# Parsing XML and JSON

M&Ms: Marshalling and Messaging
XML Document Structure and Parsing
JSON Overview
JSON Parsing
Comparing XML and JSON

# Goals of Data Exchange

Exchange data b/w applications w/ minimal effort

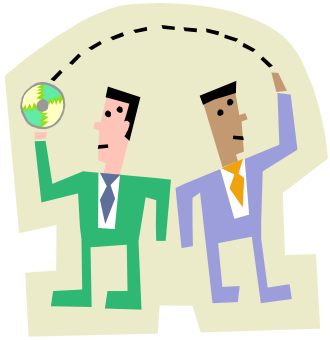
- Information must move from memory in one system to memory in another

Easily process data

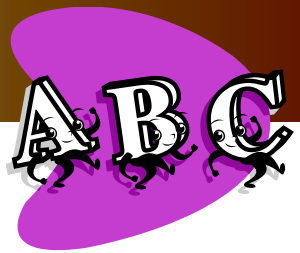
- Should be easily created and reconstructed
- Desirable to have libraries or infrastructure to translate between programming constructs and data format

Easily translate data to another form

- Allows creation of adapters for systems exchanging same information in different formats



# Two Common Forms




## Text

- Can be expensive as data is translated into textual form
  - String also need to be translated (Unicode to ASCII/UTF-8)
- Useful with heterogeneous technologies and systems
- Most common format is XML

## Binary

- Typically efficient as data can be sent w/out transform
- Primarily used when participating systems are of same technology
  - Custom, application-specific protocols
  - Technology standards - Java Serialization, .NET Remoting

# A Dr. Gary History Lesson

Back in my day…

- Developers passed data around in binary protocols

But then, on the web:

- Developers tired of passing binary (serialized) data
- Developers tired of tight coupling of business semantics to (changing) HTML markup
- Developers tired of the difficulties debugging binary data – wanted a “readable format”

Enter XML:

- Yay! No more data dictionaries, we have DTDs!
- Yay! No more binary data – we can read it!
- Yay! Decouple semantics from presentation!

# Extensible Markup Language (XML)

Tag-based language for describing data

- No fixed or even predefined tags (ala HTML)

Markup language for structured information

- Most info is structured (chapter/section, person/phone)
- Actually a meta-language for describing other languages
  - XML Schema (or DTD) defines grammar for particular language

De-facto standard for platform/language data exchange

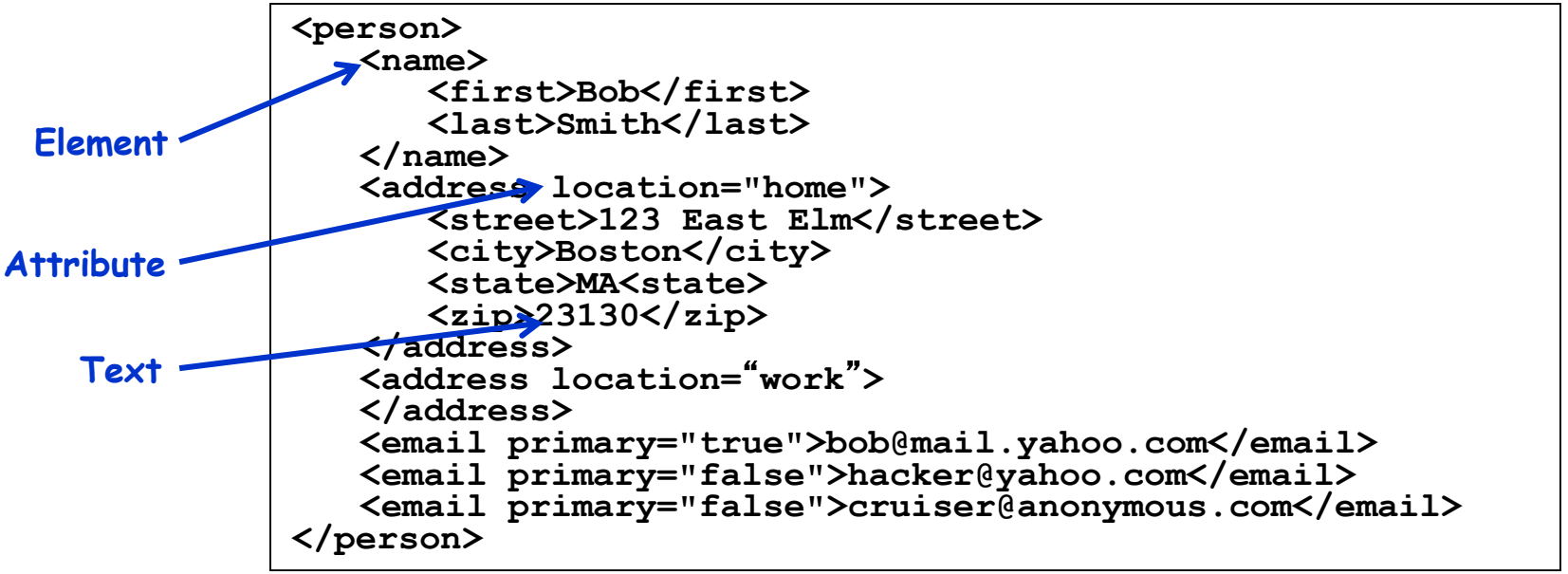
- Examples – SOAP, deployment descriptors, property files
- *Well, until JSON came along*

# Example XML Document

Contains a single, root element (person)

Composed of elements, attributes, and text

- Elements & attributes add meta-data to the textual data
- All data is in text form which may require conversions to/from binary



# What is a Valid XML Document?

*"Well-Formed"* documents have proper tag usage

- Tags must have begin *and* end tag
  Unlike the <p> tag in html (there is an XHTML)
- Tag pairs must be properly nested
  e.g., <fname>George<middle>W</fname></middle>

"Valid" documents are *well-formed* documents that validate against a grammar definition

- Document Type Definition (DTD)
- XML Schema Definition (XSD)

Document parsers have the option:

- Non-Validating – verify well-formed only
- Validating – verify well-formed & conforms to grammar

# Example DTD and XSD Grammars

**XML Grammars**

- DTDs
  - Document Type Declarations
  - Old school
- XSDs
  - XML Schema Declarations
  - Considered best practice

**XML Schemas**

- Pros
  - Has primitive types
  - Supports complex types
  - Support type restrictions
  - Are XML docs themselves
  - Basis for the Semantic Web
- Cons
  - More verbose
  - Lagging tool support

**DTD File**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT person (name, (address)+, (email)+)>
<!ELEMENT name (first, last)>
<!ELEMENT address (street, city, state, zip)>
<!ATTLIST addresslocation (home | work) #IMPLIED
<!ELEMENT first (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

**XSD File**

```
<xsd:element name="person">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="name"/>
       <xsd:element ref="address"
          minOccurs="1" maxOccurs="unbounded"/>
       <xsd:element ref="email"
          minOccurs="1" maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
 <xsd:element name="name">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="first"/>
       <xsd:element ref="last"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
```

# Parsing XML Documents

You *could* just read the XML doc in as a String…

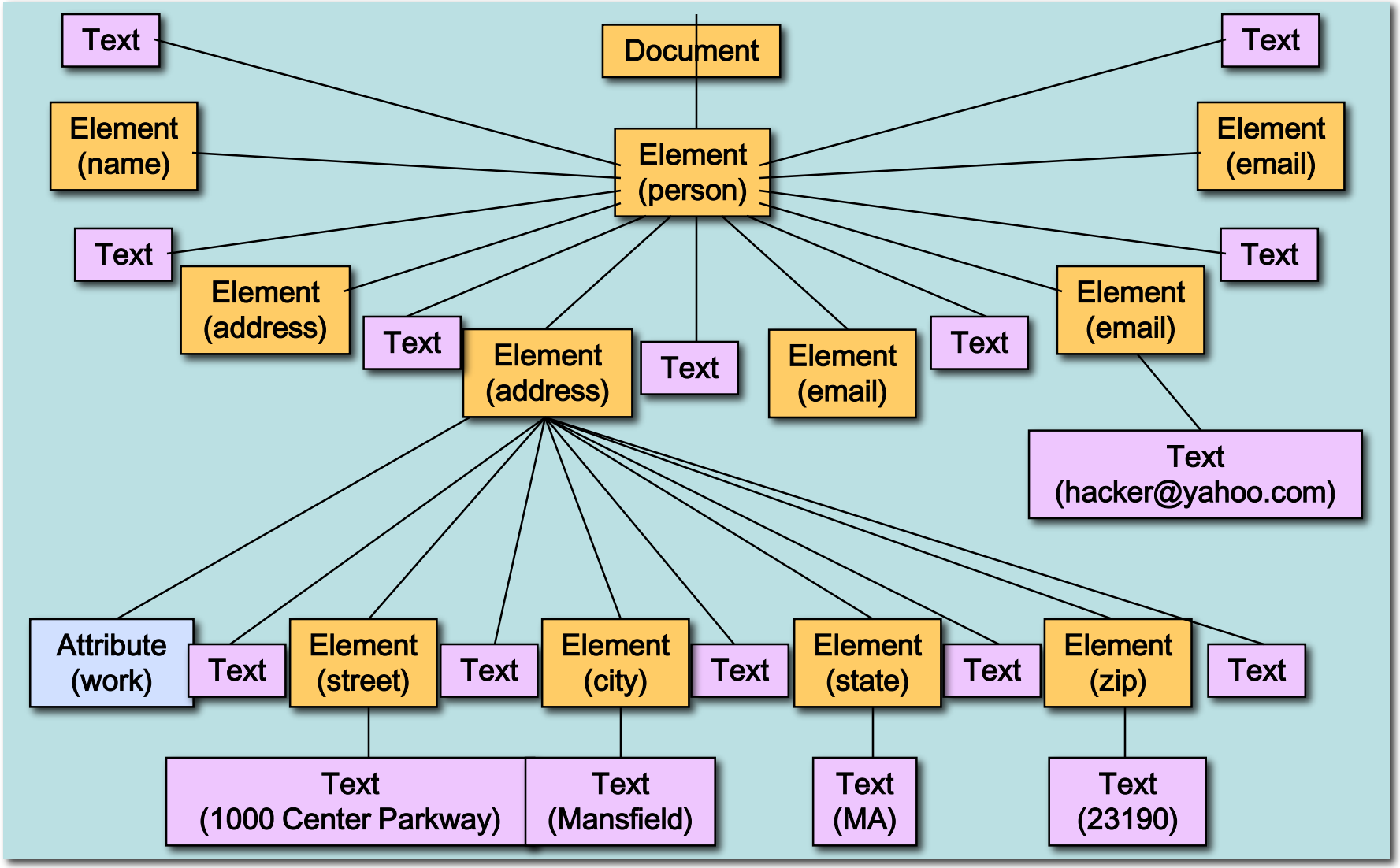
More accepted practices:

- DOM (Document Object Model)
  - Reads XML into a tree structure
  - Can use XPath to perform queries (stay tuned)
  - Resource intensive as entire tree is read into memory
- SAX (Simple API for XML)
  - Event-driven “push” - SAX notifies application through callbacks
  - Stateless - Developer must track information to construct output
- Browsers have a built-in XML parser for Javascript
  - See xml.js
  - Note this parser is not included in nodeJS, there you need to find a package, such as xml2js.

# DOM Tree

W3C defines interfaces for tree structure of *Nodes*

- Nodes subtypes: Document, Element, Text, Attribute



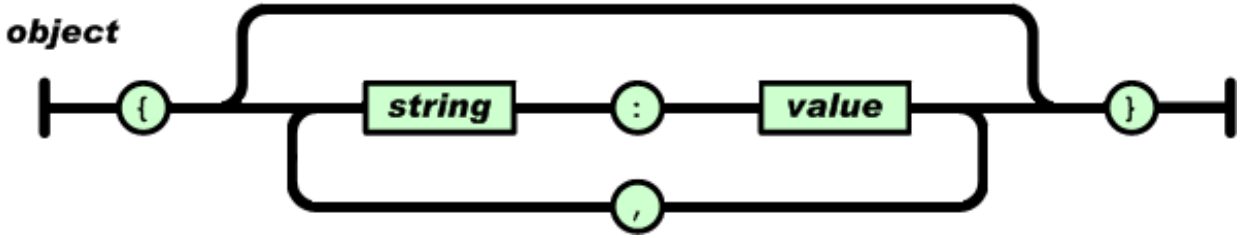
# JSON

## JavaScript Object Notation

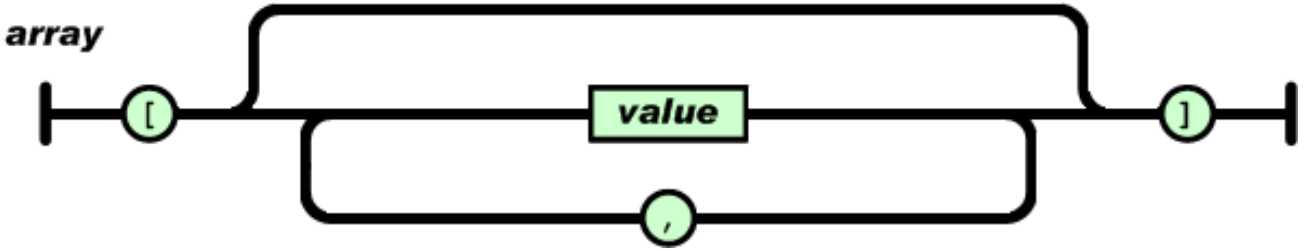
- Part of ECMA262 but distinct from Javascript
- Parsers/tools for a variety of popular languages

## Straight from json.org:

An ***object*** is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).

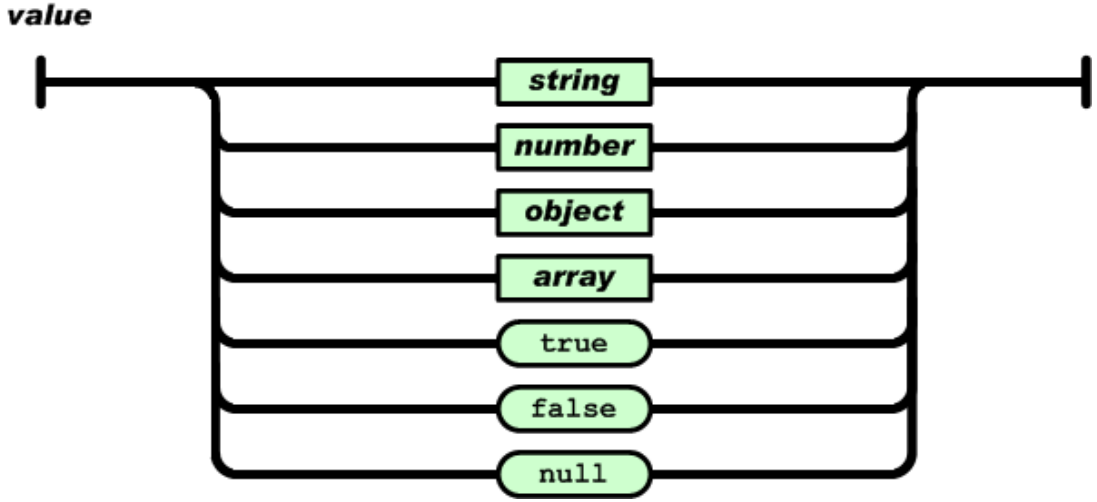



An ***array*** is an ordered collection of values. An array begins with [ (left bracket) and ends with ] (right bracket). Values are separated by , (comma).



# JSON (cont.)

A *value* can be a string, or a number, or true or false or null, or an object or an array. These structures can be nested.




*That's it!*

- Well there are definitions of strings/numbers for completeness
- Simple is Powerful
- Use a file extension of .json
- MIME type is application/json  ("application" – interesting)

# JSON Parsing

JSON can only be declared in a string

```
var person = '{"firstName":"Butch", "lastName":"Cassidy", \
"bestFriend":{"firstName":"Sundance","lastName":"Kid"}, "age":30}';

var personObj = { 'firstName': 'Butch', 'lastName': 'Cassidy',
            'bestFriend': {'firstName':'Sundance', 'lastName': 'Kid'},
            'greeting': function() { console.log("Hello " + this.firstName); }
          };

console.log(person.firstName);     // makes no sense, json string
console.log(personObj.firstName);  // makes perfect sense, object
console.log(personObj.greeting()); // you can have a function on it
```

- If we just used { } notation, it would be an object literal

JSON syntax constraints (compared to Javascript)

- Keys must be strings
- Values cannot be functions
- If you use the same key twice the result is undefined

# JSON Parsing

Lot of libraries & language bindings for JSON

- see json.org

Javascript:

- Built-in JSON library with various convenience methods:
  - `var personObj2 = JSON.parse(person);`
  - Now, you have an object so you can invoke personObj2.firstName;
    - Optional 2nd param called the *reviver* modifies the transformation of the string to object marshalling process
  - var personStr = JSON.stringify(personObj);
  - Converts an object to a JSON-compatible string

```
var foo = new Array();
foo[0] = 0;
foo[2] = 2;
console.log(JSON.stringify(foo));
[0,null,2]  // prints this out for console.log
```

# XML vs. JSON

## Why would you favor one over the other?

### XML Pros:

- Rich semantic markup useful for many applications
  - Semantic web, ontologies, rich data definitions
- Having a grammar is useful (sometimes)
  - You have types! You are not winging it!
  - Particularly helpful in server-to-server (transactional) scenarios

### JSON Pros:

- An example where lighter is better 80% of the time
  - Super-easy to parse/encode as it maps to programming structures
  - More efficient at run-time
  - Not a bandwidth or memory hog, better for web applications
  - Often one shop writes the client & server (well, not so much anymore?)
- (Foreshadow) as web applications move to JavaScript:
  - You only need an internal standard to marshal simple data
  - JavaScript + AJAX + JSON + REST == the new MVC