Extracting data from XML
Parsing - XML package

- 2 basic models - DOM & SAX

- Document Object Model (DOM)
  Tree stored internally as C, or as regular R objects

- Use XPath to query nodes of interest, extract info.

- Write recursive functions to "visit" nodes, extracting information as it descends tree

- Extract information to R data structures via handler functions that are called for particular XML elements by matching XML name

- For processing very large XML files with low-level state machine via R handler functions - closures.
Preferred Approach

- DOM (with internal C representation and XPath)

Given a node, several operations

- `xmlName()` - element name (w/w.o. namespace prefix)
- `xmlNamespace()`

- `xmlAttrs()` - all attributes
  - `xmlGetAttr()` - particular value

- `xmlValue()` - get text content.

- `xmlChildren()`, `node[[ i ]]`, `node [[ "el-name" ]]`
- `xmlSApply()`
- `xmlNamespaceDefinitions()`
Examples

- Scraping HTML - (you name it!)
- zillow - house price estimates
- PubMed articles/abstracts
- European Bank exchange rates
- itunes - CDs, tracks, play lists, ...
- PMML - predictive modeling markup language
- CIS - Current Index of Statistics/Google Scholar
- Google - Page Rank, Natural Language Processing
- Wikipedia - History of changes, ...
- SBML - Systems biology markup language
- Books - Docbook
- SOAP - eBay, KEGG, ...
- Yahoo Geo/places - given name, get most likely location
PubMed

- Professionally archived collection of "medically-related" articles.

- Vast collection of information, including
  - article abstracts
  - submission, acceptance and publication date
  - authors
  - ...

[Image 71x410 to 85x424]
[Image 71x327 to 85x340]
[Image 101x275 to 115x288]
[Image 101x223 to 115x236]
[Image 101x171 to 115x184]
[Image 101x119 to 115x132]
We'll use a sample PubMed example article for simplicity.
Can get very large, rich <ArticleSet> with many articles via an HTTP query done from within R/XML package directly.

Take a look at the data, see what is available or read the documentation
Or explore the contents.

rid=helppubmed.section.publisherhelp.XML_Tag_Descriptions
Example of a Standard XML file

Follow the links for more information about each tag.

```xml
<Articles>
  <Article>
    <Journal>
      <PublisherName>Nature Publishing Group</PublisherName>
      <JournalName>Nature Chemical Biology</JournalName>
      <ISSN>1551-4450</ISSN>
      <Volume>3</Volume>
      <Issue>1</Issue>
      <PageStart>1</PageStart>
      <PageEnd>1</PageEnd>
    </Journal>
    <ArticleTitle>High-content single-cell drug screening with phosphospecific flow cytometry</ArticleTitle>
    <FirstPage>1</FirstPage>
    <LastPage>1</LastPage>
  </Article>
  <Article>
    <Authors>
      <Author>
        <LastName>Deer</LastName>
        <MiddleName>W</MiddleName>
        <Affiliation>Department of Microbiology and Immunology, Baxter Laboratory in Genetic Pharmacology, Stanford University, 269 Campus Drive, Stanford, California 94305, USA.</Affiliation>
      </Author>
      <Author>
        <LastName>Johnson</LastName>
        <MiddleName>K</MiddleName>
        <collectiveName>Cancer Genome Project</collectiveName>
      </Author>
      <Author>
        <collectiveName>North American Barley Genome Project</collectiveName>
      </Author>
    </Authors>
  </Article>
</Articles>
```
doc = xmlTreeParse("pubmed.xml", useInternal = TRUE)

top = xmlRoot(doc)

xmlName(top)
[1] "ArticleSet"

names(top) - child nodes of this root
[1] "Article" "Article" - so 2 articles in this set.
Let's fetch the author list for each article. Do it first for just one and then use "apply" to iterate.

```haskell
names( top[[ 1 ]] )
```

```
<table>
<thead>
<tr>
<th>Journal</th>
<th>ArticleTitle</th>
<th>FirstPage</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Journal&quot;</td>
<td>&quot;ArticleTitle&quot;</td>
<td>&quot;FirstPage&quot;</td>
</tr>
<tr>
<td>LastPage</td>
<td>ELocationID</td>
<td>ELocationID</td>
</tr>
<tr>
<td>&quot;LastPage&quot;</td>
<td>&quot;ELocationID&quot;</td>
<td>&quot;ELocationID&quot;</td>
</tr>
<tr>
<td>Language</td>
<td>AuthorList</td>
<td>GroupList</td>
</tr>
<tr>
<td>&quot;Language&quot;</td>
<td>&quot;AuthorList&quot;</td>
<td>&quot;GroupList&quot;</td>
</tr>
<tr>
<td>ArticleIdList</td>
<td>History</td>
<td>&quot;History&quot;</td>
</tr>
<tr>
<td>&quot;ArticleIdList&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjectList</td>
<td></td>
<td>&quot;Abstract&quot;</td>
</tr>
<tr>
<td>&quot;ObjectList&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```haskell
art = top[[ 1 ]] [[ "AuthorList" ]]
what we want
```
So how do we get these values, e.g. to put in a dataframe.

Each element is a node with text content.
So loop over the nodes and get the content as a string

```r
xmlSApply(art[[1]], xmlValue)
```

To do this for all authors of the article

```r
xmlSApply(art, function(x) xmlSApply(x, xmlValue))
```

How do we deal with the different types of fields in the names?
- e.g. First, Middle, Last, Affiliation
- CollectiveName

data representation/analysis question from here.
In the <History> element, have date received, accepted, aheadofprint

May want to look at time publication lag (i.e. received to publication time) for different journals.

So get these dates for all the articles

<History>
    <PubDate PubStatus="received">
        <year>...</year><Month>06</Month><Day>15</Day>
    </PubDate>
    <PubDate>
    <PubDate PubStatus="accepted">
        <year>.....</day>
    </PubDate>
Find the element PubDate within History which has an attribute whose value is "received"

Can use `art["History"]["PubDate"]` to get all 3 elements.

But what if we want to access the 'received' dates for all the articles in a single operation, then the accepted, ...

Need a language to identify nodes with a particular characteristic/condition
XPath

XPath is a language for expressing such node subsetting with rich semantics for identifying nodes

- by name
- with specific attributes present
- with attributes with particular values
- with parents, ancestors, children

XPath = YALTL (Yet another language to learn)
XPath language

/node - top-level node

//node - node at any level

node[@attr-name] - node that has an attribute named "attr-name"

node[@attr-name='bob'] - node that has attribute named attr-name with value 'bob'

node/@x - value of attribute x in node with such attr.

Returns a collection of nodes, attributes, etc.
Let's find the date when the articles were received

nodes = getNodeSet(top,
    "//History/PubDate[@PubStatus='received']")

2 nodes - 1 per article

Extract year, month, day
lapply(nodes, function(x) xmlSApply(x, xmlValue))

Easy to get date "accepted" and "aheadofprint"
Text mining of abstract

Content of abstract as words

abstracts = xpathApply(top, "//Abstract", xmlValue)

Now, break up into words, stem the words, remove the stop-words,

abstractWords = lapply(abstracts, strsplit, "\[[[:space:]]\]")

library(Rstem)
abstractWords = lapply(abstractWords, function(x) wordStem[[1]])

Remove stop words
lapply(abstractWords, function(x) x[x %in% stopWords])
Zillow - house prices

Thanks to Roger, yesterday evening I found the Zillow XML API - (Application Programming Interface)

Can register with Zillow, make queries to find estimated house prices for a given house, comparables, demographics, ...

Put address, city-state-zip & Zillow login in URL request

Can put this at the end of a URL within xmlTreeParse()

"http://www.zillow.com/...../...?zws-id=...&address=1029%20Bob's%20Way&citstatezip=Berkeley"

But spaces are problematic, as are other characters.
So I use library(RCurl)

```r
reply = getForm("http://www.zillow.com/webservice/GetSearchResults.htm", 'zws-id' = "AB-XXXXXXXXXXX_10312q", address = "1093 Zuchini Way", citystatezip = "Berkeley, CA, 94212")
```

reply is text from the Web server containing XML
<request>
    <address>123 Bob's Way</address>
    <citystatezip>Berkeley, CA, 94217</citystatezip>
</request>

<message>
    <text>Request successfully processed</text>
    <code>0</code>
</message>

<response>
    <results>
        <result>
            <zpid>1111111</zpid>
            <links>
We want to get the value of the element
<amount>803000</amount>

doc =
 xmlTreeParse(reply, asText = TRUE, useInternal = TRUE)

xmlValue(doc[["//amount"]])
[1] "803000"

Other information too
2004 Election Results

http://www.princeton.edu/~rvdb/JAVA/election2004/
Where are the data?

Within days of the election?
USA Today, CNN, ...


By state, by county, by senate/house, ...
## Presidential vote by county - New Jersey

### Presidential Results - By County

<table>
<thead>
<tr>
<th>County</th>
<th>Total Precincts</th>
<th>Precincts Reporting</th>
<th>Bush</th>
<th>Kerry</th>
<th>Nader</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atlantic</td>
<td>159</td>
<td>158</td>
<td>48,197</td>
<td>52,181</td>
<td>535</td>
</tr>
<tr>
<td>Bergen</td>
<td>557</td>
<td>557</td>
<td>178,204</td>
<td>192,827</td>
<td>1,691</td>
</tr>
<tr>
<td>Burlington</td>
<td>359</td>
<td>359</td>
<td>90,112</td>
<td>103,971</td>
<td>945</td>
</tr>
<tr>
<td>Camden</td>
<td>331</td>
<td>331</td>
<td>76,925</td>
<td>82,918</td>
<td>894</td>
</tr>
<tr>
<td>Cape May</td>
<td>131</td>
<td>131</td>
<td>26,316</td>
<td>19,614</td>
<td>241</td>
</tr>
<tr>
<td>Cumberland</td>
<td>93</td>
<td>93</td>
<td>23,186</td>
<td>26,410</td>
<td>154</td>
</tr>
<tr>
<td>Essex</td>
<td>567</td>
<td>566</td>
<td>80,832</td>
<td>101,969</td>
<td>1,082</td>
</tr>
<tr>
<td>Gloucester</td>
<td>237</td>
<td>237</td>
<td>59,740</td>
<td>66,476</td>
<td>734</td>
</tr>
<tr>
<td>Hudson</td>
<td>452</td>
<td>451</td>
<td>55,530</td>
<td>113,603</td>
<td>831</td>
</tr>
<tr>
<td>Hunterdon</td>
<td>113</td>
<td>113</td>
<td>39,449</td>
<td>25,727</td>
<td>484</td>
</tr>
<tr>
<td>Mercer</td>
<td>265</td>
<td>264</td>
<td>53,469</td>
<td>85,662</td>
<td>771</td>
</tr>
<tr>
<td>Middlesex</td>
<td>597</td>
<td>597</td>
<td>119,438</td>
<td>166,168</td>
<td>1,701</td>
</tr>
<tr>
<td>Monmouth</td>
<td>437</td>
<td>437</td>
<td>181,853</td>
<td>131,808</td>
<td>2,005</td>
</tr>
<tr>
<td>Morris</td>
<td>395</td>
<td>395</td>
<td>126,761</td>
<td>90,478</td>
<td>1,154</td>
</tr>
<tr>
<td>Ocean</td>
<td>346</td>
<td>346</td>
<td>143,707</td>
<td>62,621</td>
<td>1,571</td>
</tr>
<tr>
<td>Passaic</td>
<td>288</td>
<td>288</td>
<td>73,586</td>
<td>91,939</td>
<td>833</td>
</tr>
<tr>
<td>Salem</td>
<td>45</td>
<td>45</td>
<td>15,835</td>
<td>13,650</td>
<td>165</td>
</tr>
<tr>
<td>Somerset</td>
<td>307</td>
<td>307</td>
<td>67,505</td>
<td>61,550</td>
<td>739</td>
</tr>
<tr>
<td>Sussex</td>
<td>107</td>
<td>106</td>
<td>42,085</td>
<td>22,282</td>
<td>480</td>
</tr>
<tr>
<td>Union</td>
<td>112</td>
<td>112</td>
<td>29,205</td>
<td>112,542</td>
<td>968</td>
</tr>
<tr>
<td>Warren</td>
<td>87</td>
<td>87</td>
<td>20,333</td>
<td>17,876</td>
<td>454</td>
</tr>
</tbody>
</table>

*Last updated: 11/1/2004 3:31 PM EST*
read.table ?

Within the noise/ads, look for a table whose first cell is "County"

Actually a
   <td><b>County</b></td>

How do we know this? Look at one or two HTML files out of the 50. Verify the rest.

Then, given the associated <table> element, we can extract the values row by row and get a data.frame/....
XPath expression

```xml
<table>
  <tr>
    <td class="notch_medium" width="153"><b>County</b></td>
    <td class="notch_medium" align="Right" width="65"><b>Total Precincts</b></td>
    <td class="notch_medium" align="Right" width="70"><b>Precincts Reporting</b></td>
    <td class="notch_medium" align="Right" width="60"><b>Bush</b></td>
    <td class="notch_medium" align="Right" width="60"><b>Kerry</b></td>
    <td class="notch_medium" align="Right" width="60"><b>Nader</b></td>
  </tr>
</table>
```

Little bit of trial and error

- `getNodeSet(nj, "//table[tr/td/b/text()='Total Precincts']")`
  
Could be more specific, e.g. `tr[1]` - first row
Now that we have the `<table>` node, read the data into an R data structure

```r
rows = xmlApply(v[[1]],
               function(x)
               xmlSApply(x, xmlValue))
```

i.e. for each row, loop over the `<td>` and get its value.

Got some "\n\t\t\t" and last row is "Updated...."
first row is the County, Total Precincts, ....

So discard the rows without 7 entries
then remove the 7th entry ("\n\t\t\t")
v = getNodeSet(nj, "//table[tr/td/b/text()='Total Precincts']")
rows = xmlApply(v[[1]], function(x) xmlSApply(x, xmlValue))

# only the rows with 7 elements
rows = rows[sapply(rows, length) == 7]
# Remove the 7th element, and transpose to put back into
# counties as rows, precinct, candidates, ... as columns.
# So get a matrix of counties by 6 matrix of character
# vectors.
rows = t(sapply(rows, "[", -7))
Learning XPath

- XPath is another language
- part of the XML technologies
  - XInclude
  - XPointer
  - XSL
  - XQuery
- Can't we extract the data from the XML tree/DOM (Document Object Model) without it and just use R programming – Yes
```r
doc = xmlTreeParse("pubmed.xml")

Now have a tree in R

- recursive - list of children which are lists of children
- or recursive tree of C-level nodes

Write an R function which "visits" each node and extracts and stores the data from those nodes that are relevant

- e.g. the <Author>, <PubDate> nodes
Recursive functions are sometimes difficult to write

- Have to store the results "globally"/non-locally leads to closures/lexical scoping – "advanced R"
- Have to traverse the entire tree via R code – SLOW!
Handlers

Alternative approach

- when we read the XML tree into R and convert it to a list of lists of children...

- when convert each C-level node, see if caller has a function registered corresponding to the name/type of node

- if so call it and allow it to extract and store the data.
Efficient Parsing

- Problem with previous styles is we have the entire tree in memory and then extract the data => 2 times the data in memory at the end

- Bad news for large datasets

  All of Wikipedia pages - 11Gigabytes

- Need to read the XML as it passes as a stream, extracting and storing the contents and discarding the XML.

- SAX parsing - "Simple API for XML"!
xmlEventParse(content, 
    list(startElement = function(node, ...) ...., 
         endElement = function(node, ...) ... , 
         text = function(x) ... , 
         comment = function(x) ... , ....)))

Whenever XML parser sees start/end/text/comment node, calls R function which maintains state.

Awkward to write, but there to handle very large data.
Just like a database has a schema describing the characteristics of columns in all tables within a database, XML documents often have an XML Schema (or Document Type Definition - DTD) describing the "template" tree and what elements can/must go where, attributes, etc.

The XML Schema is written in XML, so we can read it!

And we can actually create R data types to represent the same elements in XML directly in R.

So we can automate some of the reading of XML elements into useful, meaning R objects harder to programmatically flatten into data frames.
**RCurl**

- `xmlTreeParse()` & `xmlEventParse()` can read from files, compressed files, URLs, direct text - but limited connection support.

- RCurl package provides very rich ways that extend R's ability to access content from URLs, etc. over the Internet.

- HTTPS - encrypted/secure HTTP passwords/authentication efficient, persistent connections multiplexing different protocols

- Pass results to XML parser or other consumers.
Exceptions/Conditions