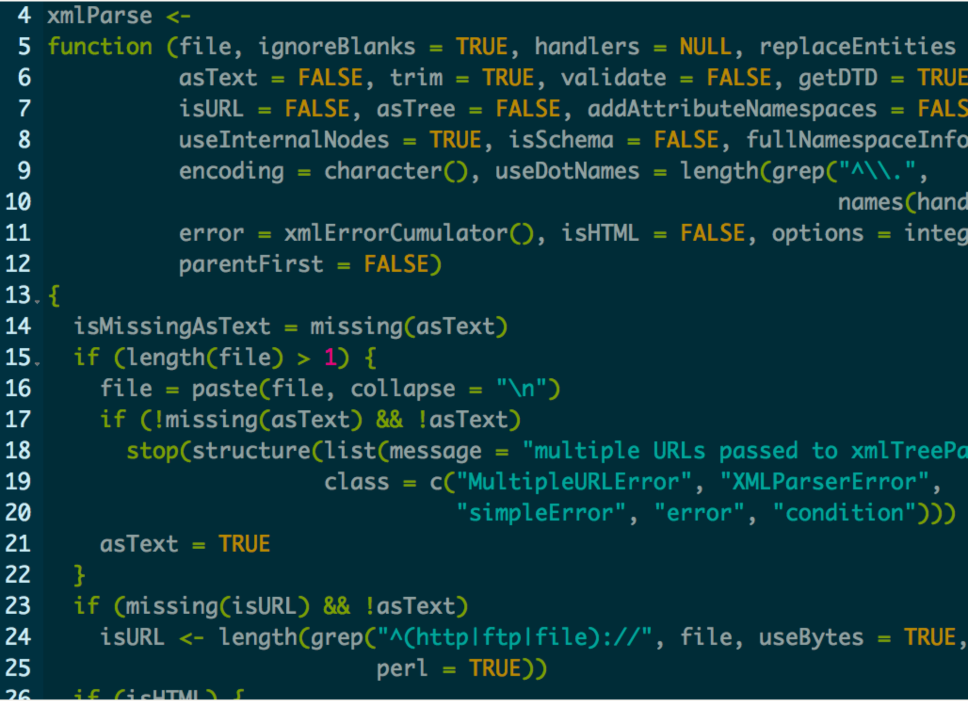
Getting Data from the Web with R

Part 4: Parsing XML/HTML Content

Gaston **S**anchez

April-May 2014



```
xmlParse <-
function (file, ignoreBlanks = TRUE, handlers = NULL, replaceEntities
          asText = FALSE, trim = TRUE, validate = FALSE, getDTD = TRUE
          isURL = FALSE, asTree = FALSE, addAttributeNamespaces = FALS
          useInternalNodes = TRUE, isSchema = FALSE, fullNamespaceInfo
          encoding = character(), useDotNames = length(grep("^\\.",
                                                            names(hand
          error = xmlErrorCumulator(), isHTML = FALSE, options = integ
          parentFirst = FALSE)
{
  isMissingAsText = missing(asText)
  if (length(file) > 1) {
    file = paste(file, collapse = "\n")
    if (!missing(asText) && !asText)
      stop(structure(list(message = "multiple URLs passed to xmlTreePa
                     class = c("MultipleURLError", "XMLParserError",
                               "simpleError", "error", "condition")))
    asText = TRUE
  }
  if (missing(isURL) && !asText)
    isURL <- length(grep("^(http|ftp|file)://", file, useBytes = TRUE,
                         perl = TRUE))
  if (isHTML) {
```

Readme

License:

You are free to:

Under the following conditions:

Lectures Menu

Slide Decks

1. Introduction
2. Reading files from the Web
3. Basics of XML and HTML
4. **Parsing XML / HTML content**
5. Handling JSON data
6. HTTP Basics and the RCurl Package
7. Getting data via web forms
8. Getting data via web services

Parsing XML and HTML Content

Goal

Parsing XML / HTML docs

The goal of these slides is to describe **how we can parse XML / HTML content** with the R package `"XML"`

Synopsis

In a nutshell

We'll cover a variety of situations you most likely will find yourself dealing with:

- R package XML
- Navigating the xml tree structure
- Main functions in package XML
- XPath

Some References

- An Introduction to the XML Package for R
  http://www.omegahat.org/RSXML/Tour.pdf
- A Short Introduction to the XML package for R
  http://www.omegahat.org/RSXML/shortIntro.pdf
- R and Splus XML Parsers
  http://www.omegahat.org/RSXML/Overview.html
- XML and Web Technlogies for Data Sciences with R
  by Deb Nolan and Duncan Temple Lang

Parsing

"A parser is a software component that takes input data (frequently text) and builds a data structure —often some kind of parse tree, abstract syntax tree or other hierarchical structure— giving a structural representation of the input, checking for correct syntax in the process"

`http://en.wikipedia.org/wiki/Parsing#Parser`

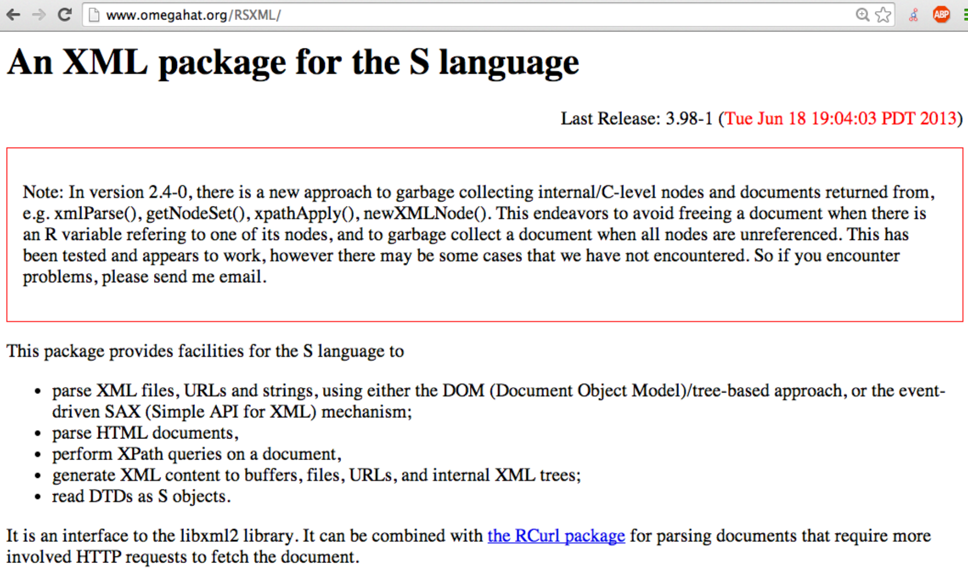
Parsing XML and HTML Content

Parsing XML and HTML?

Getting data from the web often involves reading and processing content from xml and html documents. This is known as parsing.

Luckily for us there's the R package `"XML"` (by Duncan Temple Lang) that allows us to parse such types of documents.

R package "XML"



An XML package for the S language

Last Release: 3.98-1 (Tue Jun 18 19:04:03 PDT 2013)

Note: In version 2.4-0, there is a new approach to garbage collecting internal/C-level nodes and documents returned from, e.g. xmlParse(), getNodeSet(), xpathApply(), newXMLNode(). This endeavors to avoid freeing a document when there is an R variable refering to one of its nodes, and to garbage collect a document when all nodes are unreferenced. This has been tested and appears to work, however there may be some cases that we have not encountered. So if you encounter problems, please send me email.

This package provides facilities for the S language to

- parse XML files, URLs and strings, using either the DOM (Document Object Model)/tree-based approach, or the event-driven SAX (Simple API for XML) mechanism;
- parse HTML documents,
- perform XPath queries on a document,
- generate XML content to buffers, files, URLs, and internal XML trees;
- read DTDs as S objects.

It is an interface to the libxml2 library. It can be combined with the RCurl package for parsing documents that require more involved HTTP requests to fetch the document.

R Package XML

The package `"XML"` is designed for 2 major purposes

1. parsing xml / html content
2. writing xml / html content

We won't cover the functions and utilities that have to do with writing xml / html content

What can we do with "XML"?

We'll cover 4 major types of tasks that we can perform with `"XML"`

1. parsing (ie *reading*) xml / html content
2. obtaining descriptive information about parsed contents
3. navigating the tree structure (ie accessing its components)
4. querying and extracting data from parsed contents

Using "XML"

If you don't have "XML" you'll need to install it first

```
# installing xml
install.packages("xml", dependencies = TRUE)
```

Once installed it can be loaded

```
# load XML
library(XML)
```

More info about "XML" at:

http://www.omegahat.org/RSXML

Parsing Functions

Function xmlParse()

Function xmlParse()

- "XML" comes with the *almighty* parser function xmlParse()
- the main input for xmlParse() is a file: either a local file, a complete URL or a text string

  ex1: `xmlParse("Documents/file.xml")`

  ex2: `xmlParse("http://www.xyz.com/some_file.xml")`

  ex3: `xmlParse(xml_string, asText=TRUE)`
- the rest of the 20+ parameters are optional, and provide options to control the parsing procedure

`xmlParse()` default behavior

What does `xmlParse()` do?

First let's talk about the **default behavior** of `xmlParse()`

- it is a DOM parser: it reads an XML document into a hierarchical structure representation
- it builds an XML tree as a native C-level data structure (not an R data structure)
- it returns an object of class `"XMLInternalDocument"`
- can read content from compressed files without us needing to explicitly uncompress the file
- it does NOT handle `HTTPS` (secured HTTP)

About xmlParse() (con't)

Default behavior

Simple usage of xmlParse() on an XML document:

```
# parsing an xml document
doc1 = xmlParse("http://www.xmlfiles.com/examples/plant_catalog.xml")
```

xmlParse() returns an object of class "XMLInternalDocument" which is a C-level internal data structure

```
# class
class(doc1)


## [1] "XMLInternalDocument" "XMLAbstractDocument"
```

About xmlParse() (con't)

Argument useInternalNodes = FALSE

Instead of parsing content as an internal C-level structure, we can parse it into an R structure by specifying the parameter useInternalNodes = FALSE

```
# parsing an xml document into an R structure
doc2 = xmlParse("http://www.xmlfiles.com/examples/plant_catalog.xml",
                useInternalNodes = FALSE)
```

the output is of class "XMLDocument" and is implemented as a hierarchy of lists

```
# class
class(doc2)


## [1] "XMLDocument"         "XMLAbstractDocument"


is.list(doc2)


## [1] TRUE
```

About xmlTreeParse()

Argument useInternalNodes = FALSE

"XML" provides the function xmlTreeParse() as a convenient synonym for xmlParse(file, useInternalNodes = FALSE)

```
# parse an xml document into an R structure
doc3 = xmlTreeParse("http://www.xmlfiles.com/examples/plant_catalog.xml")
```

As expected, the output is of class "XMLDocument"

```
# class
class(doc3)

## [1] "XMLDocument"         "XMLAbstractDocument"

identical(doc2, doc3)

## [1] TRUE
```

HTML Content

Parsing HTML content

In theory, we could use `xmlParse()` with its default settings to parse HTML documents.

However `xmlParse()` —with its default behavior— will not work properly when HTML documents are not well-formed:

- no xml declaration
- no DOCTYPE
- no closure of tags

xmlParse() and HTML Content

Argument isHTML = TRUE

One option to parse HTML documents is by using xmlParse() with the argument isHTML = TRUE

```
# parsing an html document with 'xmlParse()'
doc4 = xmlParse("http://www.r-project.org/mail.html", isHTML = TRUE)
```

the output is of class "HTMLInternalDocument"

```
# class
class(doc4)


## [1] "HTMLInternalDocument" "HTMLInternalDocument" "XMLInternalDocument"
## [4] "XMLAbstractDocument"
```

htmlParse() and HTML Content

Function htmlParse()

Another option is to use the function htmlParse() which is equivalent to xmlParse(file, isHTML = TRUE)

```
# parsing an html document with 'htmlParse()'
doc5 = htmlParse("http://www.r-project.org/mail.html")
```

again, the output is of class "HTMLInternalDocument"

```
# class
class(doc5)


## [1] "HTMLInternalDocument" "HTMLInternalDocument" "XMLInternalDocument"
## [4] "XMLAbstractDocument"
```

Function `htmlTreeParse()`

Function `htmlTreeParse()`

To parse content into an R structure we have to use `htmlTreeParse()` which is equivalent to `htmlParse(file, useInternalNodes = FALSE)`

```
# parsing an html document into an R structure
doc6 = htmlTreeParse("http://www.r-project.org/mail.html")
```

in this case the output is of class `"XMLDocumentContent"`

```
# class
class(doc6)


## [1] "XMLDocumentContent"
```

HTML Content

About parsing HTML documents

- xmlParse() can do the job but only on well-formed HTML
- it is better to be conservative and use the argument isHTML = TRUE, which is equivalent to using htmlParse()
- we can use htmlParse() or htmlTreeParse() which try to correct not well-formed docs by using heuristics that will take care of the missing elements
- in a worst-case scenario we can use tidyHTML() from the R package "RTidyHTML", and then pass the result to htmlParse()

Parsing Functions Summary

`xmlParse(file)`

- main parsing function
- returns class `"XMLInternalDocument"` (C-level structure)

`xmlTreeParse(file)`

- returns class `"XMLDocument"` (R data structure)
- equivalent to `xmlParse(file, useInternalNodes = FALSE)`

Parsing Functions Summary

`htmlParse(file)`

- especially suited for parsing HTML content
- returns class `"HTMLInternalDocument"` (C-level structure)
- equivalent to `xmlParse(file, isHTML = TRUE)`

`htmlTreeParse(file)`

- especially suited for parsing HTML content
- returns class `"XMLDocumentContent"` (R data structure)
- equivalent to
  - `xmlParse(file, isHTML = TRUE, useInternalNodes = FALSE)`
  - `htmlParse(file, useInternalNodes = FALSE)`

Working with Parsed Documents

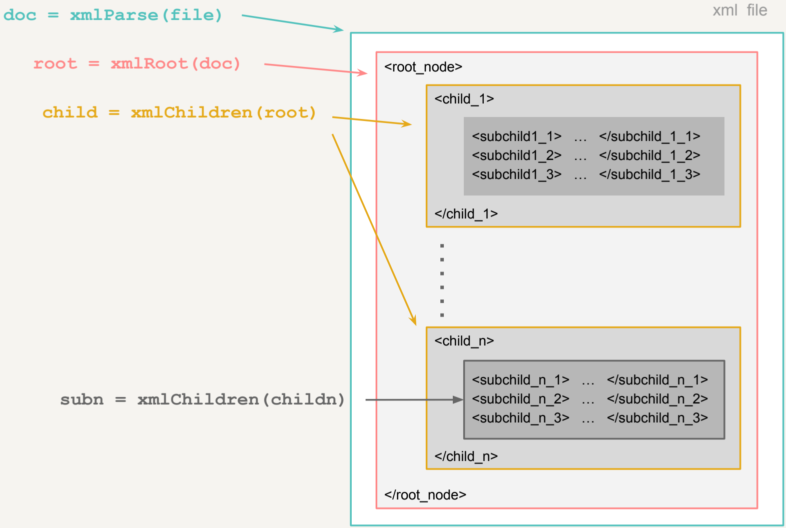
Parsed Documents

`xmlRoot()` and `xmlChildren()`

Having parsed an XML / HTML document, we can use 2 main functions to start working on the tree structure:

- `xmlRoot()` gets access to the root node and its elements
- `xmlChildren()` gets access to the child elements of a given node

Conceptual Diagram



```
doc = xmlParse(file)

root = xmlRoot(doc)

child = xmlChildren(root)

subn = xmlChildren(childn)
```

xml file

```
<root_node>
  <child_1>
    <subchild1_1> … </subchild_1_1>
    <subchild1_2> … </subchild_1_2>
    <subchild1_3> … </subchild_1_3>
  </child_1>
  ⋮
  <child_n>
    <subchild_n_1> … </subchild_n_1>
    <subchild_n_2> … </subchild_n_2>
    <subchild_n_3> … </subchild_n_3>
  </child_n>
</root_node>
```

Some Additional Functions

Functions for a given node

Function	Description
`xmlName()`	name of the node
`xmlSize()`	number of subnodes
`xmlAttrs()`	named character vector of all attributes
`xmlGetAttr()`	value of a single attribute
`xmlValue()`	contents of a leaf node
`xmlParent()`	name of parent node
`xmlAncestors()`	name of ancestor nodes
`getSibling()`	siblings to the right or to the left
`xmlNamespace()`	the namespace (if there's one)

The applicability of the functions depends on the class of objects we are working on

Toy Example: Movies XML

```r
# define some xml content
xml_string = c(
  '<?xml version="1.0" encoding="UTF-8"?>',
  '<movies>',
  '<movie mins="126" lang="eng">',
  '<title>Good Will Hunting</title>',
  '<director>',
  '<first_name>Gus</first_name>',
  '<last_name>Van Sant</last_name>',
  '</director>',
  '<year>1998</year>',
  '<genre>drama</genre>',
  '</movie>',
  '<movie mins="106" lang="spa">',
  '<title>Y tu mama tambien</title>',
  '<director>',
  '<first_name>Alfonso</first_name>',
  '<last_name>Cuaron</last_name>',
  '</director>',
  '<year>2001</year>',
  '<genre>drama</genre>',
  '</movie>',
  '</movies>')

# parse xml content
movies_xml = xmlParse(xml_string, asText = TRUE)
```

```r
# check movies_xml
movies_xml
```

```
## <?xml version="1.0" encoding="UTF-8"?>
## <movies>
##   <movie mins="126" lang="eng">
##     <title>Good Will Hunting</title>
##     <director>
##       <first_name>Gus</first_name>
##       <last_name>Van Sant</last_name>
##     </director>
##     <year>1998</year>
##     <genre>drama</genre>
##   </movie>
##   <movie mins="106" lang="spa">
##     <title>Y tu mama tambien</title>
##     <director>
##       <first_name>Alfonso</first_name>
##       <last_name>Cuaron</last_name>
##     </director>
##     <year>2001</year>
##     <genre>drama</genre>
##   </movie>
## </movies>
##
```

Movies XML: Root Node

```
# examine class
# (movies_xml is a C-level object)
class(movies_xml)


## [1] "XMLInternalDocument" "XMLAbstractDocument"


# get root node
root = xmlRoot(movies_xml)

# examine class
class(root)


## [1] "XMLInternalElementNode" "XMLInternalNode"
```

```
# display root node
root


## <movies>
##   <movie mins="126" lang="eng">
##     <title>Good Will Hunting</title>
##     <director>
##       <first_name>Gus</first_name>
##       <last_name>Van Sant</last_name>
##     </director>
##     <year>1998</year>
##     <genre>drama</genre>
##   </movie>
##   <movie mins="106" lang="spa">
##     <title>Y tu mama tambien</title>
##     <director>
##       <first_name>Alfonso</first_name>
##       <last_name>Cuaron</last_name>
##     </director>
##     <year>2001</year>
##     <genre>drama</genre>
##   </movie>
## </movies>
```

Movies XML: movie children

```
# children of root node
movie_child = xmlChildren(root)

movie_child
```

```
## $movie
## <movie mins="126" lang="eng">
##   <title>Good Will Hunting</title>
##   <director>
##     <first_name>Gus</first_name>
##     <last_name>Van Sant</last_name>
##   </director>
##   <year>1998</year>
##   <genre>drama</genre>
## </movie>
##
## $movie
## <movie mins="106" lang="spa">
##   <title>Y tu mama tambien</title>
##   <director>
##     <first_name>Alfonso</first_name>
##     <last_name>Cuaron</last_name>
##   </director>
##   <year>2001</year>
##   <genre>drama</genre>
## </movie>
##
## attr(,"class")
## [1] "XMLInternalNodeList" "XMLNodeList"
```

```
# first movie
goodwill = movie_child[[1]]
goodwill
```

```
## <movie mins="126" lang="eng">
##   <title>Good Will Hunting</title>
##   <director>
##     <first_name>Gus</first_name>
##     <last_name>Van Sant</last_name>
##   </director>
##   <year>1998</year>
##   <genre>drama</genre>
## </movie>
```

```
# second movie
tumama = movie_child[[2]]
tumama
```

```
## <movie mins="106" lang="spa">
##   <title>Y tu mama tambien</title>
##   <director>
##     <first_name>Alfonso</first_name>
##     <last_name>Cuaron</last_name>
##   </director>
##   <year>2001</year>
##   <genre>drama</genre>
## </movie>
```

Movies XML: movie children

```
# node name
xmlName(goodwill)

## [1] "movie"

# number of children
xmlSize(goodwill)

## [1] 4

# node attributes
xmlAttrs(goodwill)

##  mins  lang
## "126" "eng"

# get specific attribute value
xmlGetAttr(goodwill, name = 'lang')

## [1] "eng"
```

```
# node name
xmlName(tumama)

## [1] "movie"

# number of children
xmlSize(tumama)

## [1] 4

# node attributes
xmlAttrs(tumama)

##  mins  lang
## "106" "spa"

# get specific attribute value
xmlGetAttr(tumama, name = "lang")

## [1] "spa"
```

Movies XML: movie Good Will Hunting

```
# node content (as character string)
xmlValue(goodwill)


## [1] "Good Will HuntingGusVan Sant1998drama"


# child nodes of goodwill node
xmlChildren(goodwill)


## $title
## <title>Good Will Hunting</title>
##
## $director
## <director>
##   <first_name>Gus</first_name>
##   <last_name>Van Sant</last_name>
## </director>
##
## $year
## <year>1998</year>
##
## $genre
## <genre>drama</genre>
##
## attr(,"class")
## [1] "XMLInternalNodeList" "XMLNodeList"
```

```
# director nodes of goodwill node
gusvan = xmlChildren(goodwill)[[2]]
gusvan


## <director>
##   <first_name>Gus</first_name>
##   <last_name>Van Sant</last_name>
## </director>


# parent
xmlParent(gusvan)


## <movie mins="126" lang="eng">
##   <title>Good Will Hunting</title>
##   <director>
##     <first_name>Gus</first_name>
##     <last_name>Van Sant</last_name>
##   </director>
##   <year>1998</year>
##   <genre>drama</genre>
## </movie>
```

Movies XML: movie Good Will Hunting

```r
# director children
xmlChildren(gusvan)
```

```
## $first_name
## <first_name>Gus</first_name>
##
## $last_name
## <last_name>Van Sant</last_name>
##
## attr(,"class")
## [1] "XMLInternalNodeList" "XMLNodeList"
```

```r
# sibling of goodwill node
getSibling(goodwill)
```

```
## <movie mins="106" lang="spa">
##   <title>Y tu mama tambien</title>
##   <director>
##     <first_name>Alfonso</first_name>
##     <last_name>Cuaron</last_name>
##   </director>
##   <year>2001</year>
##   <genre>drama</genre>
## </movie>
```

Looping Over Nodes

Looping Over Nodes

Looping over nodes

Extracting data from an XML / HTML document involves applying a given function to a subset of nodes. This means iterating over such subset.

There are various ways to loop over a subset of nodes:

- the most basic approach is with `sapply()` or `lapply()`
- anoter way is by using the ad-hoc functions `xmlApply()` and `xmlSApply()`, which are simple wrappers for the `lapply()` and `sapply()` functions.

Looping Over Nodes

Some iteration examples with sapply()

```
# length
sapply(movie_child, length)
```

```
## movie movie
##     1     1
```

```
# names in child nodes
sapply(movie_child, names)
```

```
##          movie      movie
## title    "title"    "title"
## director "director" "director"
## year     "year"     "year"
## genre    "genre"    "genre"
```

```
sapply(movie_child, xmlSize)
```

```
## movie movie
##     4     4
```

```
# attributes of root child nodes
sapply(movie_child, xmlAttrs)
```

```
##      movie movie
## mins "126" "106"
## lang "eng" "spa"
```

```
# names in child nodes
sapply(movie_child, xmlValue)
```

```
##                                      movie
##   "Good Will HuntingGusVan Sant1998drama"
##                                      movie
## "Y tu mama tambienAlfonsoCuaron2001drama"
```

Looping Over Nodes

`xmlApply()` and `xmlSApply()` operate on the sub-nodes of the XML node:

```
# names in child nodes
xmlSApply(root, names)
```

```
##          movie      movie
## title    "title"    "title"
## director "director" "director"
## year     "year"     "year"
## genre    "genre"    "genre"
```

```
# size of movie children
xmlSApply(root, xmlSize)
```

```
## movie movie
##     4     4
```

```
# attributes of root child nodes
xmlSApply(root, xmlAttrs)
```

```
##      movie movie
## mins "126" "106"
## lang "eng" "spa"
```

```
# names in child nodes
xmlSApply(root, xmlValue)
```

```
##                                     movie
##   "Good Will HuntingGusVan Sant1998drama"
##                                     movie
## "Y tu mama tambienAlfonsoCuaron2001drama"
```

Looping Over Nodes

```
# length of nodes in movie 1
xmlSApply(root[[1]], length)

##    title director     year    genre
##        1        1        1        1

# size in child nodes in movie 1
xmlSApply(root[[1]], xmlSize)

##    title director     year    genre
##        1        2        1        1

# attribute values of nodes in movie 1
xmlSApply(root[[1]], xmlValue)

##               title            director
## "Good Will Hunting"       "GusVan Sant"
##               genre
##             "drama"
```

```
# length of nodes in movie 2
xmlSApply(root[[2]], length)

##    title director     year    genre
##        1        1        1        1

# size in child nodes in movie 2
xmlSApply(root[[2]], xmlSize)

##    title director     year    genre
##        1        2        1        1

# attribute values of nodes in movie 2
xmlSApply(root[[2]], xmlValue)

##               title            director
## "Y tu mama tambien"     "AlfonsoCuaron"
##               genre
##             "drama"
```

XPath Language

XPath

Querying Trees

The real parsing power comes from the ability to **locate nodes and extract information from them**. For this, we need to be able to perform queries on the parsed content.

XPath

The solution is provided by **XPath**, which is a language to navigate through elements and attributes in an XML/HTML document

XPath

XPath

- is a language for finding information in an XML document
- uses path expressions to select nodes or node-sets in an XML document
- works by identifying patterns to match data or content
- includes over 100 built-in functions

About XPath

XPath Syntax

XPath uses **path expressions** to select nodes in an XML document. It has a computational model to identify sets of nodes (node-sets)

XPath Syntax

We can specify paths through the tree structure:

- based on node names
- based on node content
- based on a node's relationship to other nodes

About XPath

XPath Syntax

The key concept is knowing how to write XPath expressions. XPath expressions have a syntax similar to the way files are located in a hierarchy of directories/folders in a computer file system. For instance:

```
/movies/movie[1]
```

is the XPath expression to locate the first `movie` element that is the child of the `movies` element

Selecting Nodes

XPath Syntax

The main path expressions (ie symbols) are:

Symbol	Description
`/`	selects from the root node
`//`	selects nodes anywhere
`.`	selects the current node
`..`	Selects the parent of the current node
`@`	Selects attributes
`[]`	Square brackets to indicate attributes

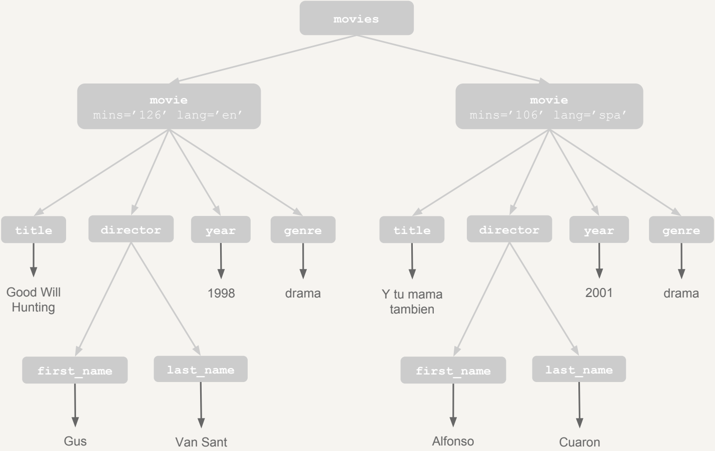
Selecting Unknown Nodes

XPath wildcards for unknown nodes

XPath wildcards can be used to select unknown XML elements

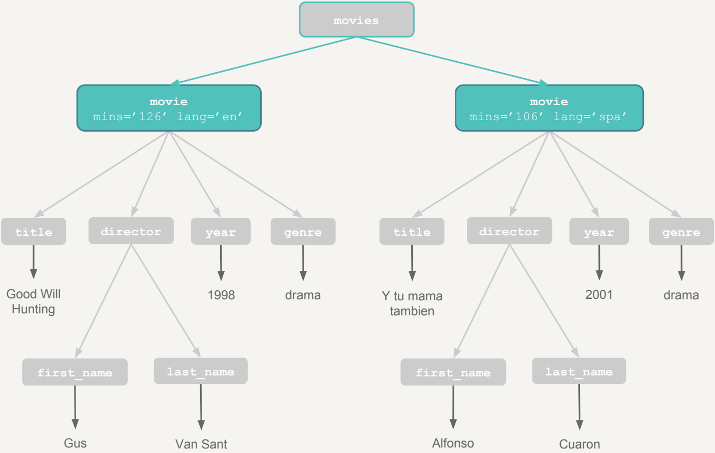
Symbol	Description
`*`	matches any element node
`@*`	matches any attribute node
`node()`	matches any node of any kind

Movies Tree Structure



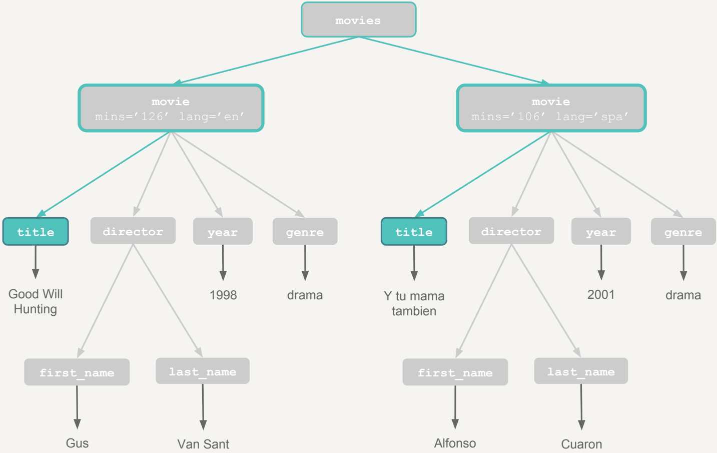
XPath: movie nodes

/movies/movie



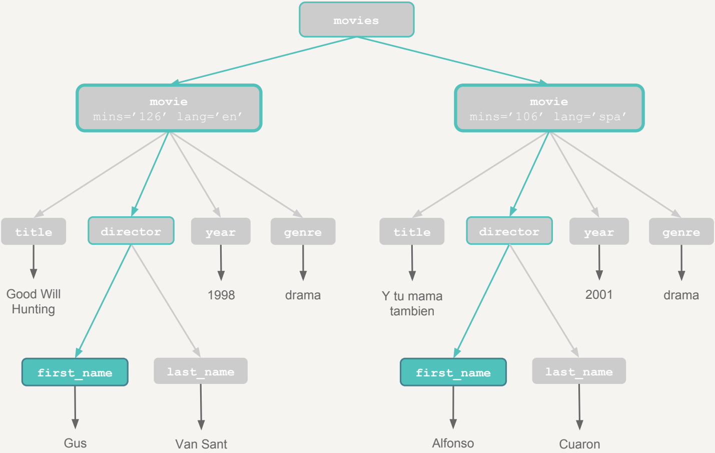
XPath: movie title nodes

/movies/movie/title



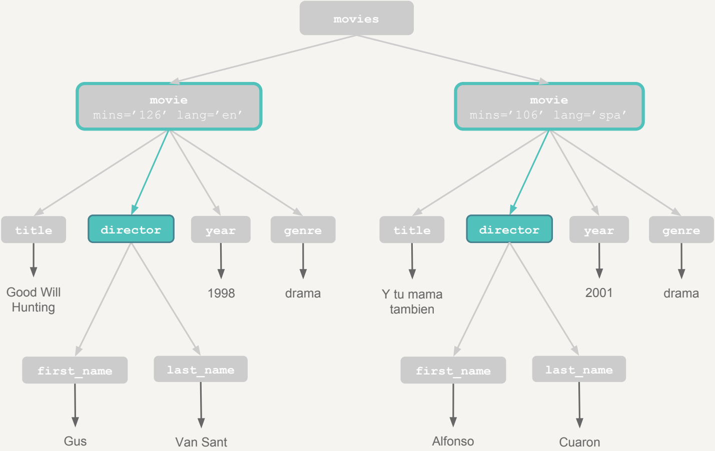
XPath: movie director's first name nodes

`/movies/movie/director/first_name`



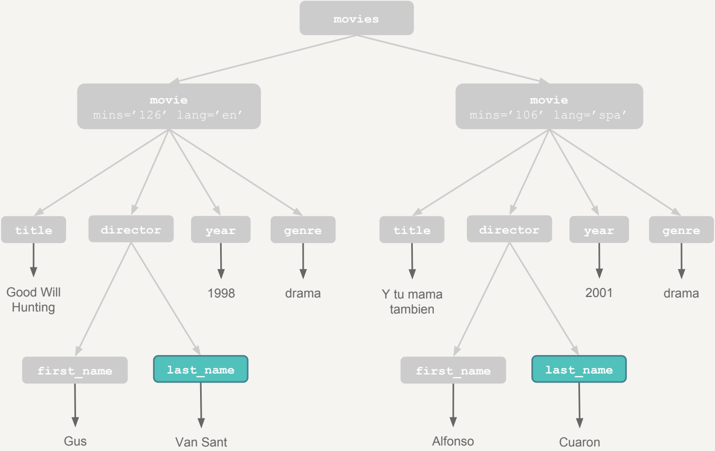
XPath: movie director nodes

`//movie/director`



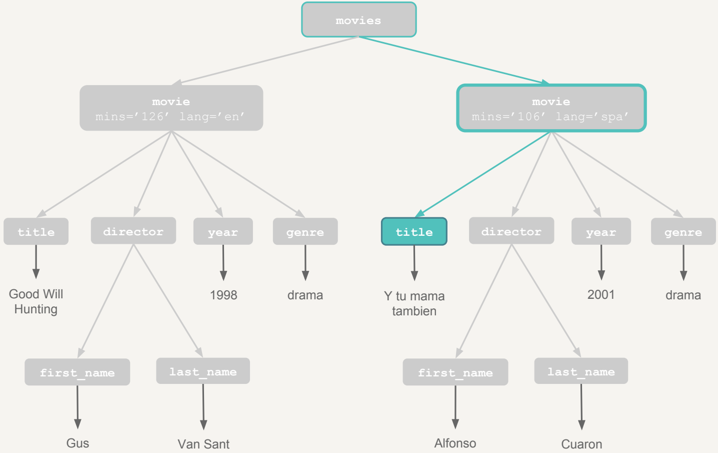
XPath: last name nodes

`//last_name`



XPath: itle node of movie in Spanish

`/movies/movie[@lang='spa']/title`



Querying Parsed Documents

XPath in ”XML”

XPath in "XML"

To work with XPath expressions using the "XML" package, we have the auxiliary function getNodeSet() that accepts XPath expressions in order to select node-sets. Its main usage is:

```
getNodeSet(doc, path)
```

where doc is an object of class "XMLInternalDocument" and path is a string giving the XPath expression to be evaluated

XML Movies Example

```r
# define some xml content
xml_string = c(
  '<?xml version="1.0" encoding="UTF-8"?>',
  '<movies>',
  '<movie mins="126" lang="eng">',
  '<title>Good Will Hunting</title>',
  '<director>',
  '<first_name>Gus</first_name>',
  '<last_name>Van Sant</last_name>',
  '</director>',
  '<year>1998</year>',
  '<genre>drama</genre>',
  '</movie>',
  '<movie mins="106" lang="spa">',
  '<title>Y tu mama tambien</title>',
  '<director>',
  '<first_name>Alfonso</first_name>',
  '<last_name>Cuaron</last_name>',
  '</director>',
  '<year>2001</year>',
  '<genre>drama</genre>',
  '</movie>',
  '</movies>')

# parse xml content
movies_xml = xmlParse(xml_string, asText = TRUE)
```

```r
# check movies_xml
movies_xml
```

```
## <?xml version="1.0" encoding="UTF-8"?>
## <movies>
##   <movie mins="126" lang="eng">
##     <title>Good Will Hunting</title>
##     <director>
##       <first_name>Gus</first_name>
##       <last_name>Van Sant</last_name>
##     </director>
##     <year>1998</year>
##     <genre>drama</genre>
##   </movie>
##   <movie mins="106" lang="spa">
##     <title>Y tu mama tambien</title>
##     <director>
##       <first_name>Alfonso</first_name>
##       <last_name>Cuaron</last_name>
##     </director>
##     <year>2001</year>
##     <genre>drama</genre>
##   </movie>
## </movies>
##
```

Getting Nodes

```r
# set of movie nodes
getNodeSet(movies_xml, "/movies/movie")


## [[1]]
## <movie mins="126" lang="eng">
##   <title>Good Will Hunting</title>
##   <director>
##     <first_name>Gus</first_name>
##     <last_name>Van Sant</last_name>
##   </director>
##   <year>1998</year>
##   <genre>drama</genre>
## </movie>
##
## [[2]]
## <movie mins="106" lang="spa">
##   <title>Y tu mama tambien</title>
##   <director>
##     <first_name>Alfonso</first_name>
##     <last_name>Cuaron</last_name>
##   </director>
##   <year>2001</year>
##   <genre>drama</genre>
## </movie>
##
## attr(,"class")
## [1] "XMLNodeSet"
```

```r
# equivalently
getNodeSet(movies_xml, "//movie")


## [[1]]
## <movie mins="126" lang="eng">
##   <title>Good Will Hunting</title>
##   <director>
##     <first_name>Gus</first_name>
##     <last_name>Van Sant</last_name>
##   </director>
##   <year>1998</year>
##   <genre>drama</genre>
## </movie>
##
## [[2]]
## <movie mins="106" lang="spa">
##   <title>Y tu mama tambien</title>
##   <director>
##     <first_name>Alfonso</first_name>
##     <last_name>Cuaron</last_name>
##   </director>
##   <year>2001</year>
##   <genre>drama</genre>
## </movie>
##
## attr(,"class")
## [1] "XMLNodeSet"
```

Getting Nodes

```r
# set of title nodes
getNodeSet(movies_xml, "//title")
```

```
## [[1]]
## <title>Good Will Hunting</title>
##
## [[2]]
## <title>Y tu mama tambien</title>
##
## attr(,"class")
## [1] "XMLNodeSet"
```

```r
# set of year nodes
getNodeSet(movies_xml, "//year")
```

```
## [[1]]
## <year>1998</year>
##
## [[2]]
## <year>2001</year>
##
## attr(,"class")
## [1] "XMLNodeSet"
```

```r
# set of director nodes
getNodeSet(movies_xml, "//director")
```

```
## [[1]]
## <director>
##   <first_name>Gus</first_name>
##   <last_name>Van Sant</last_name>
## </director>
##
## [[2]]
## <director>
##   <first_name>Alfonso</first_name>
##   <last_name>Cuaron</last_name>
## </director>
##
## attr(,"class")
## [1] "XMLNodeSet"
```

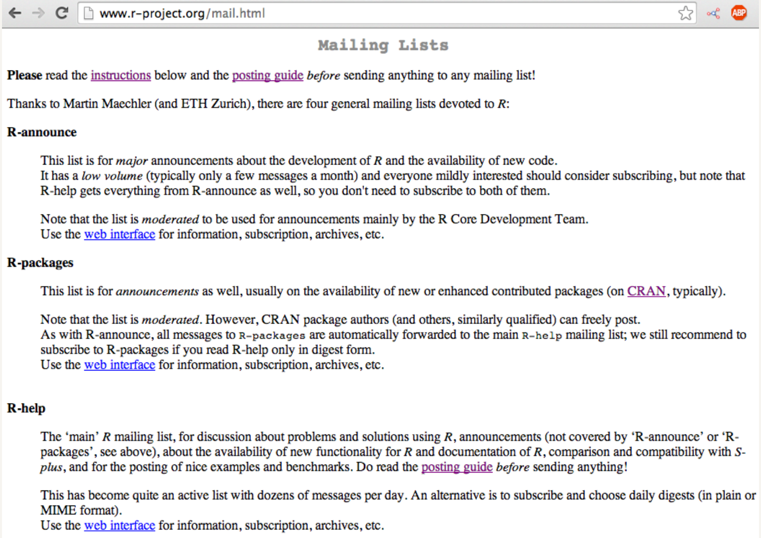
Getting Nodes

```
# set of movie nodes with attribute lang = 'eng'
getNodeSet(movies_xml, "//movie[@lang='eng']")


## [[1]]
## <movie mins="126" lang="eng">
##   <title>Good Will Hunting</title>
##   <director>
##     <first_name>Gus</first_name>
##     <last_name>Van Sant</last_name>
##   </director>
##   <year>1998</year>
##   <genre>drama</genre>
## </movie>
##
## attr(,"class")
## [1] "XMLNodeSet"
```

```
# set of movie nodes with attribute lang = 'spa'
getNodeSet(movies_xml, "//movie[@lang='spa']")


## [[1]]
## <movie mins="106" lang="spa">
##   <title>Y tu mama tambien</title>
##   <director>
##     <first_name>Alfonso</first_name>
##     <last_name>Cuaron</last_name>
##   </director>
##   <year>2001</year>
##   <genre>drama</genre>
## </movie>
##
## attr(,"class")
## [1] "XMLNodeSet"
```

Case Study

R Mailing Lists



Mailing Lists

Please read the instructions below and the posting guide *before* sending anything to any mailing list!

Thanks to Martin Maechler (and ETH Zurich), there are four general mailing lists devoted to *R*:

R-announce

This list is for *major* announcements about the development of *R* and the availability of new code.
It has a *low volume* (typically only a few messages a month) and everyone mildly interested should consider subscribing, but note that R-help gets everything from R-announce as well, so you don't need to subscribe to both of them.

Note that the list is *moderated* to be used for announcements mainly by the R Core Development Team.
Use the web interface for information, subscription, archives, etc.

R-packages

This list is for *announcements* as well, usually on the availability of new or enhanced contributed packages (on CRAN, typically).

Note that the list is *moderated*. However, CRAN package authors (and others, similarly qualified) can freely post.
As with R-announce, all messages to `R-packages` are automatically forwarded to the main `R-help` mailing list; we still recommend to subscribe to R-packages if you read R-help only in digest form.
Use the web interface for information, subscription, archives, etc.

R-help

The 'main' *R* mailing list, for discussion about problems and solutions using *R*, announcements (not covered by 'R-announce' or 'R-packages', see above), about the availability of new functionality for *R* and documentation of *R*, comparison and compatibility with *S-plus*, and for the posting of nice examples and benchmarks. Do read the posting guide *before* sending anything!

This has become quite an active list with dozens of messages per day. An alternative is to subscribe and choose daily digests (in plain or MIME format).
Use the web interface for information, subscription, archives, etc.

Parsing HTML document

Let's parse the R mailing lists page:

http://www.r-project.org/mail.html

```
# R mailing lists url
mailing_url = "http://www.r-project.org/mail.html"

# parse html content
mailing_doc = htmlParse(mailing_url)
```

```
# check class 'HTMLInternalDocument'
class(mailing_doc)


## [1] "HTMLInternalDocument" "HTMLInternalDocument" "XMLInternalDocument"
## [4] "XMLAbstractDocument"


# get root node
mailing_root = xmlRoot(mailing_doc)
```

Inspecting Root Node

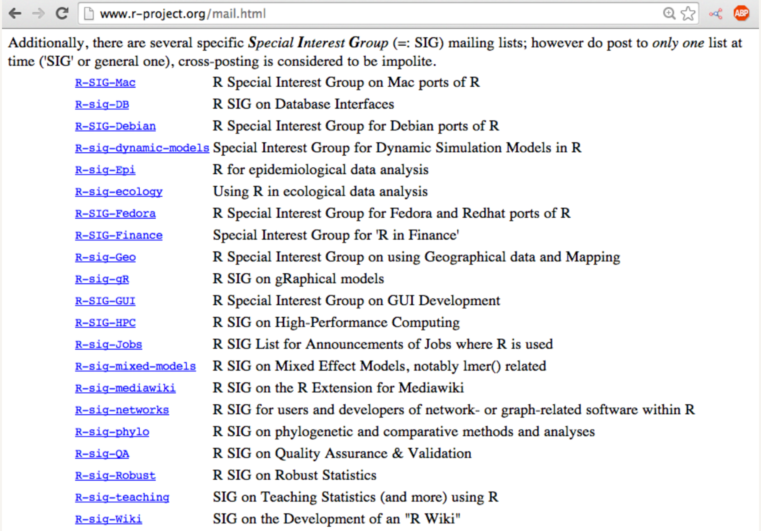
Let's inspect the Root Node

```
# how many child nodes in root
xmlSize(mailing_root)


## [1] 2


# names of root child nodes
xmlSApply(mailing_root, xmlName)


##   head   body
## "head" "body"


# how many nodes in each children
xmlSApply(mailing_root, xmlSize)


## head body
##    2   23
```

Getting HTML body element

```r
# get the html body node
mailing_body = xmlChildren(mailing_root)$body

# get all 'h1' elements
xpathSApply(mailing_body, "h1")


## [[1]]
## <h1>Mailing Lists</h1>


# get all 'h2' elements
xpathSApply(mailing_body, "h2")


## [[1]]
## <h2>Archives and Search Facilities</h2>
```

Special Interest Group

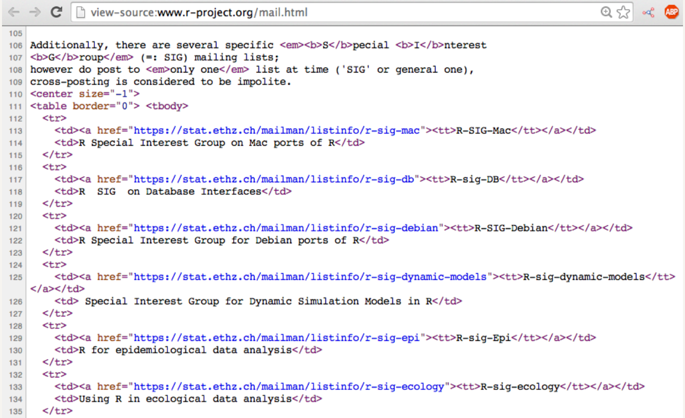


Additionally, there are several specific *Special Interest Group* (=: SIG) mailing lists; however do post to *only one* list at time ('SIG' or general one), cross-posting is considered to be impolite.

R-SIG-Mac	R Special Interest Group on Mac ports of R
R-sig-DB	R SIG on Database Interfaces
R-SIG-Debian	R Special Interest Group for Debian ports of R
R-sig-dynamic-models	Special Interest Group for Dynamic Simulation Models in R
R-sig-Epi	R for epidemiological data analysis
R-sig-ecology	Using R in ecological data analysis
R-SIG-Fedora	R Special Interest Group for Fedora and Redhat ports of R
R-SIG-Finance	Special Interest Group for 'R in Finance'
R-sig-Geo	R Special Interest Group on using Geographical data and Mapping
R-sig-gR	R SIG on gRaphical models
R-SIG-GUI	R Special Interest Group on GUI Development
R-SIG-HPC	R SIG on High-Performance Computing
R-sig-Jobs	R SIG List for Announcements of Jobs where R is used
R-sig-mixed-models	R SIG on Mixed Effect Models, notably lmer() related
R-sig-mediawiki	R SIG on the R Extension for Mediawiki
R-sig-networks	R SIG for users and developers of network- or graph-related software within R
R-sig-phylo	R SIG on phylogenetic and comparative methods and analyses
R-sig-QA	R SIG on Quality Assurance & Validation
R-sig-Robust	R SIG on Robust Statistics
R-sig-teaching	SIG on Teaching Statistics (and more) using R
R-sig-Wiki	SIG on the Development of an "R Wiki"

Special Interest Group

Take a look at the source code: note that it is a `table` node



```
Additionally, there are several specific <em><b>S</b>pecial <b>I</b>nterest
<b>G</b>roup</em> (=: SIG) mailing lists;
however do post to <em>only one</em> list at time ('SIG' or general one),
cross-posting is considered to be impolite.
<center size="-1">
<table border="0"> <tbody>
  <tr>
    <td><a href="https://stat.ethz.ch/mailman/listinfo/r-sig-mac"><tt>R-SIG-Mac</tt></a></td>
    <td>R Special Interest Group on Mac ports of R</td>
  </tr>
  <tr>
    <td><a href="https://stat.ethz.ch/mailman/listinfo/r-sig-db"><tt>R-sig-DB</tt></a></td>
    <td>R  SIG  on Database Interfaces</td>
  </tr>
  <tr>
    <td><a href="https://stat.ethz.ch/mailman/listinfo/r-sig-debian"><tt>R-SIG-Debian</tt></a></td>
    <td>R Special Interest Group for Debian ports of R</td>
  </tr>
  <tr>
    <td><a href="https://stat.ethz.ch/mailman/listinfo/r-sig-dynamic-models"><tt>R-sig-dynamic-models</tt></a></td>
    <td> Special Interest Group for Dynamic Simulation Models in R</td>
  </tr>
  <tr>
    <td><a href="https://stat.ethz.ch/mailman/listinfo/r-sig-epi"><tt>R-sig-Epi</tt></a></td>
    <td>R for epidemiological data analysis</td>
  </tr>
  <tr>
    <td><a href="https://stat.ethz.ch/mailman/listinfo/r-sig-ecology"><tt>R-sig-ecology</tt></a></td>
    <td>Using R in ecological data analysis</td>
  </tr>
```

Getting "Special Interest Groups"

Let's work with the *Special Interest Groups* section. We can read the content of the table with `readHTMLTable()`

```
# get the html table 'Special Interest Groups'
sig_content = readHTMLTable(mailing_doc, which = 1)

head(sig_content)
```

```
##                      V1
## 1           R-SIG-Mac
## 2            R-sig-DB
## 3        R-SIG-Debian
## 4 R-sig-dynamic-models
## 5           R-sig-Epi
## 6       R-sig-ecology
##                                                           V2
## 1               R Special Interest Group on Mac ports of R
## 2                           R  SIG  on Database Interfaces
## 3           R Special Interest Group for Debian ports of R
## 4 Special Interest Group for Dynamic Simulation Models in R
## 5                      R for epidemiological data analysis
## 6                      Using R in ecological data analysis
```

Getting "Special Interest Groups" (SIG)

We can also work with XPath expressions in order to get the information contained in the table of *Special Interest Groups*

For instance, let's work with the `a` elements in the `table` (i.e. the links)

```
# SIG table from doc 'HTMLInternalDocument'
sig_from_doc = xpathSApply(mailing_doc, "//table/..//a")

# SIG table from root 'XMLInternalElementNode'
sig_from_root = xpathSApply(mailing_root, "//table/..//a")
```

Getting "Special Interest Groups" (SIG)

Let's compare the results:

```
head(sig_from_doc, n = 2)


## [[1]]
## <a href="https://stat.ethz.ch/mailman/listinfo/r-sig-mac">
##   <tt>R-SIG-Mac</tt>
## </a>
##
## [[2]]
## <a href="https://stat.ethz.ch/mailman/listinfo/r-sig-db">
##   <tt>R-sig-DB</tt>
## </a>


head(sig_from_root, n = 2)


## [[1]]
## <a href="https://stat.ethz.ch/mailman/listinfo/r-sig-mac">
##   <tt>R-SIG-Mac</tt>
## </a>
##
## [[2]]
## <a href="https://stat.ethz.ch/mailman/listinfo/r-sig-db">
##   <tt>R-sig-DB</tt>
## </a>
```

Getting "Special Interest Groups" (SIG)

Here's one way to get the link names of the Special Interest Groups
(ie values in the first column of the html table)

```
# names of SIG links
sapply(sig_from_root, xmlValue)
```

```
##  [1] "R-SIG-Mac"            "R-sig-DB"             "R-SIG-Debian"
##  [4] "R-sig-dynamic-models" "R-sig-Epi"            "R-sig-ecology"
##  [7] "R-SIG-Fedora"         "R-SIG-Finance"        "R-sig-Geo"
## [10] "R-sig-gR"             "R-SIG-GUI"            "R-SIG-HPC"
## [13] "R-sig-Jobs"           "R-sig-mixed-models"   "R-sig-mediawiki"
## [16] "R-sig-networks"       "R-sig-phylo"          "R-sig-QA"
## [19] "R-sig-Robust"         "R-sig-teaching"       "R-sig-Wiki"
```

Getting ”Special Interest Groups” (SIG)

Extracting a elements in table

```
# XPath expression to extract 'a' nodes
sig_links = xpathSApply(mailing_root, "//table/..//a")
head(sig_links, n = 4)


## [[1]]
## <a href="https://stat.ethz.ch/mailman/listinfo/r-sig-mac">
##   <tt>R-SIG-Mac</tt>
## </a>
##
## [[2]]
## <a href="https://stat.ethz.ch/mailman/listinfo/r-sig-db">
##   <tt>R-sig-DB</tt>
## </a>
##
## [[3]]
## <a href="https://stat.ethz.ch/mailman/listinfo/r-sig-debian">
##   <tt>R-SIG-Debian</tt>
## </a>
##
## [[4]]
## <a href="https://stat.ethz.ch/mailman/listinfo/r-sig-dynamic-models">
##   <tt>R-sig-dynamic-models</tt>
## </a>
```

Getting "Special Interest Groups" (SIG)

Extracting attributes from a elements in table

```
# XPath expression attributes of nodes
sig_link_attrs = xpathSApply(mailing_root, "//table/..//a", xmlAttrs)
head(sig_link_attrs)
```

```
##                                                          href
##            "https://stat.ethz.ch/mailman/listinfo/r-sig-mac"
##                                                          href
##             "https://stat.ethz.ch/mailman/listinfo/r-sig-db"
##                                                          href
##         "https://stat.ethz.ch/mailman/listinfo/r-sig-debian"
##                                                          href
## "https://stat.ethz.ch/mailman/listinfo/r-sig-dynamic-models"
##                                                          href
##            "https://stat.ethz.ch/mailman/listinfo/r-sig-epi"
##                                                          href
##        "https://stat.ethz.ch/mailman/listinfo/r-sig-ecology"
```

Getting "Special Interest Groups" (SIG)

Extracting values from `href` attributes of `a` elements in `table`

```r
# XPath expression to extract attribute values of nodes
sig_link_attr_vals = xpathSApply(mailing_root, "//table/..//a", xmlGetAttr, "href")
head(sig_link_attr_vals, n = 10)
```

```
##  [1] "https://stat.ethz.ch/mailman/listinfo/r-sig-mac"
##  [2] "https://stat.ethz.ch/mailman/listinfo/r-sig-db"
##  [3] "https://stat.ethz.ch/mailman/listinfo/r-sig-debian"
##  [4] "https://stat.ethz.ch/mailman/listinfo/r-sig-dynamic-models"
##  [5] "https://stat.ethz.ch/mailman/listinfo/r-sig-epi"
##  [6] "https://stat.ethz.ch/mailman/listinfo/r-sig-ecology"
##  [7] "https://stat.ethz.ch/mailman/listinfo/r-sig-fedora"
##  [8] "https://stat.ethz.ch/mailman/listinfo/r-sig-finance"
##  [9] "https://stat.ethz.ch/mailman/listinfo/r-sig-geo"
## [10] "https://stat.ethz.ch/mailman/listinfo/r-sig-gr"
```

Getting ”Special Interest Groups” (SIG)

Extracting content from a elements in table

```
# XPath expression to extract content from nodes
sig_link_values = xpathSApply(mailing_root, "//table/..//a", xmlValue)

sig_link_values


##  [1] "R-SIG-Mac"            "R-sig-DB"             "R-SIG-Debian"
##  [4] "R-sig-dynamic-models" "R-sig-Epi"            "R-sig-ecology"
##  [7] "R-SIG-Fedora"         "R-SIG-Finance"        "R-sig-Geo"
## [10] "R-sig-gR"             "R-SIG-GUI"            "R-SIG-HPC"
## [13] "R-sig-Jobs"           "R-sig-mixed-models"   "R-sig-mediawiki"
## [16] "R-sig-networks"       "R-sig-phylo"          "R-sig-QA"
## [19] "R-sig-Robust"         "R-sig-teaching"       "R-sig-Wiki"
```

Getting "Special Interest Groups" (SIG)

Extracting second `td` elements in `table` (ie those containing the desccription of the SIGs)

```
# XPath expression to extract SIG descriptions
sig_desc_nodes = xpathSApply(mailing_root, "//table/tbody/tr/td[2]")

head(sig_desc_nodes, n = 5)


## [[1]]
## <td>R Special Interest Group on Mac ports of R</td>
##
## [[2]]
## <td>R  SIG  on Database Interfaces</td>
##
## [[3]]
## <td>R Special Interest Group for Debian ports of R</td>
##
## [[4]]
## <td> Special Interest Group for Dynamic Simulation Models in R</td>
##
## [[5]]
## <td>R for epidemiological data analysis</td>
```

Getting "Special Interest Groups" (SIG)

Extracting the content from the second `td` elements in `table` (ie the actual desccription of the SIGs)

```
# XPath expression to extract SIG descriptions
sig_descriptions = xpathSApply(mailing_root, "//table/tbody/tr/td[2]", xmlValue)

head(sig_descriptions, n = 10)


##  [1] "R Special Interest Group on Mac ports of R"
##  [2] "R  SIG  on Database Interfaces"
##  [3] "R Special Interest Group for Debian ports of R"
##  [4] " Special Interest Group for Dynamic Simulation Models in R"
##  [5] "R for epidemiological data analysis"
##  [6] "Using R in ecological data analysis"
##  [7] "R Special Interest Group for Fedora and Redhat ports of R"
##  [8] "Special Interest Group for 'R in Finance'"
##  [9] "R Special Interest Group on using Geographical data and Mapping"
## [10] "R  SIG  on  gRaphical models"
```