XML

XML

- **XML**
  - DOCTYPE: Document Type Définition (DTD)
  - XML Scheme
- **DOM**
  - Tree-based APIs – DOM
- **SAX**
  - Event-based APIs – SAX
- **XML Parsing**
  - Example in Java XML Parsers
    - JAXP 1.3: Java API for XML Parsing (comes with JDK)
    - Xerces: software that reads XML doc and do something with it

What is XML?

- XML is:
  - XML is a language for creating other languages!
    - meta language that defines other languages (XSSLT)
  - XML lets you define schemas for tag-based languages ("markup language")
  - XML allows you to extend any existing language (schema) with your own tags ("eXtensible")
- Examples of XML schemas
  - Financial transactions (stock transactions)
  - business documents (purchase order, invoice)
  - remote procedure calls (SOAP)
  - configuration files (security, server properties)
  - Italian Leather Store

XML

- The World Wide Web Consortium (W3C) formed an XML working group in 1996 with these design goals:
  1. XML shall be straightforwardly usable over the Internet
  2. XML shall support a wide variety of applications
  3. It shall be easy to write programs which process XML documents
  4. XML shall be compatible with SGML (Standard Generalized Markup Language [ISO 8879])
  5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
  6. XML documents should be human-legible and reasonably clear.
  7. The XML design should be prepared quickly.
  8. The design of XML shall be formal and concise (http://www.w3.org/TR/REC-xml/)
  9. XML documents shall be easy to create.

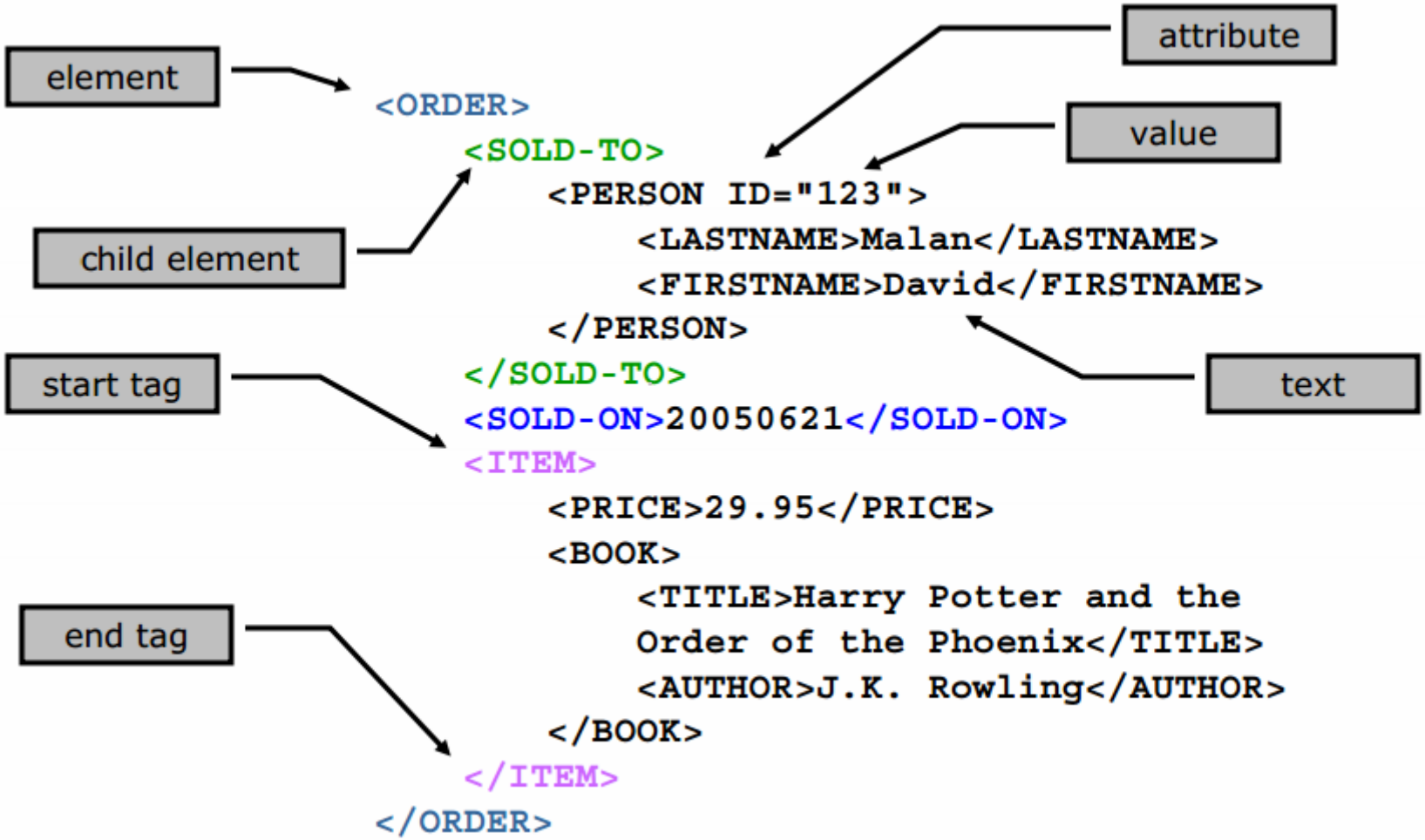
Application & XML Processor

- Definition: A software module called an XML processor is used to read XML documents and provide access to their content and structure
  - Example Xerces 2.9.1
  - Build your own using JAXP/DOM

- Definition: It is assumed that an XML processor is doing its work on behalf of another module, called the application

XML

- When
  - XML 1.0 became a standard (W3C recommendation) on 10 February 1998
  - XML 1.1 became a standard (W3C recommendation) on 4 February 2004
- XML Document
  - [Definition: A data object is an **XML document** if it is **well-formed**, as defined in this specification. In addition, the XML document is **valid** if it meets certain further constraints.]
  - Each XML document has both a **logical** and a **physical** structure.
    - **Physically**, the document is composed of units called entities.
      - An entity may refer to other entities to cause their inclusion in the document.
      - A document begins in a "**root**" or document entity.
    - **Logically**, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup


element
attribute
value
child element
start tag
text
end tag
<ORDER>
<SOLD-TO>
<PERSON ID="123">
<LASTNAME>Malan</LASTNAME>
<FIRSTNAME>David</FIRSTNAME>
</PERSON>
</SOLD-TO>
<SOLD-ON>20050621</SOLD-ON>
<ITEM>
<PRICE>29.95</PRICE>
<BOOK>
<TITLE>Harry Potter and the
Order of the Phoenix</TITLE>
<AUTHOR>J.K. Rowling</AUTHOR>
</BOOK>
</ITEM>
</ORDER>

```
<ORDER>
    <SOLD-TO>
        <PERSON ID="123">
            <LASTNAME>Malan</LASTNAME>
            <FIRSTNAME>David</FIRSTNAME>
            <INITIAL>J</INITIAL>
            <ADDRESS>
                <STREET>Oxford Street</STREET>
                <NUMBER>33</NUMBER>
                <CITY>Cambridge</CITY>
                <STATE>MA</STATE>
            </ADDRESS>
        </PERSON>
    </SOLD-TO>
    <SOLD-ON>20050621</SOLD-ON>
    <ITEM>
        ...
    </ITEM>
</ORDER>
```

extend structure without breaking existing data and applications

Well-Formed XML

- Well-Formed XML Documents: adheres to the XML specifications
  - Correct syntax
  - One root element
  - Start-tag must have an end-tag
  - Respect parent child relationship
    - Child elements are unique within the a parent element
  - "Well formed XML is XML that has **all tags closed** in the proper order and, if it has a declaration, it has it first thing in the file with the proper attributes."

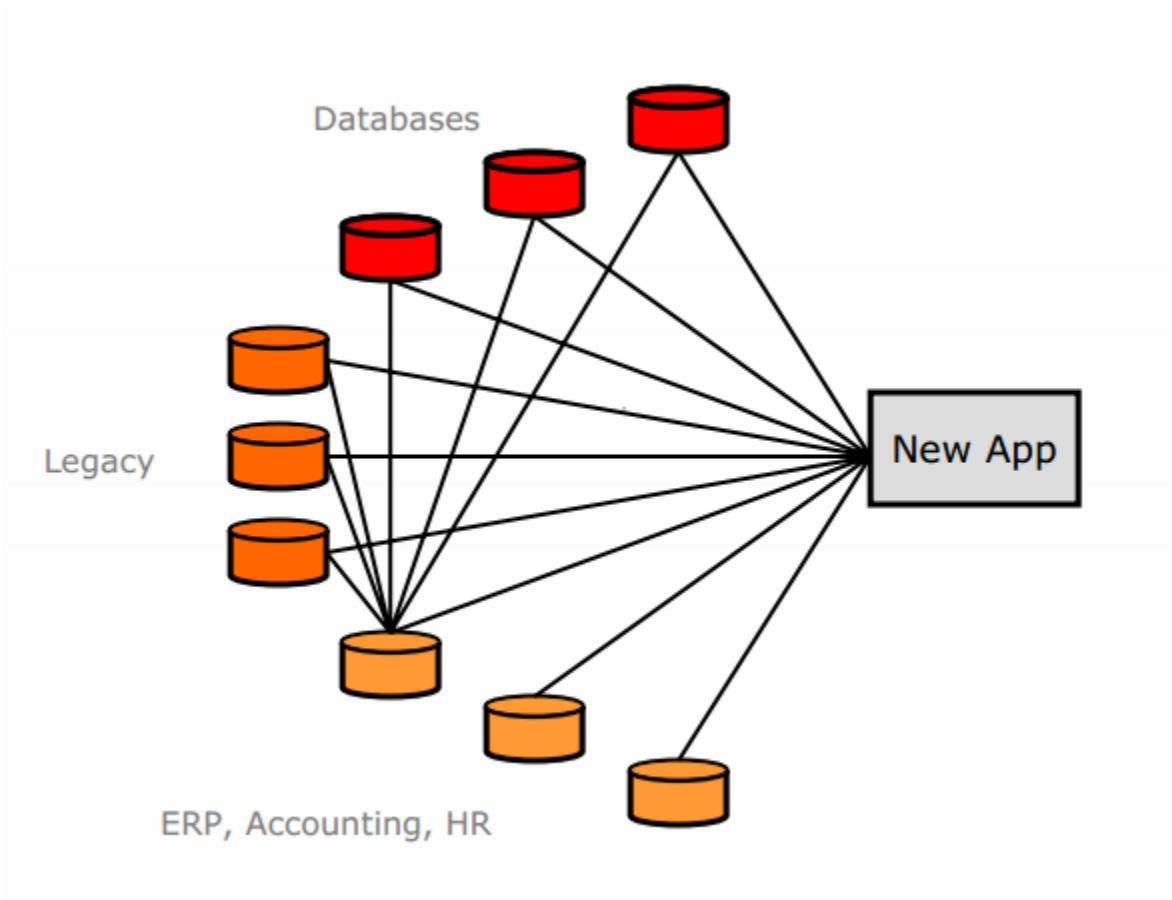
The syntax rules:
•XML documents must have a root element
•XML elements must have a closing tag
•XML tags are case sensitive
•XML elements must be properly nested
•XML attribute values must be quoted

Valid XML Document

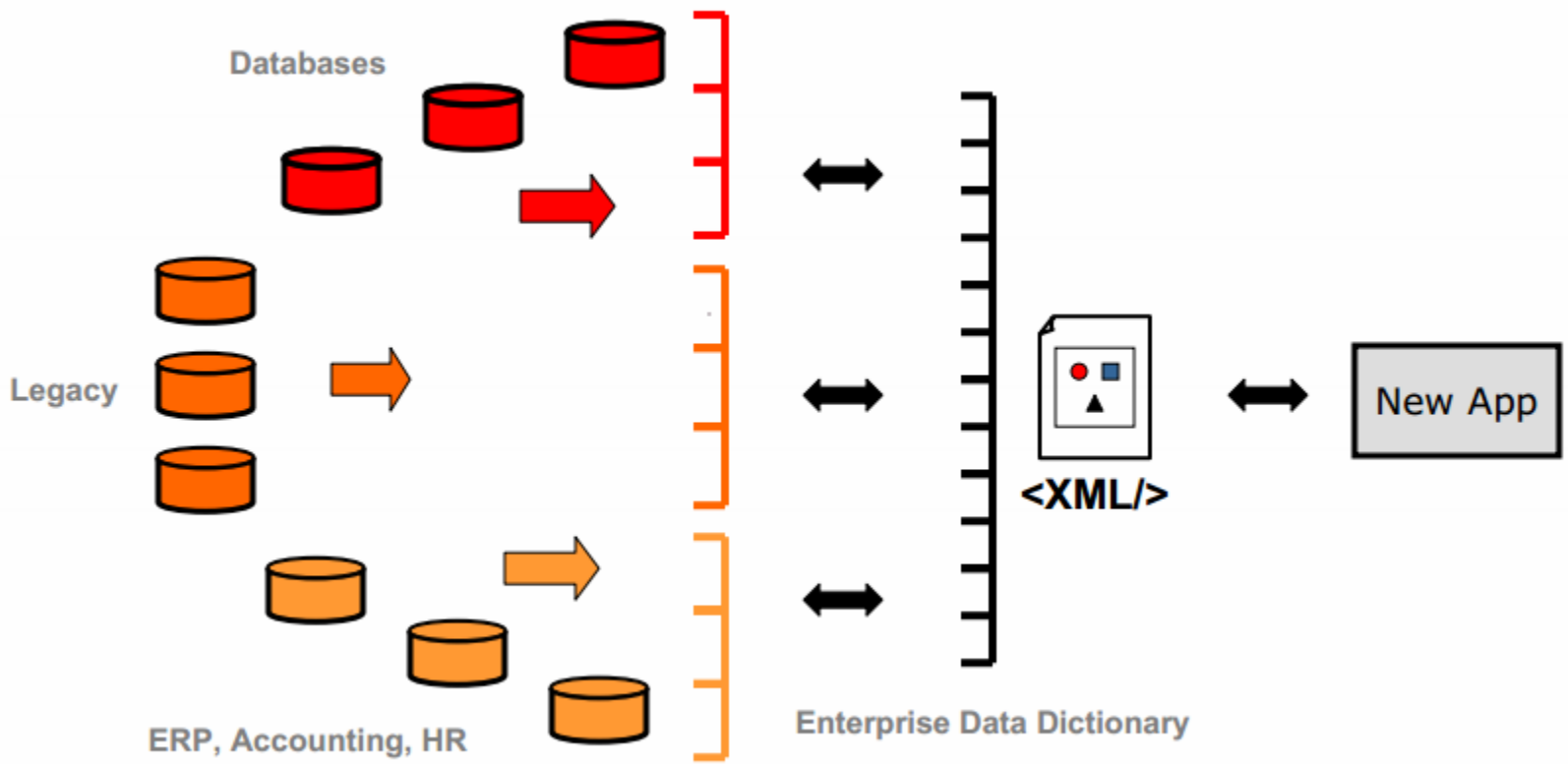
- A "**Valid**" XML document must be:
  1. **Well Formed and**
  2. **Conform to a specified Document Type Definition (DTD)**

  – **Rules** that defines legal elements and attributes for XML documents are often called: **document definitions**, or **document schemas**.

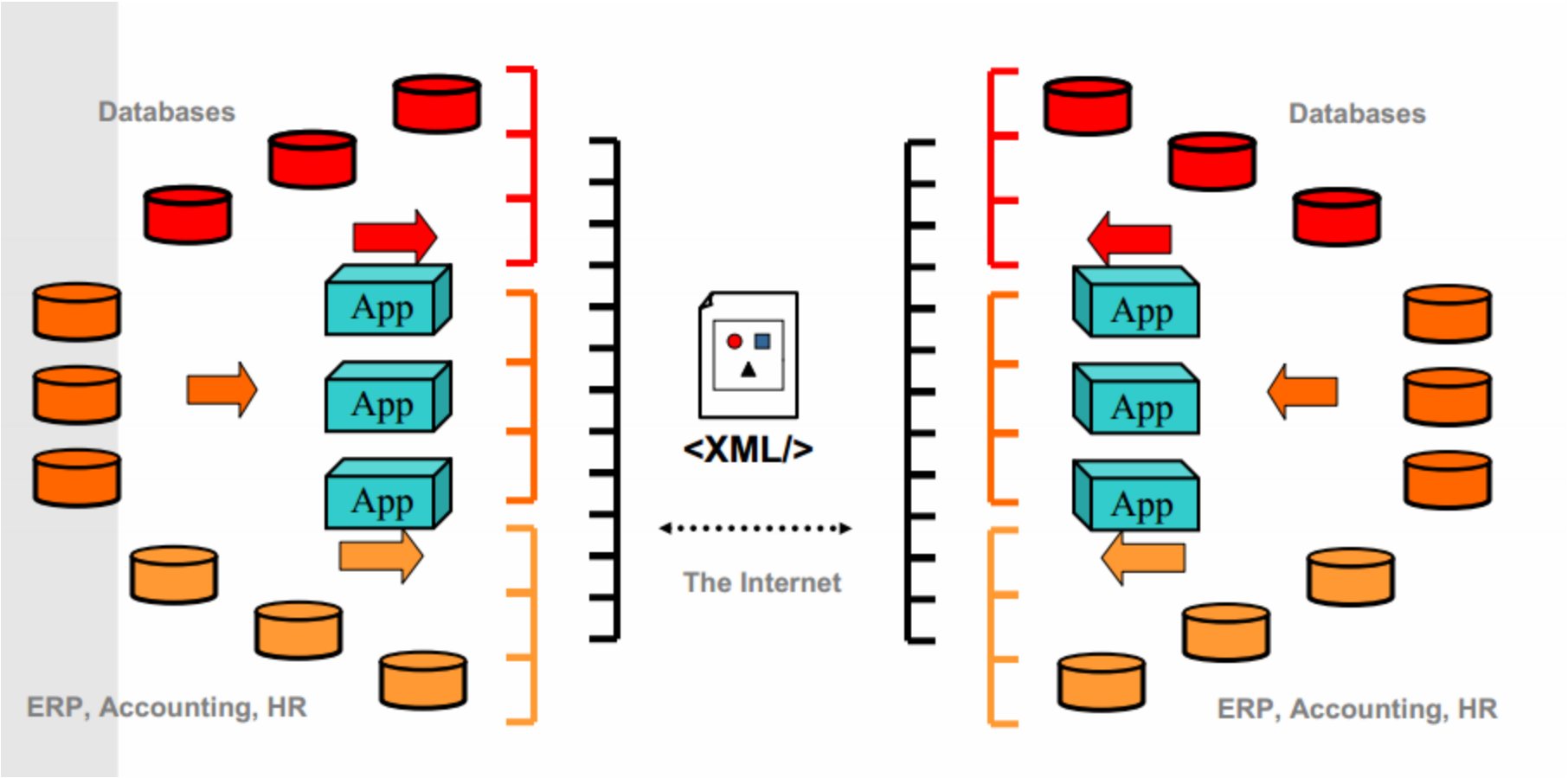
Application Integration



Application Integration



Application Integration



Platform-Independent Services

- Web Services provide a new way to expose functionality
  - use XML and XML data types for transport
  - work with any platform
  - provide a bridge to existing business services

  - XML Enable Platform Independent Services

Introduction to XML

XML

XML Documents

A Representative Document

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE students SYSTEM "student.dtd">

<!-- This is an XML document that describes students -->
<?studentdb displaydesc="true"?>
<students>
        <student id="0001">
                <name>Jim Bob</name>
                <status>graduate</status>
                <dorm/>
                <major>Computer Science &amp; Music</major>
                <description>
                        <![CDATA[ <h1>Jim Bob!</h1>
                        Hi my name is jim.  I look like
                        <img src="jim.jpg"> ]]>
                </description>
        </student>
        <student id="0002">
                ...
        </student>
</students>
```

- **CDATA** means, Character Data. **CDATA** is defined as blocks of text that are not parsed by the parser, but are otherwise recognized as markup

XML 1.1

- XML Declaration
  - <?xml version="1.0" encoding="UTF-8"?>
  - Optional
  - Must appear at the very top of an XML document
  - Used to indicate the version of the specification to which the document conforms (and whether the document is "standalone")
  - Used to indicate the character encoding of the document
    - UTF-8
      - **UTF**-8 is a variable-width encoding that can represent every character in the Unicode character set. It was designed for backward compatibility with ASCII
      - The first 128 characters of Unicode, which correspond one-to-one with **ASCII**
    - UTF-16
    - iso-8859-1

XML 1.1: DOCTYPE

- DOCTYPE
  - <!DOCTYPE **students** SYSTEM "**students.dtd**">
  - Associate an XML document with its definition
    - Can refer to an external DTD file **or**
    - include some DTD information within the tag itself
  - Well formed and Valid XML Document
    - An XML document with correct syntax is called "**Well Formed**".
    - An XML document validated against a DTD is "**Well Formed**" and "**Valid**".
  - The purpose of a DTD is to define the structure of an XML document.
    - It defines the structure with a list of legal elements

XML 1.1: DTD

- DOCTYPE: Document Type Definition (DTD)
  - <?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE **note** SYSTEM "**Note.dtd**">
    <note>
    <to>Tom</to>
    <from>Jane</from>
    <heading>Reminder</heading>
    <body>Don't forget my book!</body>
    </note>

external DTD file

Contents of Note.dtd:

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

XML 1.1: DTD

- The DTD above is interpreted like this:
  - !DOCTYPE **note** defines that the **root element** of the document is *note*
  - !ELEMENT note defines that the note element contains four elements: "to, from, heading, body"
    - !ELEMENT **to** defines the to element to be of type "#PCDATA"
    - !ELEMENT **from** defines the from element to be of type "#PCDATA"
    - !ELEMENT **heading** defines the heading element to be of type "#PCDATA"
    - !ELEMENT **body** defines the body element to be of type "#PCDATA"
- #PCDATA means parse-able text data (Parsed Character Data):

external DTD file

Contents of Note.dtd:

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

XML 1.1: DTD

- A DOCTYPE declaration can also be used to define special characters and character strings, used in the document:

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

<!DOCTYPE note [
<!ENTITY nbsp " ">
<!ENTITY writer "Writer: Donald Duck.">
<!ENTITY copyright "Copyright: W3Schools.">
]>

<note>
<to>Tom</to>
<from>Jane</from>
<heading>Reminder</heading>
<body>Don't forget my book!</body>
<footer>&writer; &copyright;</footer>
</note>
```

An **entity** has three parts: an ampersand (&), an entity name, and a semicolon (;).

© is ©

XML 1.1: DTD

- Why Use a DTD?
  - With a DTD, your XML files can carry a description of its own format.
  - With a DTD, independent groups of people can agree on a standard for interchanging data.
  - With a DTD, you can verify that the data you receive from the outside world is valid.

- DTD: cannot specify data types , use **XML Scheme** to overcome this limitation

XML Scheme

- A set of rules that a document (XML) need to follow
  - <xsd: element name="quantity" type="xsd: integer" />
  - XML Scheme is written in XML
  - XML Schemas are much more powerful than DTDs.
  - XML Schemas Support Data Types

  - You can define your own data type (user defined data types)

XML 1.1: Scheme

XML Scheme file note.xsd

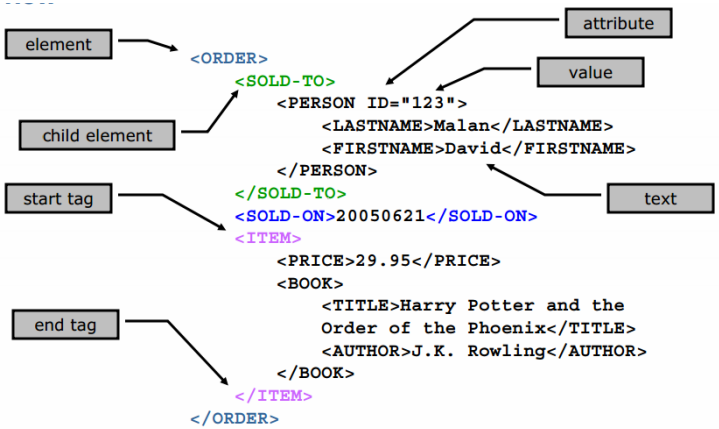
- An XML Scheme
  - <?xml version="1.0" ?>
  - **<note**
    xmlns="http://www.w3schools.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3schools.com/note.xsd">
  - <to>Tom</to>
    <from>Jane</from>
    <heading>Reminder</heading>
    <body>Don't forget my book!</body>
    </note>

The note element is a **complex type** because it contains other elements.
The other elements (to, from, heading, body) are **simple types** because they do not contain other elements

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

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

XML 1.1: Elements



- **Element**
- <name>Mark James</name>
- Main structure in an XML document
- Only **one root element** allowed
- **Start Tag**
  - Allows specification of zero or more attributes
    <student id="0001" ...>
- **End Tag**
  - Must match *name, case, and* ***nesting level*** of start tag
  - </student>

- **Name must start** with <u>letter</u> or <u>underscore</u> and can contain only *letters, numbers, hyphens, periods*, and *underscores*

XML 1.1: Elements

1. **Element Content**

   <student>

   <status>...</status>

   </student>

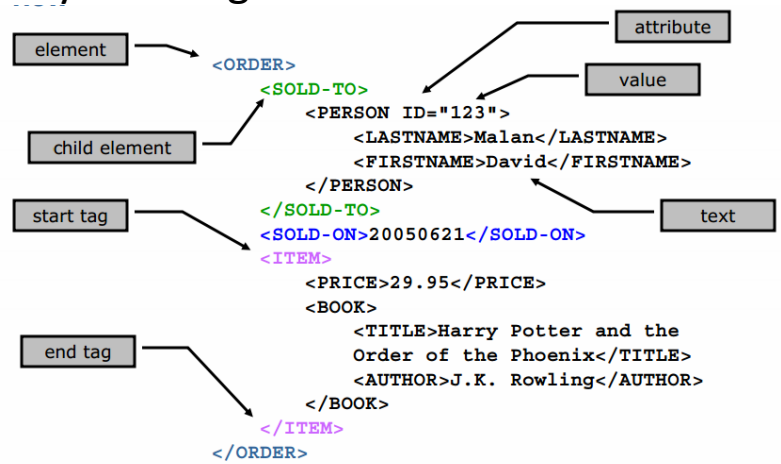
2. Parsed Character Data (aka **PCDATA**, aka Text)
   - PCDATA is text that is parsed char by char
   - <name>Jim Bob</name>
3. Mixed Content
   - <name>Jim <initial>J</initial> Bob</name>
4. No Content
   - Can be useful like line breaks or empty data elements (no value)
   - <dorm/>
     - No data associated with students dorm but we need the element could be mapped to a NULL value in DB for example

XML 1.1: Attributes

- Attributes
  - Describe elements further and can mean anything

- <student **id**="3598">
- Name
  - Must start with letter or underscore and can contain only letters, numbers, hyphens, periods, and underscores
- Value
  - Can be of several types, but is almost always a string
    - Must be quoted
      - title="Lecture 2"
    - Nested
      - match='item="baseball bat"'

- Cannot contain **<** or **&** (by itself)



Discussion: Elements vs. Attributes

- When to use elements and not attributes
  - If Data is extensible (can refine further)
  - Coding implications

```
<student  id="12345" name="Joe Smith">
</student>
```

  - Vs. ??????????

```
<student id="12345">
    <name>Joe smith"</name>
</student>
```

XML 1.1: PCDATA

- #PCDATA - parse-able text data.
  - Called ENTITIES that have predetermined meaning

- Jim Bob
- Text that appears as the content of an element
- Can reference entities
- Cannot contain **<** or **&** (by itself)

XML 1.1: Entities

- Entities
- Five pre-defined entities representing special characters :
  - & < > ' "
- Used to "escape" content or **include** content that is hard to enter or repeated frequently

Name	Character	Unicode code point (decimal)	Standard	Description
quot	"	U+0022 (34)	XML 1.0	double quotation mark
amp	&	U+0026 (38)	XML 1.0	ampersand
apos	'	U+0027 (39)	XML 1.0	apostrophe *(apostrophe-quote)*
lt	<	U+003C (60)	XML 1.0	less-than sign
gt	>	U+003E (62)	XML 1.0	greater-than sign

XML 1.1: Entities

- Character entities
  - can refer to a single character by unicode number
    - e.g., © is ©
  - Must be declared to be legal (in DTD)
    - <!ENTITY nbsp " ">
  -   is not XML (its html) and present coding issues for software like XSSLT which generates HTML form XML

XML 1.1: CDATA

- CDATA
- <![CDATA[<h1>Jim Bob!</h1> ...]]>
- Text that is NOT Parsed (as is)
  - Data within is not checked for subelements, entities, etc.
- Allows you to include badly formed markup or character data that would cause a problem during parsing
- Examples
  - Including HTML tags in an XML document

- In HMTL
  - Javascript code is in an hmtl document to tell the parser/browser not to parse/render it.

XML 1.1: Comments

- Comments
- <!-- This is ... -->
- Can include any text inside a comment to make it easier for human readers to understand your document
- Generally not available to applications reading the document
- Always begin with <!-- and end with -->
- Cannot contain --

XML 1.1: Processing Instructions (PI)

- Processing Instructions (PI)
- <?studentdb displaydesc="true"?>

- "Sticky notes" to applications processing an XML document that explain how to handle content
- The target portion (e.g., studentdb) of a PI indicates the application that is to process this instruction;
  - cannot start with "xml"
- The remainder of the PI can be any text that gives instructions to the application
- Examples
  - Instructions to an application to display different versions of an image
  - Instructions to an application to suppress display of certain content
  - Configuration Parameters

XML docs are Well-Formed

- A well-formed XML document is a document that conforms to the XML syntax rules, like:
  - it must begin with the XML declaration
  - it must have one unique root element
  - start-tags must have matching end-tags
  - elements are case sensitive
  - all elements must be closed
  - all elements must be properly nested
  - all attribute values must be quoted
  - entities must be used for special characters

XML docs are Valid

- Valid XML Documents: Conforms to rules in DTD or XML Scheme
  - XML documents can have a reference to a DTD or to an XML Schema

discussion

1

- Is this valid XML document

- <foo>
- </foo>
- <bar>
- </bar>

2

- Is this valid XML document

- <foo>
- <bar>
- </foo>
- </bar>

3

- Is this valid XML document

- <foo>
- </bar>
- </foo>

XML 1.1 Parsers

TREE BASED VS. EVENT BASED

Events vs. Trees

- There are two major types of XML APIs:
  - **Tree-based APIs**
    - These map an XML document into an internal tree structure, then allow an application to navigate that tree.
    - The Document Object Model (DOM) working group at the World-Wide Web Consortium (W3C) maintains a recommended tree-based API for XML and HTML documents, and there are many such APIs from other sources.
  - **Event-based APIs**
    - An event-based API, on the other hand, reports parsing events (such as the start and end of elements) directly to the application through callbacks,
    - Does not build an internal tree.
    - The application implements handlers to deal with the different events, much like handling events in a graphical user interface.
    - SAX is the best known example of such an API.

Tree-based APIs - DOM

- They normally put a great strain on system resources, especially if the **document is large**
- Many applications need to build their own **strongly typed data structures** rather than using a generic tree corresponding to an XML document
- **in-memory** parse tree
- DOM:
  - Presents an XML document as a **tree-structure**
  - The **XML DOM parser** converts XML into an **XML DOM object** that can be accessed with JavaScript, Java, .Net, etc.

Event-based APIs SAX

- Reports parsing events (such as the start and end of elements) directly to the application through callbacks,
  - The application implements handlers to deal with the different events, much like handling events in a graphical user interface.
- Does not build an internal tree.
- Event-based API provides a simpler, lower-level access to an XML document:
  - you can parse documents much larger than your available system memory

- **SAX** is the best known example of such an API