    - not well supported yet; can output HTML formatted with inline CSS for now

XSL Style Sheet Content

- Uses XML syntax, CSS doesn’t
  - can edit, validate, and transform using standard XML tools
- XSL style sheets are composed of templates
- Each template contains
  - a pattern to be matched
  - the formatting objects to be output
- Two basic approaches
  - template-driven
    - relies primarily on procedural traversal of contents using `<xsl:if>`, `<xsl:for-each>` and `<xsl:choose>` constructs
    - typically results in fewer templates that tend to be larger
  - data-driven
    - relies primarily on pattern matching
    - typically results in more templates that tend to be smaller

Many Ways To Apply XSL

Example Tools
- Microsoft IE5
- XT from James Clark
- IBM XML Enabler servlet



Data-Driven XSL Example

(generates same output as CSS example)

`1999/XSL/Transform` in latest spec.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <xsl:template match="/">
    <html>
      <body style="background:yellow; font-family:sans-serif">
        <xsl:apply-templates select="collection"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="collection">
    <div style="font-size:large; color:red">
      <xsl:value-of select="owner"/>
    </div>
    <xsl:apply-templates select="artist"/>
  </xsl:template>
```

`"/"` matches document root;
must have one template like this

Data-Driven XSL Example (Cont’d)

```
<xsl:template match="artist">
  <div style="margin-top:1em">
    <div style="font-size:large; color:green; cursor:hand">
      <xsl:value-of select="name"/>
    </div>
  </div>
  <xsl:apply-templates select="cd"/>
</xsl:template>

<xsl:template match="cd">
  <div style="border:solid red 1px; padding:10px; width:500px">
    <div style="font-size:medium; color:blue">
      <xsl:value-of select="title"/>
    </div>
    <xsl:apply-templates select="track"/>
  </div>
</xsl:template>
```

Data-Driven XSL Example (Cont’d)

```
  <xsl:template match="track">
    <div style="font-size:small; color:purple; text-indent:2em">
      <xsl:value-of select="name"/>
      <xsl:value-of select="time"/>
    </div>
  </xsl:template>

</xsl:stylesheet>
```

Simple API for XML (SAX)

- Event-driven method of processing XML documents
  - doesn’t create a data structure representing the parsed document
  - generates events during parsing for which applications can listen
- An interface or API, not an implementation
  - many implementations exist
  - any implementation can be used without making code changes
- Developed by
  - David Megginson and members of the xml-dev mailing list
- Not a W^{3}C standard
  - but supported by most XML processors

When Is It Useful?

- When data can be processed in the order in which it appears
  - if a piece of data can’t be processed until some subsequent data is parsed then it must be stored in memory
    - puts the burden of creating a data structure on the developer
- When the entire document does not have to be parsed to begin processing
  - more efficient
- When the document being parsed is large and only some of the data is needed
  - no sense storing data that will never be used in a data structure
- When a new document will not be created from the parsed document

SAX Events

- Setting of Locator
  - used to get current line and column number for each subsequent event
- Start and end of document
- Start and end of each element
- End of each processing instruction
- End of each run of character data
  - what constitutes a "run" of characters is implementation dependent
    - could break on any whitespace or be the entire node value
    - all the characters will be from the same node
- End of all ignorable whitespace
  - multiple consecutive whitespace characters that may not be preserved
  - whitespace not in a CDATA section (similar to HTML `<pre>` tag)

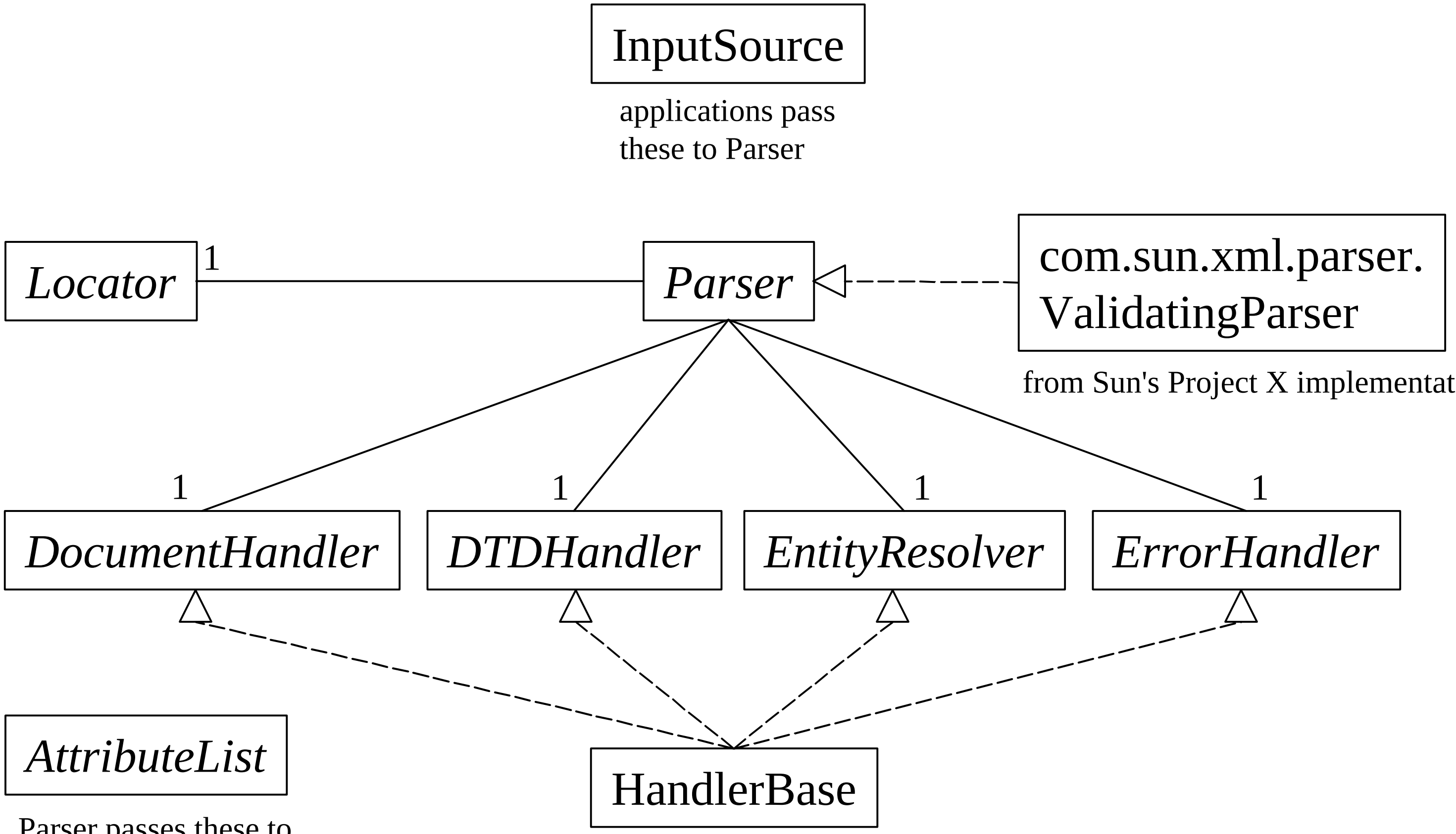
for applications that process the XML data

Listening For SAX Events

- Create a “handler” class that implements DocumentHandler or extends HandlerBase
  - this listens for SAX events
  - HandlerBase is a class that implements DocumentHandler with empty methods
- Setup steps
  - create a SAX Parser object to parse XML documents
  - create a “handler” object to receive the events from the SAX parser
  - pass the “handler” object to the `setDocumentHandler` method of the SAX Parser
  - tell the parser to parse a specific XML document
- Appropriate methods in the DocumentHandler object will be invoked by the Parser as it parses an XML document

org.xml.sax Package

(interfaces are in *italics*)



InputSource

applications pass
these to Parser

Locator 1

Parser

com.sun.xml.parser.
ValidatingParser

from Sun's Project X implementation

1 *DocumentHandler*

1 *DTDHandler*

1 *EntityResolver*

1 *ErrorHandler*

AttributeList

Parser passes these to
DocumentHandler
when the start tag of
an element is parsed

HandlerBase

applications typically
extend this

SAX Example

This prints the name of all artists in an XML music collection.

```
import com.sun.xml.parser.*;
import java.io.*;
import org.xml.sax.*;

public class SAXExample extends HandlerBase {
    private String currentElementName;
    private String previousElementName;

    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("Usage: java SAXExample filename");
            System.exit(1);
        }

        Parser parser = new ValidatingParser(true); // true means to validate
        parser.setDocumentHandler(new SAXExample()); // this class
        InputSource source = Resolver.createInputSource(new File(args[0]));
        parser.parse(source);
    }
}
```

SAX Example (Cont’d)

```
    public void startElement(String name, AttributeList atts) {
        previousElementName = currentElementName;
        currentElementName = name;
    }

    public void characters(char[] ch, int start, int length) {
        // Print the content of every "name" element
        // this is a child of an "artist" element.
        if ("artist".equals(previousElementName) &&
            "name".equals(currentElementName)) {
            String content = String.valueOf(ch, start, length);
            System.out.println(content);
        }
    }

    public void endDocument() {
        // Processing which cannot begin until parsing has completed
        // should be initiated here.
    }
}
```

Document Object Model (DOM)

- Data structure-driven method of processing XML documents
  - more complex and more capable than SAX
    - can add, modify, and delete content
    - can create new documents
  - creates a data structure representing the parsed document
  - doesn't generate events during parsing
- An interface or API, not an implementation
  - many implementations exist
  - any implementation can be used without making code changes
- Developed by the W^{3}C
- A W^{3}C standard
  - supported by most XML processors

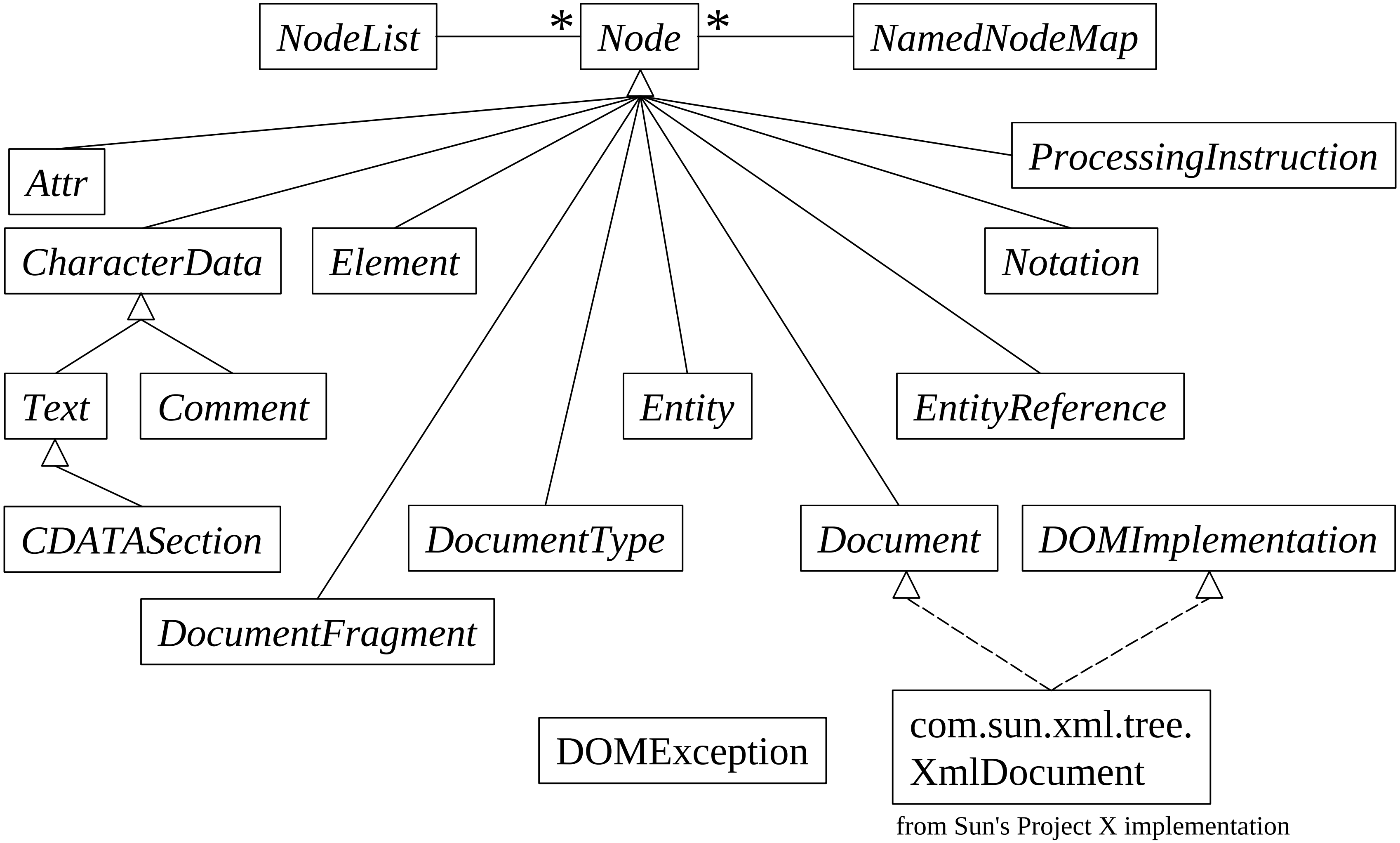
IBM’s xml4j parser can generate SAX events AND create a data structure

Document Object Model (Cont'd)

- Creates a tree data structure representing the parsed XML document that can be traversed in any order, any number of times
  - SAX only allows XML data to be processed as it is parsed, in the order it is parsed (unless you create your own data structure)

org.w3c.dom Package

(interfaces are in *italics*)



from Sun's Project X implementation

DOM Example

This creates an XML music collection document.

```
import com.sun.xml.tree.XmlDocument;
import java.io.FileWriter;
import java.io.IOException;
import org.w3c.dom.*;

public class DOMCreate {

    public static void main(String[] args) {
        XmlDocument doc = new XmlDocument();
        doc.setDoctype(null, "musicCollection.dtd", null);

        Element musicCollectionElement =
            doc.createElement("musicCollection");
        doc.appendChild(musicCollectionElement);

        Element ownerElement = doc.createElement("owner");
        ownerElement.appendChild(doc.createTextNode("Mark Volkmann"));
        musicCollectionElement.appendChild(ownerElement);
```

SYSTEM id

causes a document type declaration to be generated

no PUBLIC id

no internal subset

root element

adding text as the content of an element requires creation of a text node

- conformance to the DTD is not checked while tree nodes are added, modified, and deleted
- it is only checked when an existing XML document text file is parsed

DOM Example (Cont’d)

```
        Element artistElement = doc.createElement("artist");
        artistElement.setAttribute("type", "solo");
        artistElement.setAttribute("vocals", "female");
        Element nameElement = doc.createElement("name");
        nameElement.appendChild
            (doc.createTextNode("Sarah McLachlan"));
        artistElement.appendChild(nameElement);
        musicCollectionElement.appendChild(artistElement);

        // Output the XML document.
        try {
            FileWriter fw = new FileWriter
                ("NewCollection.xml");
            doc.write(fw, "UTF-8");
            fw.close();
        } catch (IOException ioe) {
            System.err.println(ioe);
        }
    }
}
```

<u>The Output</u>

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

<!DOCTYPE musicCollection
SYSTEM 'musicCollection.dtd'>

<musicCollection>
  <owner>Mark Volkmann</owner>
  <artist type="solo" vocals="female">
    <name>Sarah McLachlan</name>
  </artist>
</musicCollection>
```