Introduction to XML

Large Scale Programming, 1DL410, autumn 2009
Cons T Åhs
Input files, i.e., scene descriptions to our ray tracer are written in XML.

What is XML?

- XML - extensible markup language (W3C recommendation 1998)
- A format for external representation of structured data.
- Portable.
- Human readable (at least sort of)
- Machine readable.
- Standard!
- Very much used everywhere.
- Lots of tools available.
A reasonable view is that an XML document is an external and transportable representation of a data structure/class/instance/method call/etc.

Compare to the serialisation of an object.

Only in very rare cases would one try to edit a serialised object and/or generate one by hand.

The big difference is that an XML document is, and is meant to be, humanly readable.

This is a mixed blessing, since one might think it is easy to write and read XML "manually", i.e., either with your own tools and/or by hand.

The XML format is very general which means that tools that handle XML are non trivial.

Avoid "short cuts", such as doing things by hand, writing your own "simpler" tool etc.

Learn the general tools and use them!

XML can not be "parsed" using regular expressions.

It is ok to write XML by hand or with your own tool, but less ok to read - why?
Anatomy of an XML document

<?xml version="1.0" encoding="ISO-8859-1"?>
<Order xmlns="http://ns.cafeconleche.org/Orders/"

  <Customer id="c32">Chez Fred</Customer>

  <Product>
    <Name>Birdsong Clock</Name>
    <SKU>244</SKU>
    <Quantity>12</Quantity>
    <Price currency="USD">21.95</Price>
    <ShipTo>
      <Street>135 Airline Highway</Street>
      <City>Narragansett</City>
      <State>RI</State>
      <Zip>02882</Zip>
    </ShipTo>
  </Product>

  <Subtotal currency='USD'>263.405</Subtotal>
  <Tax rate="7.0" currency='USD'>18.44</Tax>
  <Shipping method="USPS" currency='USD'>8.95</Shipping>

  <!-- It compiles, ship it! -->
  <Total currency='USD'>290.79</Total>
</Order>
XML Anatomy

- An XML document describes a tree structure.
- As such it has a root element.
- An XML document has only ONE root element, ie, it contains exactly one tree structure.
- A tree structure is recursive and so is an XML document. Elements can contain new elements etc.
- Don’t assume that there is a fixed depth on the element structure.
Well formed and valid XML

- We are only concerned with XML documents that are well formed and valid.
- “Well formed” states that it follows the syntax rules of XML.
  - Being well formed is a rather weak requirement since the syntax rules of XML are quite simple.
- “Valid” states that the contents of the XML document follow a specification in a DTD or a schema.
- Compare to syntax and semantics of a ordinary program. It is possible to write a program that has a correct syntax, but lacks semantics.
- A program should NOT try to do anything “sensible” with XML documents that are not well formed or that are invalid.
  - We wouldn’t try to run a program that was syntactically incorrect.
  - Neither would we try to run a program that couldn’t be assigned meaningful semantics.
Validity

- The syntax of XML tells us how a well formed XML document should look.
- For a specific XML document, we can also specify the “semantics” of the document, e.g.,
  - the types of the element values, string, number etc
  - constraints, e.g., a number > 0 or a zip code in a certain format
- This describes what needs to be satisfied for the XML document to have a “meaning”.
- This can be done using a schema language.
- DTD - document type definition, standard but limited.
- Handling of DTDs is built into most XML parsers.
- W3C XML Schema Language - overcomes limitations of DTDs
- Several schema languages exist, with different properties.
External vs. Internal

- A file housing an XML document can be seen as the external representation.
- To get an internal representation, we need to read, parse, an XML document.
- The parser needs to (re)present the XML document to the program. This is done using an API for XML.
- SAX is a read only API generating events during the parsing of the XML document.
  - The whole XML document will not reside in memory.
  - An internal representation can be constructed from the events.
- DOM is a read/write API that represents the XML document as a tree in memory.
  - The whole XML document will reside in memory, directly corresponding to the XML tree.
  - Depending on what one wants to do, an alternative internal representation might be constructed. This is probably beneficial in terms of efficiency.
Additional XML Tools

- XSL/XSLT - tools for transforming XML documents.
- XSLT is a complete functional programming language!
- XPath - find and address parts of an XML document.
Basic validity with DTDs

- A DTD (Document Type Definition) can be used to define the structure of a XML document.
- For each element you can specify attributes, simple attribute types and sub elements.
- The specifications express syntactic restrictions.
- It has limited expressiveness in that it can not describe the actual type (string, number, date etc) of element values.
- Additional validation must be done during parsing.
- The DTD consists of a number of declarations
  - element declarations specifying valid elements and their contents.
  - attribute declarations specifying valid attributes and their type for each element.
Element declarations

- An element declaration states the name of the element and the contents.
  - `<!ELEMENT name contents>`

- The contents can be
  - EMPTY - this element has no contents
    - `<!ELEMENT leaf EMPTY>`
  - ANY - this element contains essentially anything and we don’t specify it further
    - `<!ELEMENT data ANY>`
  - PCDATA (parsed character data) - this element has contents which is parsed, i.e., entities are translated.
    - `<!ELEMENT price (#PCDATA)>`

- Note that this is as “specific” as we can get when specifying the contents of a leaf.
- We can’t say that it must be a number etc.
Element declarations

- children, ie, a non empty sequence of subelements (where #PCDATA can be included)
  - <!ELEMENT product (artno, description)>  
  - Exactly two sub elements in that order
- Each element in the sequence can be qualified further
  - e1* - zero or more occurrences
  - e1? - zero or one occurrences
  - e1+ - at least one occurrence
  - e1|e2 - e1 or e2 (not both)
- <!ELEMENT message (from, to+, txt | ml, data*)>
- What does the above express?
Attribute declarations

‣ <product id="4711" currency="SEK">..</..>
‣ <!ATTLIST element name type default>
  ‣ the attribute name belongs to element, having the type type and the default value default
‣ Possible types (not complete):
  ‣ CDATA - character data, any string (not parsed)
  ‣ NMTOKEN - one token, may start with a digit
  ‣ NMTOKENS - space separated sequence of above.
  ‣ enumeration - list of possible NMTOKENs
‣ Additional attribute types can be useful, but do not increase the expressiveness in terms of making the types more precise.
‣ This means, eg, that a DTD can not say that we want numbers in an attribute, only characters.
‣ <!ATTLIST price amount CDATA #REQUIRED>
‣ We must check ourselves that the amount indeed is a valid number.
Attribute declarations

- The default value is one of
  - #REQUIRED - this attribute must be present
  - #IMPLIED - this attribute need not be present
  - #FIXED "value" - always has the value value
  - "value" - default value if nothing is supplied.

- <!ATTLIST product name CDATA #REQUIRED>
- <!ATTLIST product alias CDATA #IMPLIED>
- <!ATTLIST price currency (SEK|EUR|YEN) "SEK">
DTD example

<!ELEMENT calendar (day*)>
<!ELEMENT day (timeslot|todo)*)>
<!ELEMENT timeslot (from, to, item)>
<!ELEMENT todo (due?, prio?, item)>
<!ELEMENT item (#PCDATA)>
<!ELEMENT prio EMPTY>

...
<!ATTLIST prio level (LOW|NORMAL|HIGH) "LOW">

...
Use of DTD

- A DTD can be included in the document itself.
  - `<!DOCTYPE world [<!ELEMENT ..> ..]>`
- It can also be an external reference.
  - `<!DOCTYPE world SYSTEM “world.dtd”>`
  - `<!DOCTYPE world SYSTEM “http://.../world.dtd”>`
- Also, we want the parser to actually validate the XML document when we read it. This is not on by default.
DTD for the assignment

<!-- Scene is the top element of the xml-file. -->
<!ELEMENT scene (camera, background?, world)>

A scene world consists of a camera, possibly a background and a world, in that order.

<!-- World. -->
<!ELEMENT world (plane | light | sphere | triangle)*)>

The world consists of a lot of objects.

<!-- The camera is described in the assignment. -->
<!ELEMENT camera (location, sky, look_at)>
<!ELEMENT sky (vector)>
<!ELEMENT look_at (vector)>

The camera consists of a location, up and a target. The latter two are vectors.

<!-- Background. -->
<!ELEMENT background (color)>

The background is a colour.
DTD for the assignment

<!ELEMENT sphere (location, pole?, equator?, surface?)>
<!ATTLIST sphere radius CDATA #REQUIRED>

<!ELEMENT pole (vector)>
<!ELEMENT equator (vector)>

The location of a sphere is a sub element, but the radius is an attribute. The pole and equator (used for mapping images) are not mandatory. A surface description need not be present.

<!ELEMENT triangle (c0, ((v1, v2) | (c1, c2)), surface?)>
<!ELEMENT c0 (vector)>
<!ELEMENT c1 (vector)>
<!ELEMENT c2 (vector)>
<!ELEMENT v1 (vector)>
<!ELEMENT v2 (vector)>

The geometry of a triangle can be given either by one corner and two relative vectors or three corners. A surface description need not be present.
A plane is described by a normal and either a distance (from the origin) along the normal or a point on the plane. Post processing must be used to determine which is used.

Distance is given as an attribute but the point as a sub element.

Lights are just points in space; no colour, no volume.

Locations and positions are just different ways of saying the same thing.
DTD for the assignment

<!ELEMENT vector EMPTY> 
<!--
The attributes x, y, and z should contain a real-number in ASCII-format. This cannot be enforced by the DTD, only by schemas. -->
<!ATTLIST vector 
  x CDATA #REQUIRED 
  y CDATA #REQUIRED 
  z CDATA #REQUIRED>

A vector has no sub elements, the coordinates are given in attributes. Post processing must be used to determine that the values are indeed numbers.

<!-- and more.. -->

The rest should be simple to understand, so look at the full DTD if you are in doubt.
Reading XML with SAX

- SAX - Simple API for XML
- A streaming read only API
- streaming - process the document while it is read, alternatively build your own document model during the reading.
- read only - it can only be used to read XML documents, not write them.
- SAX generates “events” when the parser encounters well defined points of the XML document.
- You need to provide an “event handler” in addition to the XML document to read.
- The parser reads the XML document and reports events if an event handler is installed.
- If no event handler is installed, the parser will just check if the document is well formed.
- A parser in SAX needs to implement the XMLReader interface.
- A default parser can be created by XMLReaderFactory.createXMLReader();
- Use the parse() method to read an XML document.
Reading XML with SAX

// Create a parser
try {
    XMLReader parser = XMLReaderFactory.createXMLReader();

    // Set the parser to do validation, i.e, check the DTD
    parser.setFeature("http://xml.org/sax/features/validation", true);

    // Parse the document
    parser.parse(args[0]);

    // All went well if we see this. Otherwise an exception is thrown
    System.out.println(args[0] + " is well-formed and valid.");
} catch {
    // Something failed - you need to handle it!
    ...
}
Reading XML with SAX

- An XML document is a tree rooted at the root element, i.e., the single element present at top level.
- The parser will read the document sequentially and the tree will thus be traversed in a depth first manner.
- We go down one level in the tree when we see a start tag for an element.
- We go up one level in the tree when we see an end tag for an element.
- An element is thus completely traversed only when we see the end tag for the element.
- There can be information in the start tag of an element, i.e., attributes.
- The end tag contributes only with the information that we have reached the end of the element.
- We can see an arbitrary amount of information between the start tag and end tag of an element.
- It is thus not meaningful to report an “element”.
- Instead, SAX will report tags (start/end).
- It is the responsibility of an event handler to react to the tags.
A event handler for SAX implements the ContentHandler interface.
The interface consists of eleven (11) methods.
A method in the content handler is called when the corresponding location in the XML document is seen.
This means that methods in the content handler will be called before the whole XML document has been parsed, validated and traversed.
Be careful with committing anything before the end of the document has been seen (and reported).
An implementation of ContentHandler can choose to ignore events simply by doing nothing when the method is called.
Depending on the actual application more or less of the methods will actually do something useful.
Remember that XML is very general, but a specific XML document/application might not be using the full generality.
For convenience there is an adapter class DefaultHandler that implements every needed method with empty bodies. Extend this class if you just need to implement a small set of handlers.
SAX events/methods

- public void startDocument()
  - called when the start of the document is seen.
  - good place to do any initialisation needed, especially if the same content handler is used for several documents.

- public void endDocument()
  - called when the end of the document is seen.
  - any non reversible processing “should” be done here, not inside any of the other methods.

- void startElement(String namespaceURI, String localname, String qname, Attributes atts)
  - called when the start of an element is seen
  - namespaceURI is “location” of the namespace this element belongs to.
  - localname is the element name without namespace information
  - qname is the fullname as written, ie, namespace and element (with a colon in between).
  - (we’ll come to the attributes later)
SAX events/methods

- If the XML document contains no namespaces you can safely use qname
- If the parser is not namespace aware (this can be turned on or off), namespaceURI and localname are empty strings.
- If the document uses namespaces you must use both namespaceURI and localname to correctly identify an element.
- using only localname is not safe in combination with namespaces.
- In general, there’s not too much interesting work that can be done in startElement as we have not seen the contents yet.
- Bookkeeping will need to be done to make subsequent calls aware of where we are in the document.
- void endElement(String namespace, String localname, String qname)
  - called when the end of an element is seen.
  - here we can do interesting processing as we now have seen the full contents of a element
  - the contents have been processed as sub elements or by the next interesting handler method.
SAX events/methods

- void characters(char[] chars, int start, int length)
  - called to report actual character data, i.e., a leaf in the XML document.
  - chars contains the relevant data, starting at start and being length characters onward.
  - here we convert character data to, e.g., a number.
  - to do so we must be aware of where we are in the XML document and what kind of data to expect.
  - Additional validation, not expressed in a DTD/Schema must be done here.
  - A word of warning:
    - You might not get all character data in one single call to characters.
    - This means that the above is actually false - where should the character data processing really take place?
    - what is the only thing the characters() method can safely do?
    - Is there a “reasonable assumption” you can make to know that you have the whole data?
    - Is the assumption safe, ie, can you rely on it?
  - If you have a DTD with empty leaves this will never be called.
Handling attributes

- `startElement(String uri, String lname, String qname, Attributes attributes)`
- We get access to element attributes when seeing the start of the element.
- Attributes is an interface which provides methods for accessing the element attributes.
- Each attribute has three interesting properties we might want to know about
  - name ("known"), type ("known") and value
  - take care of both namespace and the local name
- `int getLength()` returns the number of attributes
  - makes it possible to obtain attributes by index
  - don’t rely on any particular order!
- `String getQName(int index)`
- `String getURI(int index)`
- `String getLocalName(int index)`
- `String getType(int index)`
- `String getValue(int index)`
Handling attributes

- Attributes can also be retrieved using their name (usual note regarding namespace!)
  - String getType(String URI, String localname)
  - String getType(String qname)
  - String getValue(String URI, String localname)
  - String getValue(String qname)
  - int getIndex(String URI, String localname)
  - int getIndex(String qname)

- Attributes are only passed in startElement, not endElement
- With attributes there is more to do in startElement, since we need to remember them and use them in endElement.
- We can take care of them in much the same way as sub elements.
- The only special thing is that we know somewhat more of an attribute’s type, but in practice it doesn’t make any real difference.
Program Structure

- The bulk of the work is done in endElement.
- The characters() method should just copy characters to an internal buffer.
- Each (relevant) element must be recognised and acted upon [are there elements that are irrelevant?].
- When we see the endElement we know that we’ve seen the sub elements of that element so we can construct the whole element [or act upon it].
- When should we insert things into the world?
- Consider the DTD presented sofar.
  - We have several instances of location and position
  - The parent is not unique.
- In general, we have to know where in the recursive structure we are.

söndag 27 september 2009
public static class GLHandler extends DefaultHandler {
    public void endElement(String ns, String lname, String qname) {
        System.err.println("Time to process element: " + lname);
    }

    ...
}

...

XMLReader parser = XMLReaderFactory.createXMLReader();
parser.setFeature("http://xml.org/sax/features/validation", true);
parser.setContentHandler(new GLHandler());
parser.parse(input);
...