

Web Application Development



Week 2 XML

Hamilton V. Niculescu

hamilton.niculescu@ncirl.ie

(please add 'App Development' in the subject line when emailing me)

Scenario

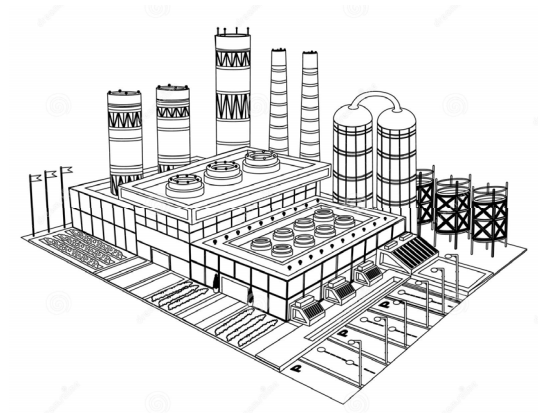


Image source: <https://www.dreamstime.com>

Scenario



Exchange Information about Products

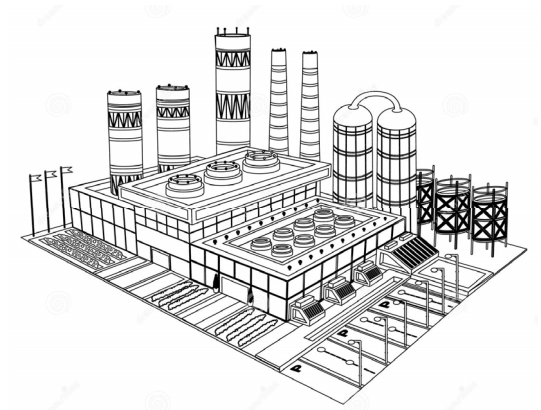
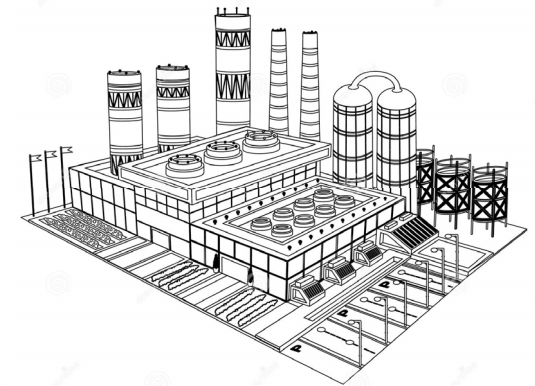


Image source: <https://www.dreamstime.com>

Scenario



Exchange Information about Products



products.txt

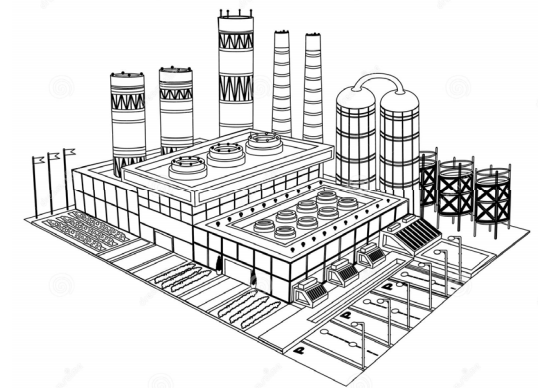
```
Product 1
12.09
1.40
Product 2
100.42
10.23
```

Image source: <https://www.dreamstime.com>

Scenario



Exchange Information about Products



products.txt

```
Product 1  
12.09  
1.40  
Product 2  
100.42  
10.23
```

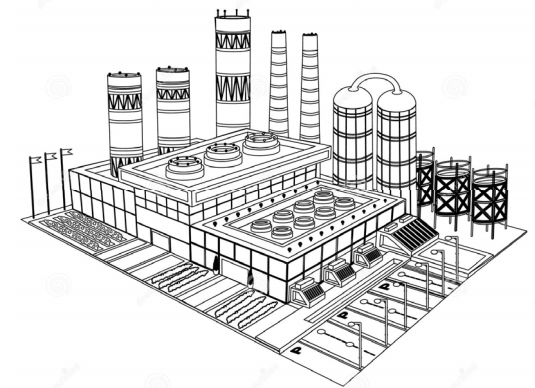
Send the file products.txt



Scenario



Exchange Information about Products



Send the file products.txt

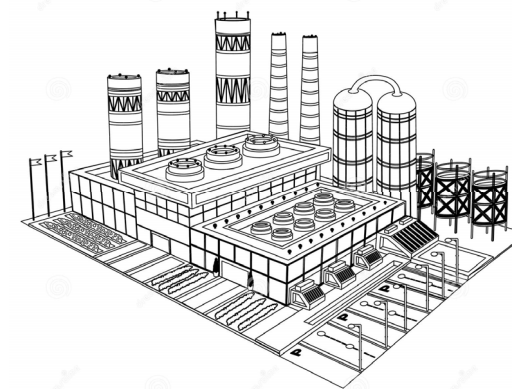


```
Product 1
12.09
1.40
Product 2
100.42
10.23
```

Scenario



Exchange Information about Products



Send the file products.txt



products.txt

```
Product 1  
12.09  
1.40  
Product 2  
100.42  
10.23
```

- This file contains product information
- Each product consumes 3 lines
- Line 1: Name, Line 2: Price, Line 3: Tax

Image source: <https://www.dreamstime.com>

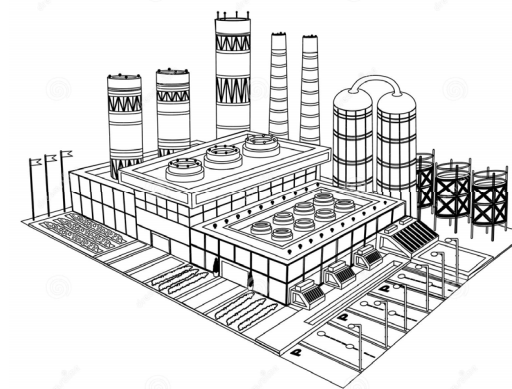
Scenario

Problems with this approach:

1. Need to define the **syntax rules**
2. Need to define the **semantics**



Exchange Information about Products



Send the file products.txt



products.txt

```
Product 1  
12.09  
1.40  
Product 2  
100.42  
10.23
```

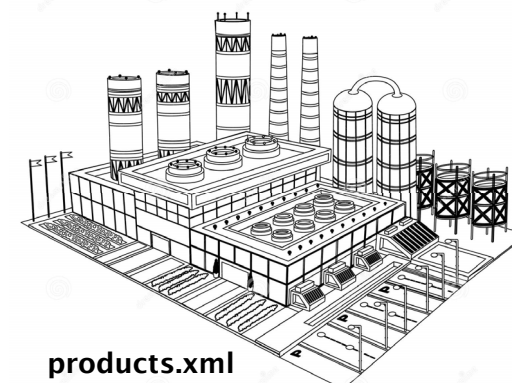
- This file contains product information
- Each product consumes 3 lines
- Line 1: Name, Line 2: Price, Line 3: Tax

Image source: <https://www.dreamstime.com>

Scenario



Exchange Information about Products



products.xml

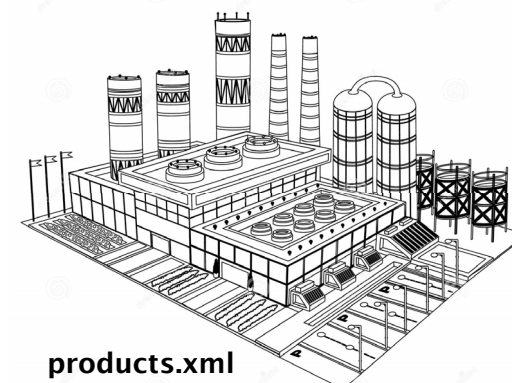
```
<products>
  <product>
    <name>Product 1</name>
    <price>12.09</price>
    <tax>1.40</tax>
  </product>
  <product>
    <name>Product 2</name>
    <price>100.42</price>
    <tax>10.23</tax>
  </product>
</products>
```

Image source: <https://www.dreamstime.com>

Scenario



Exchange Information about Products



Easier to interpret the content of the document because

- uses the standard XML syntax rules
- Tag names define the content semantics

Send the file products.xml



```
<products>
  <product>
    <name>Product 1</name>
    <price>12.09</price>
    <tax>1.40</tax>
  </product>
  <product>
    <name>Product 2</name>
    <price>100.42</price>
    <tax>10.23</tax>
  </product>
</products>
```

Image source: <https://www.dreamstime.com>

Constraints

- Assume that the Corporation automatically uploads the received product files from the Factory into the Enterprise Resource Planning (ERP) system
- They want to **guarantee that the structure and content of the XML files** are valid, thus they do not cause any error when processed by the ERP system
- The XML file structure and content expected by the ERP system is
 - Multiple products
 - Product's Name is a string
 - Product's Price and Tax are decimals
- These constraints will allow
 - Factory to check if the generated files are valid
 - Corporation to check if there was no error during the file transmission and if the file is valid

Scenario

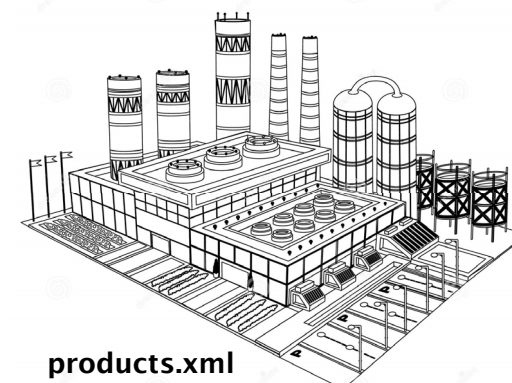


Exchange Information about Products



```
<xs:schema version="1.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="products">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="product">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string" />
              <xs:element name="price" type="xs:decimal" />
              <xs:element name="tax" type="xs:decimal" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



```
<products>
  <product>
    <name>Product 1 </name>
    <price>12.09</price>
    <tax>1.40</tax>
  </product>
  <product>
    <name>Product 2</name>
    <price>100.42</price>
    <tax>10.23</tax>
  </product>
</products>
```

Image source: <https://www.dreamstime.com>

Scenario

```
<xs:schema version="1.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="products">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="product">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string" />
              <xs:element name="price" type="xs:decimal" />
              <xs:element name="tax" type="xs:decimal" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

products.xml

```
<products>
  <product>
    <name>Product 1</name>
    <price>12.09</price>
    <tax>1.40</tax>
  </product>
  <product>
    <name>Product 2</name>
    <price>100.42</price>
    <tax>10.23</tax>
  </product>
</products>
```

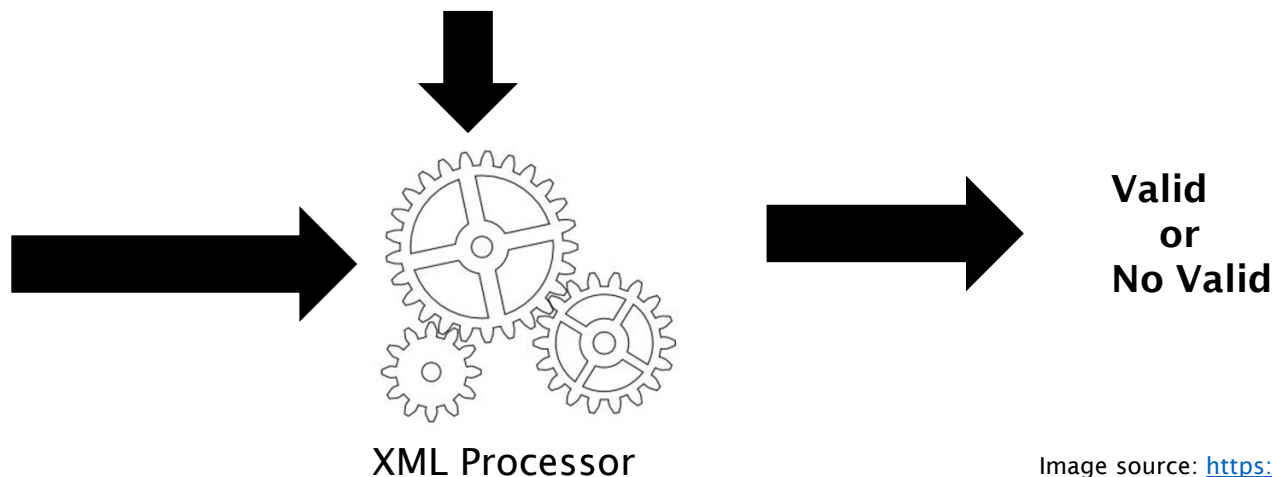


Image source: <https://paintingvalley.com>

What is XML?

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed **to carry data**, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML is a W3C Recommendation

XML Does Not Do Anything!

- Maybe it is a little hard to understand, but XML does not DO anything. XML was created to structure, store, and transport information.
- ```
<note>
< to>Tove</to>
< from>Jani</from>
< heading>Reminder</heading>
< body>Don't forget me this weekend!</body>
< /note>
```

Starting and Closing tags in various colours clearly describe the meaning of the information.
- The note above is quite self-descriptive. It has sender and receiver information, it also has a heading and a message body.
- But still, this XML document does not DO anything. It is just information wrapped in tags. We must write a piece of software to send, receive or display it.

# XML



- Extensible Markup Language (XML) is a markup language that defines **a set of rules for encoding documents** in a format that is both **human-readable** and **machine-readable**.
- It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications, all gratis open standards.
- It has undergone minor revisions since then, without being given a new version number, and is currently in its fifth edition as published on November 26, 2008.
- Many application programming interfaces (APIs) have been developed for software developers to use and to process XML data, and several schema systems exist to aid in the definition of XML-based languages.



# Why XML?

- XML allows us to share and exchange data over networks and between applications.
- XML is **text-based**, **portable**, CPU and OS independent.
- No special tools are needed (although they can help!).
- It has a low cost of entry: a simple standardized format.
- It separates description of content from behaviour or formatting, so that one XML file can be used in many ways.
- Network effect: every new XML document or tool increases the value of all the others.



- 
- A young boy with glasses and a red bow tie is giving a thumbs up in front of a chalkboard. The chalkboard is filled with various words in different languages, including "Hello", "CIAO", "GODDAG", "Zdrav", "cześć", "PIPIBET", "Ahoj", "namaste", "hoy", "mahalo", "halla", "Din", "ma le le", "sveiki", "lorem", "D", "A", "du", "it", "Din", "ma le le", "sveiki", "lorem", "D", "A", "du", "it", "Din", "ma le le", "sveiki", "lorem", "D", "A", "du", "it".

# The Benefits of XML



- Programmers structure data in an infinite variety of ways.
- With every new way of structuring data comes a new methodology for extracting the information we need.
- If the data changes, the methodologies also have to change, and testing and tweaking begin again.
- With XML, there is a standardized way to get the information, no matter how we structure it.
- XML is used in all major websites.
- Many software applications (Excel and Word), and server languages (Java, .NET, Perl, PHP) can read and create XML files.

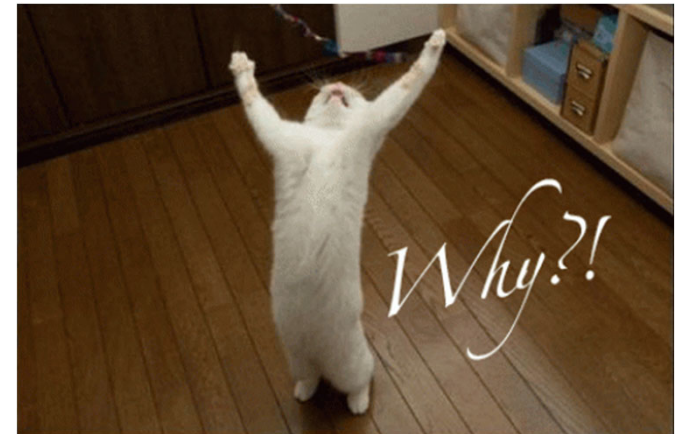
# Where is XML used?

- Store data in the applications
- Share and reuse data in common file formats (e.g., Open Document Format)
- Website content
- Distributed computing
- e-Commerce (e.g., B2B)
- All major databases can read and create XML files
- Mobile device platforms (Google's Android and Apple's iOS) use XML in a variety of ways



# Why 'eXtensible'?

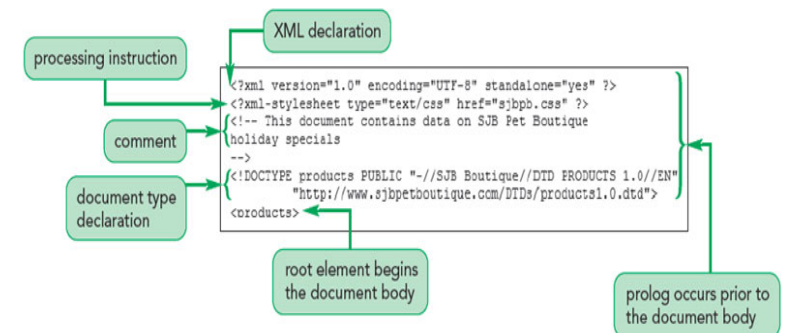
- We have control over how to structure our XML files
- The data can be marked-up in any way we choose
- Agreed standard formats to perform common tasks that can make data exchange between applications easier



# HTML & XML

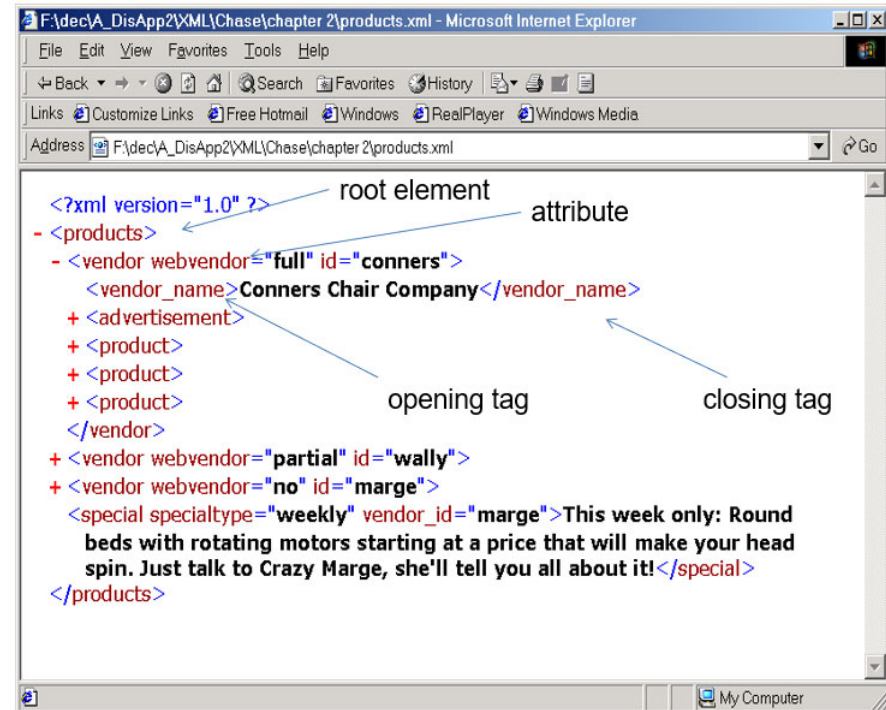
- HTML for content display
- CSS for style formatting and presentations
- XML for data exchange
- HTML is designed for a specific application and to convey information visually in a browser
- Because HTML has a specific application it is also restricted by having a specific set of finite markup constructs that are used to create an HTML document

```
<?xml version="1.0" standalone="yes"?>
<BankAccount>
 <Number>1234</Number>
 <Type>Checking</Type>
 <OpenDate>11/04/1974</OpenDate>
 <Balance>25382.20</Balance>
 <AccountHolder>
 <LastName>Singh</LastName>
 <FirstName>Darshan</FirstName>
 </AccountHolder>
</BankAccount>
```



# Basic Structure of XML

- Trees
- Root, Branches and Leaves
- One Root Element
- Well Formed
  - All elements must have matching **start** and **end tags**.
  - All attributes must be in quotes.
  - All elements must nest properly



The screenshot shows a web browser window displaying an XML file. The XML content is as follows:

```
<?xml version="1.0" ?>
<products>
 <vendor webvendor="full" id="conners">
 <vendor_name>Connors Chair Company</vendor_name>
 <advertisement>
 <product>
 <product>
 <product>
 </vendor>
 <vendor webvendor="partial" id="wally">
 <vendor webvendor="no" id="marge">
 <special specialtype="weekly" vendor_id="marge">This week only: Round
 beds with rotating motors starting at a price that will make your head
 spin. Just talk to Crazy Marge, she'll tell you all about it!</special>
 </products>
 </vendor>
 </product>
 </advertisement>
 </vendor>
 </products>
```

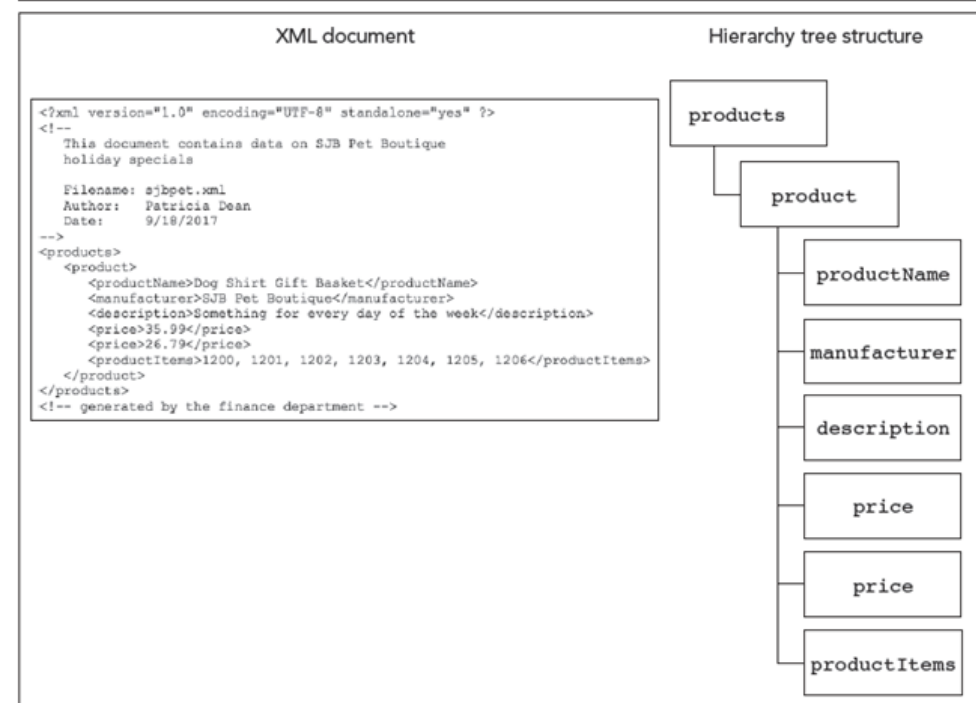
Annotations in the image point to specific parts of the XML:

- root element**: Points to the opening `<products>` tag.
- attribute**: Points to the `id="conners"` attribute in the `<vendor>` tag.
- opening tag**: Points to the `<product>` tag.
- closing tag**: Points to the `</product>` tag.

# The Element Hierarchy

- Better to group large amounts of data into related sub-topics (rather than a single blob of information)
- Developers can structure data into an object model
- Objects in the object model are placed in a hierarchical structure

Code for an XML document along with its corresponding tree diagram

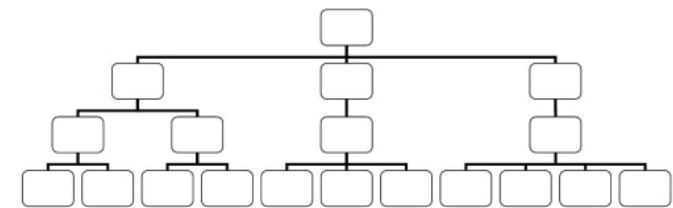




# XML: Hierarchies of Information

- XML also groups information in hierarchies. The items in a document relate to each other in a parent/child and sibling/sibling relationships. e.g.,

```
<name>
----- <first>
----- "John"
----- <middle>
----- "Fitzgerald Johansen"
----- <last>
----- "Doe"
```



# XML Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<CATALOG>
 <CD>
 <TITLE>Empire Burlesque</TITLE>
 <ARTIST>Bob Dylan</ARTIST>
 <COUNTRY>USA</COUNTRY>
 <COMPANY>Columbia</COMPANY>
 <PRICE>10.90</PRICE>
 <YEAR>1985</YEAR>
 </CD>
 <CD>
 <TITLE>Hide your heart</TITLE>
 <ARTIST>Bonnie Tyler</ARTIST>
 <COUNTRY>UK</COUNTRY>
 <COMPANY>CBS Records</COMPANY>
 <PRICE>9.90</PRICE>
 <YEAR>1988</YEAR>
 </CD>
</CATALOG>
```

*This is an  
example ☺*

# Stylesheets

- To format and display the information, **stylesheets** are used.
- By combining **stylesheets** with XML data you can separate your data from your presentation.



**KEEP  
CALM  
AND  
GANGNAM  
STYLE**

```
1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
2 "http://www.w3.org/TR/html4/strict.dtd">
3 <html>
4
5 <!-- Fig. 4.1: inline.html -->
6 <!-- Using inline styles -->
7
8 <head>
9 <title>XML How to Program - Inline Styles</title>
10 </head>
11
12 <body>
13
14 <p>This text does not have any style applied to it.</p>
15
16 <!-- The style attribute allows you to declare inline -->
17 <!-- styles. Separate multiple styles with a semicolon. -->
18 <p style = "font-size: 20pt">This text has the font-size
19 style applied to it, making it 20pt.</p>
20
21 <p style = "font-size: 20pt; color: #0000ff">This text has the
22 font-size and color styles applied to it,
23 making it 20pt. and blue.</p>
24
25 </body>
26 </html>
```

Define style for following text

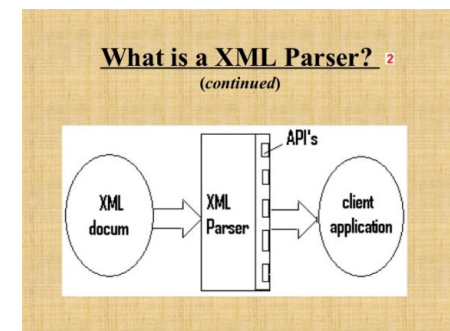
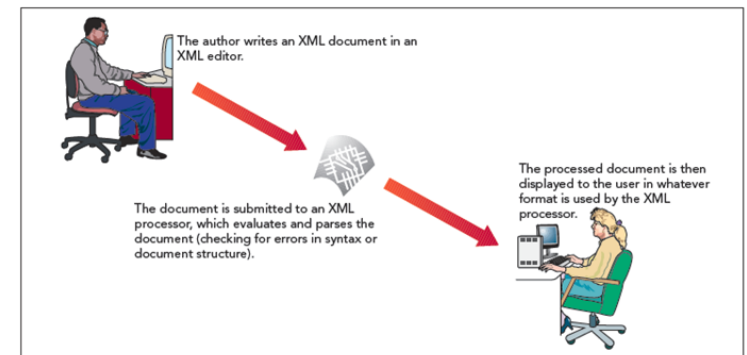
# Stylesheets

- The two most common ways of creating style sheets are using a stylesheet language called
  - **Cascading Style Sheets (CSS)** and
  - A transformation stylesheet language called **Extensible Stylesheet Language Transformation (XSLT)**.



# XML Parsers

- Parser (or processor) is a program that reads and interprets an XML document.
- Parser interprets a document's code and verifies that it satisfies all the XML specifications for document structure and syntax.
- All major web browsers include an XML parser.
- There are different types of parsers, and each has its own advantages.
- The main types of parsers are known by some names: **SAX**, **DOM** and **pull**. SAX stands for Simple API for XML.



# XML Parsers

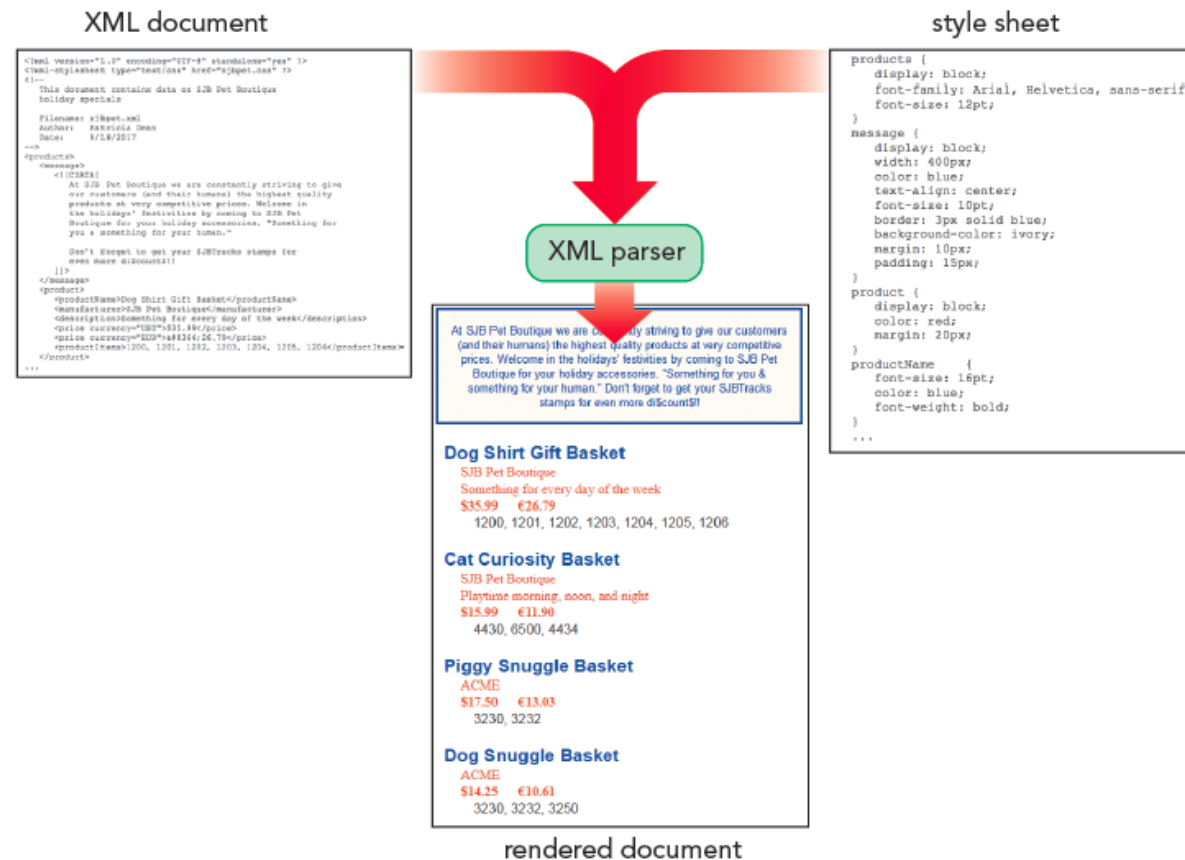
- For example, an XML parser given the file

```
<name>
 <first>John</first>
 <middle>Miles</middle>
 <last>Turner</last>
</name>
```

- can tell us there is a piece of data called <middle> and the information stored there is “Miles”. The parser writer did not have to know any rules about where the first name ends and the middle name begins.
- The same parser can be used to determine the data values for any XML document.
- XML data files could be modified to hold new pieces of information without requiring modification to the underlying application programs.

# Formatting XML Data with CSS

Figure 1-34 Combining an XML document and a style sheet





# What is a Document Type?

- A specific set of elements called a vocabulary, used to mark up the data.
- A document type is a document that is structured in a specific way using a specific vocabulary, to describe a certain type of information.
- **For example**, the **vocabulary** in the previous example is <name>, <first>, <middle> and <last>.
- **Document Type Definition (DTD)** is a collection of rules that define the content and structure of an XML document.

# The Pieces that Make Up XML

- **XML**
  - the base specification upon which the XML family is built. It describes the syntax that XML documents and XML parsers have to follow
- **DTD's (document type definition) and XML Schemas**
  - provide ways to create **templates** or **structures** for documents
- **NameSpaces**
  - provide a means to distinguish one XML vocabulary from another when they are combined into one document type
- **XPath**
  - describes a querying language for addressing parts of an XML document
- **XSL (eXtensible Stylesheet Language)**
  - XSL is used to display XML documents
  - XSL is used for more complex cases (uses XSLT to transform one document into another)
- **XLink & Xpointer**
  - used to link XML documents together

# Basic XML File Structure

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE airlocksystem SYSTEM "airlocks.dtd">
<airlocksystem>
 <airlock lockid="A23b">
 <size height="320" width="500"/>
 <type>Bronson</type>
 <location>Level 2 aft</location>
 <status>open</status>
 </airlock>
 <airlock lockid="Q36b">
 <size height="200" width="300"/>
 <type>Perch</type>
 <location>Level 15 starboard</location>
 <status>closed</status>
 </airlock>
</airlocksystem>
```

XML Declaration

DOCTYPE Declaration

Root Element

Element

Nested Element

Attribute

# XML Declaration

<?xml	Indicates the start of an XML document
version="1.0"	Information included in anticipation when more XML versions exist. Allow applications to reject a version it doesn't understand
encoding="ISO-8859-1"	XML allows the use of several international character sets. ISO-8859-1 represents Latin-1 character set. Most common used with XML is UTF-8. <b>UTF-8 includes encodings for most of the world's common alphabets</b>
standalone="no"	Determines if the document contains external entities such as Document Type Definitions (DTD)

# XML Declaration

- It is not mandatory for an XML document to have an XML declaration – it is, however, a good practice.
- If an XML declaration is present then it must be the first piece of information in the document.
  - No whitespace
  - No comments, etc.
- The version attribute
  - version="1.0" will be appropriate for most cases

# DOCTYPE Declaration



```
<!DOCTYPE airlocksystem SYSTEM
"airlocks.dtd">
```

- A DTD (Document Type Definition) file specifies rules for how the XML document elements, attributes, and other data are defined and logically related.
- Optional, unless you want to verify the content through validation.
- Validation entails specifying a file that contains a series of definitions to which the document must conform.
- Definitions may be in
  - the XML document (internal)
  - a separate file (external)

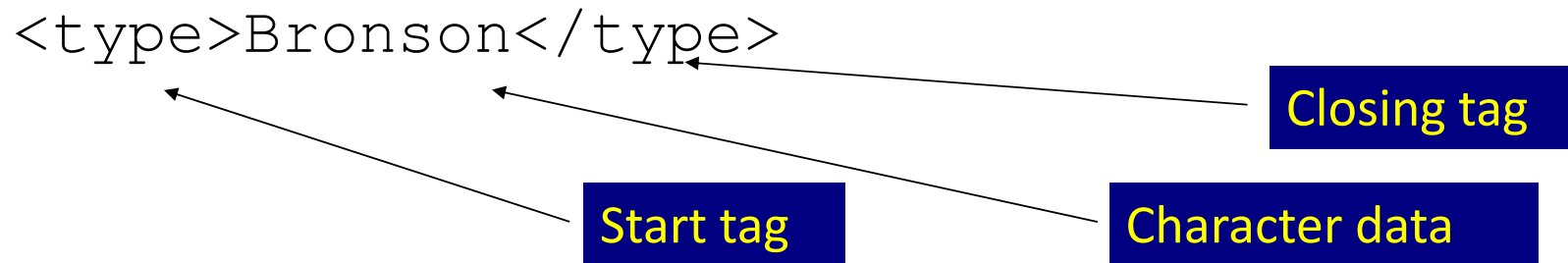
# Root Element

<airlocksystem>

- XML Documents must consist of a single root element.
- All other information will appear between the **start tag** and **end tag**

</airlocksystem>

# Elements



- An XML element is everything from (including) the element's start tag to (including) the element's end tag.



# Elements

`<type>Bronson</type>`

- Must start with a **letter** or underscore character \_
- End tag must contain the same name as the start tag (case sensitive)
- Names should not contain colons
- Names must not contain spaces
- A non-empty element must have an opening (start) tag and a closing (end) tag

# Empty Elements

- An empty element has no content
- Empty elements are useful for specifying non-textual content.
- There is a special shortened syntax we can use for empty elements
  - We can use a single element tag that begins with < but ends with />.

`<ReturnDate></ReturnDate>`

is equivalent to

`<ReturnDate/>`

# Elements

- Element tags must be nested correctly

`<italic><bold>This is incorrect</italic></bold>`

X

`<italic><bold>This is correct</bold></italic>`

✓

- Elements can contain text, other elements, or a combination of both.
- XML is a case sensitive
  - E.g., `<person>`, `<Person>`, and `<PERSON>` are all distinct tag names



# Whitespace

```
<?xml version="1.0"?>
<airlocksystem>
 <airlock lockid="Q36b">
 <size height="200" width="300"/>
 <type>Perch</type>
 <location>Level 15 starboard</location>
 <status>closed</status>
 </airlock>
</airlocksystem>
```

Pretty Printed Document

Easier for humans to read

```
<?xml version="1.0"?><airlocksystem><airlock lockid="Q36b">
<size height="200" Width="300"/><type>Perch</type>
<location>Level 15 starboard</location><status>closed</status>
</airlock></airlocksystem>
```

NOTE: Space and tabs are considered character data within the elements

# Attributes

```
<size height="200" width="300"/>
```

- Name-value pair e.g., height = “200”
- Attributes are used to further refine or modify the default behaviour of an element
- Attribute names must conform to the same rules as Element names
- The names must be unique within a specific element

# Processing Instructions

```
<?xml-stylesheet href="airlocks.xml" type="text/xsl"?>
```

Instructions

Target

Start of processing instruction

- The browser interprets `<?` As the start of the processing instruction
- The target is an XML stylesheet
- The instructions for formatting the XML data are in the `airlocks.xml` file
- Can be used to pass information to particular applications that may read the document.

# Processing Instructions

- A processing instruction begins with `<?` and ends with `?>`
- Immediately following the `<?` is an XML name called the target
  - The target is possibly the name of the application for which this processing instruction is intended or possibly just an identifier for this particular processing instruction.
  - Another example is mentioned below

- To attach a CSS style sheet to an XML document, insert the processing instruction `<?xml-stylesheet type="text/css" href="url" media="type" ?>` within the XML document's prolog, where *url* is the name and location of the CSS file, and the value of the optional *media* attribute describes the type of output device to which the style sheet should be applied. If no *media* value is specified, a default value of *all* is used.

# Processing Instructions

- The rest of the processing instruction contains text in a format appropriate for the applications for which the instruction is intended.

- **Example:**

```
<?robots index="yes" follow="no"?>
```

- Could be used to indicate to search-engine robots that the document should be indexed but that links should not be followed.



# Comments

```
<!-- This is a Comment -->
```



Start

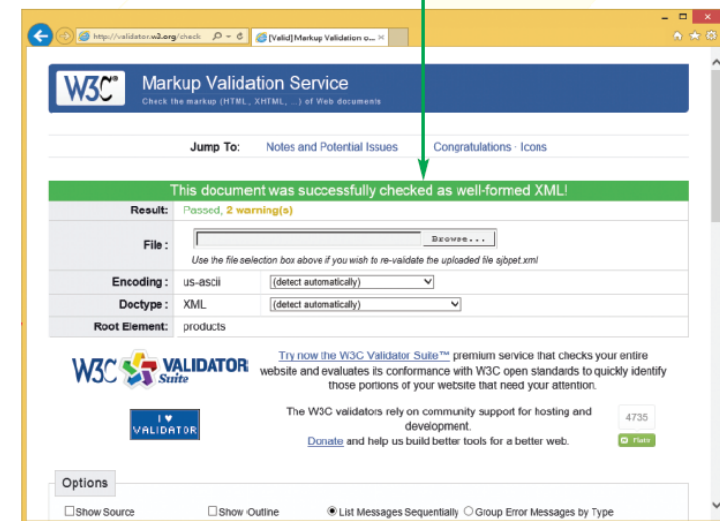
End

- Ignored by the processor
- Use if you want to
  - Add information that is intended to be read by a human
  - Remove sections of the XML document
- Comments may not appear inside a tag or inside another comment.

# Well-Formed XML

- Every XML document must be well-formed.
- An XML document must adhere to certain formatting rules.
- A well-formed document has no syntax errors and satisfies the general specifications for XML code defined by the World Wide Web Consortium (W3C).

A well-formed document has no syntax errors and satisfies the general specifications for XML code defined by the World Wide Web Consortium (W3C).



Copyright © 2014 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University, Beihang). All Rights Reserved.  
<http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231>



# XML Syntax Rules

Syntax Rule	Application
<b>Every XML element must have a closing tag.</b>	Every element must have a closing tag. A self-closing tag is permitted.
<b>XML tags are case sensitive.</b>	Opening and closing tags (or start and end tags) must be written with the same case.
<b>XML elements must be properly nested.</b>	All elements can have child (sub) elements. Child elements must be in pairs and be correctly nested within their respective parent element.
<b>Every XML document must have a root element.</b>	Every XML document must contain a single tag pair that defines the root element. All other elements must be nested within the root element.
<b>XML elements can have attributes in name-value pairs.</b>	Each attribute name within the same element can occur only once. Each attribute value must be <b>quoted</b> ("").
<b>Some characters have a special meaning in XML.</b>	The use of certain characters is restricted. If these characters are needed, entity references or character references may be used. References always begin with the character "&" (which is specially reserved) and end with the character
<b>XML allows for comments.</b>	Comments cannot occur prior to the XML Declaration. Comments cannot be nested.

## Highlights of XML syntax rules

# Well-Formed XML



- Rules include, but not limited to
  - Every start-tag must have a matching end-tag
  - Elements must be nested correctly – they cannot overlap
  - Only a single root element is allowed
  - Attribute values must be quoted
  - An element cannot have two attributes with the same name
  - Comments and processing instructions are not allowed to appear within tags
  - The characters < and & may not appear (unescaped) in the character data of an element

# Task

- Create your own XML document
- The XML document must contain information regarding your timetable:
  - Course
  - Year
  - Module
  - Time table for module
  - Lecture or Lab
  - Etc.



# Namespaces

- **Namespace**—a defined collection of element and attribute names
- Namespaces serve two purposes
  - Distinguish between elements from different vocabularies with different meanings that happen to share the same name
  - Group all the related elements from a single XML application together so that they can be easily recognised
- We implement namespaces by attaching **prefixes** to elements and attributes
  - e.g., product:description



# Namespaces

- Each prefix is mapped to a URI in the following way:

`xmlns:prefix="URI"`

- Example:

`xmlns:product="http://www.mycompany.com/product"`

# Namespaces

- Default URIs can be used for elements that don't have a prefix in the following way: `xmlns="URI"`  
e.g., `xmlns="http://www.mycompany.com/main"`
- Generally, all the elements from one XML application are assigned to one URI, and all the elements from a different XML application are assigned to a different URI.
- Namespaces can be declared in the elements where they are used or in the XML root element.



# Namespaces

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
```

Namespace

Alias

Element

- Alias (xsl) refers to the namespace
- The namespace is at the URI location
  - URI is only for uniqueness and doesn't necessarily refer to a Web address.
  - Good idea to include namespaces particularly if the data will be combined with data from other sources.

# Namespaces

Suppose we have an XML document containing some HTML source code such as

```
<table>
 <tr>
 <td>Crazy Furniture Ltd.</td>
 <td>Insanely Expensive</td>
 </tr>
</table>
```

... and some XML from another document such as

```
<table>
 <type>Coffee</type>
 <material>Plastic</material>
 <price ccy="EUR">17865.99</price>
</table>
```

- We'd like to combine the documents into a single document.
- How can we distinguish the HTML table tag from the table tag used to describe an actual piece of furniture?

# Namespaces

We could use prefixes to avoid the *name conflict*. Let's prefix the HTML table tag with the prefix *htm*

```
<htm:table>
 <htm:tr>
 <htm:td>Crazy Furniture Ltd.</htm:td>
 <htm:td>Insanely Expensive</htm:td>
 </htm:tr>
</htm:table>
```

... and let's prefix the furniture table tag with *furn*

```
<furn:table>
 <furn:type>Coffee</furn:type>
 <furn:material>Plastic
 </furn:material>
 <furn:price ccy="EUR">17865.99</furn:price>
</furn:table>
```

- No more name conflict!
- When using a prefix like this in XML, we must also define a namespace for the prefix.

# Namespaces

- Our complete XML file will then be:

```
<root>
 <htm:table xmlns:htm="http://www.crazyfurniture.com/co/html">
 <htm:tr>
 <htm:td>Crazy Furniture Ltd.</htm:td>
 <htm:td>Insanely Expensive</htm:td>
 </htm:tr>
 </htm:table>

 <furn:table xmlns:furn="http://www.crazyfurniture.com/co/furniture">
 <furn:type>Coffee</furn:type>
 <furn:material>Plastic</furn:material>
 <furn:price ccy="EUR">17865.99</furn:price>
 </furn:table>
</root>
```

# Namespaces

- We could also put the namespace definitions in the root element tag:

```
<root xmlns:htm="http://www.crazyfurniture.com/co/html"
 xmlns:furn="http://www.crazyfurniture.com/co/furniture">

 <htm:table>
 <htm:tr>
 <htm:td>Crazy Furniture Ltd.</htm:td>
 <htm:td>Insanely Expensive</htm:td>
 </htm:tr>
 </htm:table>

 <furn:table>
 <furn:type>Coffee</furn:type>
 <furn:material>Plastic</furn:material>
 <furn:price ccy="EUR">17865.99</furn:price>
 </furn:table>
</root>
```

# Default & Multiple Namespaces

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<airlocksystem xmlns="http://www.example.com/airlock/"
 xmlns:ls="http://www.example.com/lifesupport/">
 <airlock lockid="A23b">
 <size height="320" width="500"/>
 <type>Bronson</type>
 <location>Level 2 aft</location>
 <status>open</status>
 <maintenance responsible="M83" ls:supervisor="D59">
 <lastdone>2/23/2000</lastdone>
 <frequency>once per month</frequency>
 <ls:checkmethod>Negative pressure</ls:checkmethod>
 </maintenance>
 </airlock>
</airlocksystem>
```

- Bold elements are considered part of the airlocksystem namespace
- Attributes are considered not to have a namespace

# Self-Assessment

- What is the error in the following code? `<Title>Grilled Steak</title>`
- Name three limitations of HTML that led to the development of XML.
- What attribute would you add to a document's root element to declare a default namespace with the URI <http://ns.doc.book?>
- What is the main benefit of namespaces?
- Create an XML document that has two `<name>` elements with different meanings in it.
- Write an xml file for

BookID, Author, Title, Genre, Price and Publisher including Name and Year of Publication

# Summary



- Basic XML File Structure: root, elements, attributes
- Processing Instructions

```
<?xml-stylesheet href="airlocks.xsl" type="text/xsl"?>
```

- Comments

```
<!-- This is a Comment -->
```

- Rules for well-formed XML
- Namespaces

```
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0">
```



# Resources and References

- **Recommended Book Reading**

- Castledine, E. & Sharkie, C. (2012). *jQuery: Novice to Ninja*, 2nd Ed. SitePoint.
- Keith, J. & Sambells, J. (2010). *DOM Scripting: Web Design with JavaScript and the Document Object Model*, 2nd Ed. Springer.

- **Supplementary Book Resources**

- Fawcett, J., Ayers, D., & Quin, L. R. E. (2012). *Beginning XML*, 5th Ed. Wrox.
- Deitel, P. J. & Deitel, H. M. (2008). *AJAX, Rich Internet Applications, and Web Development for Programmers*. Prentice Hall
- Vodnik, S. & Gosselin, D. (2014). *JavaScript*, 6th Ed. Cengage Learning.
- Marc, W. (2016). *Learning Node.js: A Hands-On Guide to Building Web Applications in JavaScript*, 2nd Ed. Addison-Wesley.