Parsing XML

STAT 133

Gaston Sanchez

Department of Statistics, UC–Berkeley

gastonsanchez.com
github.com/gastonstat
Course web: gastonsanchez.com/teaching/stat133

Parsing XML and HTML Content

Motivation

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

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"

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

Using "XML"

Remember to install "XML" first

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

# load XML
library(XML)
```

More info about "XML" at:

http://www.omegahat.org/RSXML

Parsing Functions

Parsing Functions

Main parsing functions in `"XML"`

- `xmlParse()`
- `xmlTreeParse()`
- `htmlParse()`
- `htmlTreeParse()`

Function xmlParse()

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

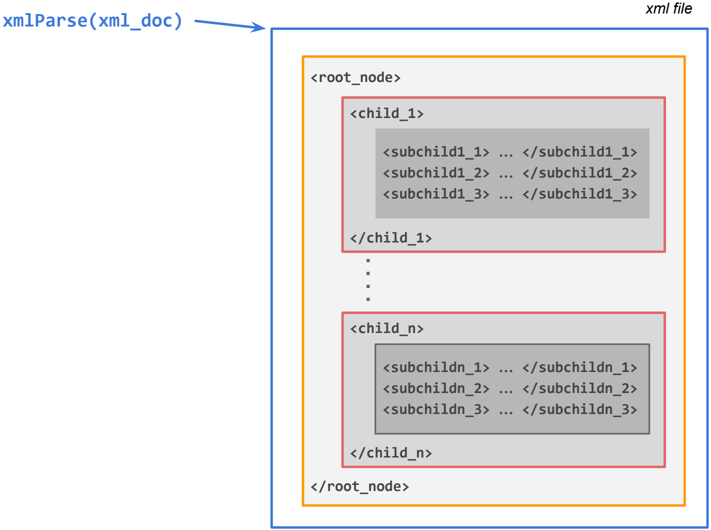
xmlParse()

Ultra simple example:

```r
doc <- xmlParse("<foo><bar>Some text</bar></foo>",
                asText = TRUE)

doc

## <?xml version="1.0"?>
## <foo>
##   <bar>Some text</bar>
## </foo>
##
```



xmlParse(xml_doc)

xml file

```
<root_node>
    <child_1>
        <subchild1_1> … </subchild1_1>
        <subchild1_2> … </subchild1_2>
        <subchild1_3> … </subchild1_3>
    </child_1>
    .
    .
    .
    .
    <child_n>
        <subchildn_1> … </subchildn_1>
        <subchildn_2> … </subchildn_2>
        <subchildn_3> … </subchildn_3>
    </child_n>
</root_node>
```

`xmlParse()` default behavior

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)

xmlParse() default behavior

Simple usage of `xmlParse()` on an XML document:

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

by default `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

About xmlParse() (con't)

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

```
# 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"
```

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" "XMLInternal
## [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" "XMLInternal
## [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)`

Parsing Functions

Function	relation with `xmlParse()`
`xmlParse()`	*default*
`xmlTreeParse()`	`useInternalNodes = FALSE`
`htmlParse()`	`isHTML = TRUE`

`htmlTreeParse()`	`isHTML = TRUE`
	`useInternalNodes = FALSE`

Working with Parsed Documents

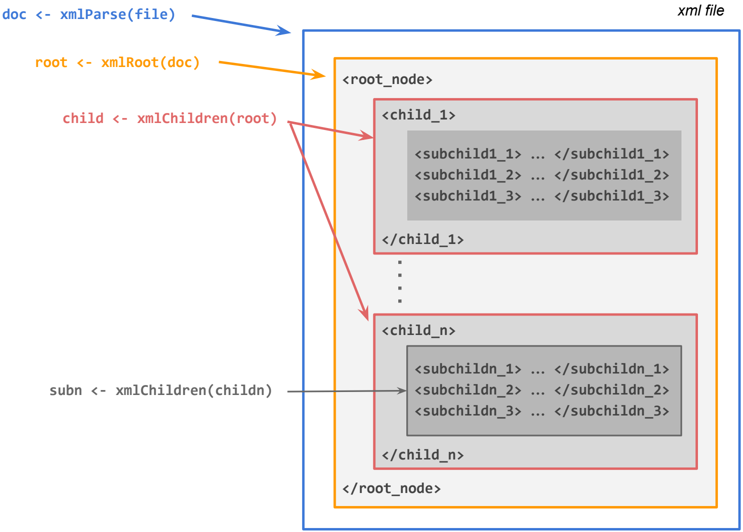
Parsed Documents

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> … </subchild1_1>
    <subchild1_2> … </subchild1_2>
    <subchild1_3> … </subchild1_3>
  </child_1>
  ⋮
  <child_n>
    <subchildn_1> … </subchildn_1>
    <subchildn_2> … </subchildn_2>
    <subchildn_3> … </subchildn_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)
```

Toy Example: Movies XML

```
# 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"
```

Movies XML: movie children

```
# 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

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

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

```
# 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.

Looping Over Nodes

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 an XMLNode:

```
# 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"
```

Case Study

XML

Example from www.xmlfiles.com
http://www.xmlfiles.com/examples/plant_catalog.xml

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 (i.e. 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

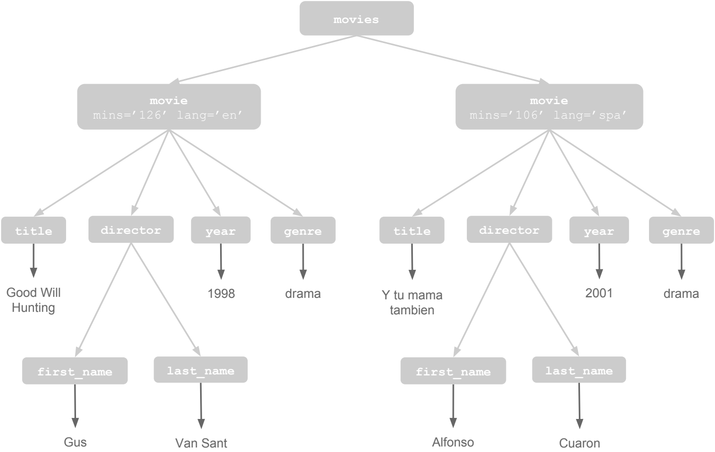
Selecting Unknown Nodes

XPath wildcards for unknown nodes

XPath wildcards can be used to select unknown XML elements

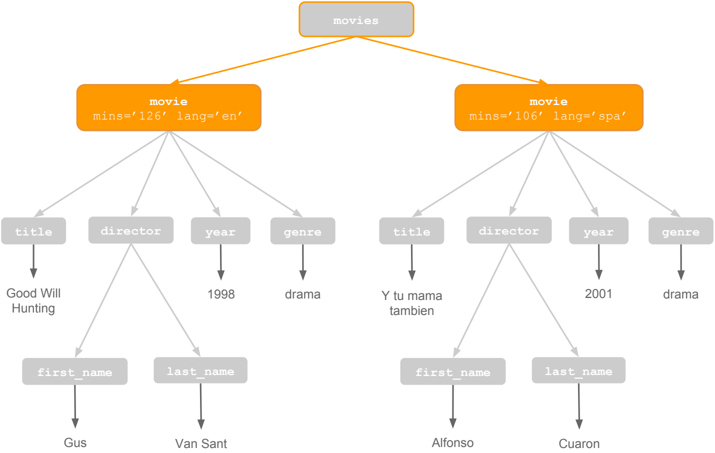
Symbol	Description
`*`	matches any element node
`@*`	matches any attribute node
`node()`	matches any node of any kind

Movies Tree Structure



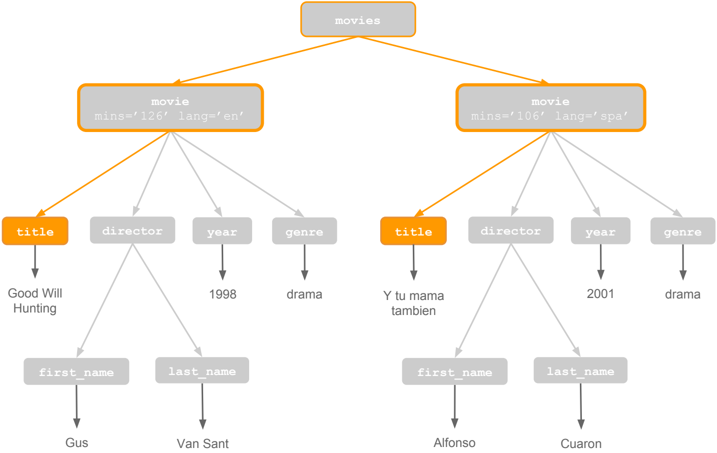
XPath: movie nodes

`/movies/movie`



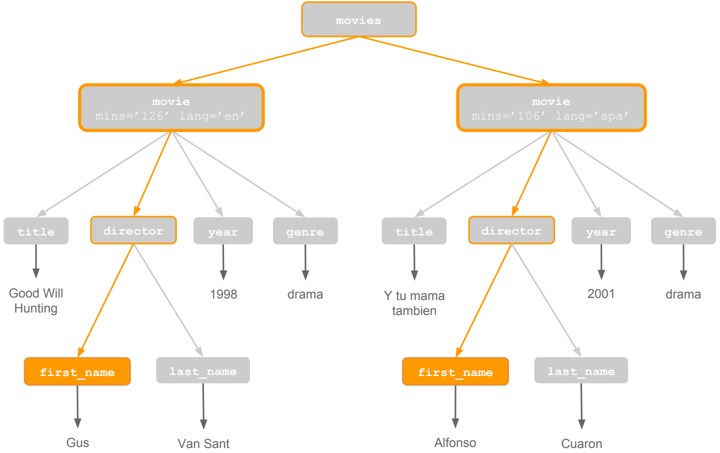
XPath: movie title nodes

`/movies/movie/title`



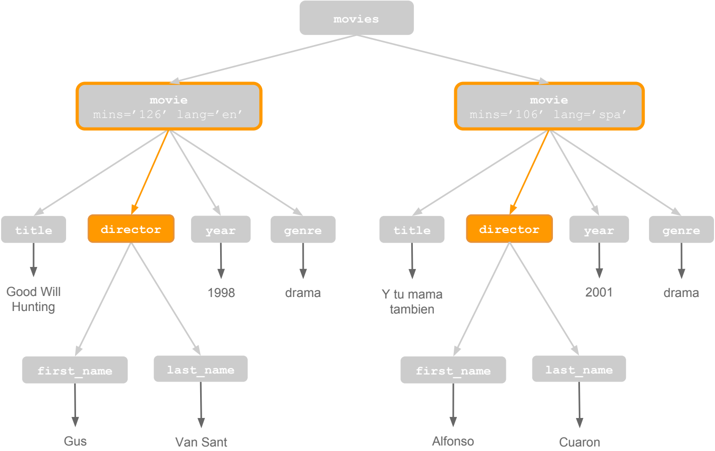
XPath: movie director's first name nodes

/movies/movie/director/first_name



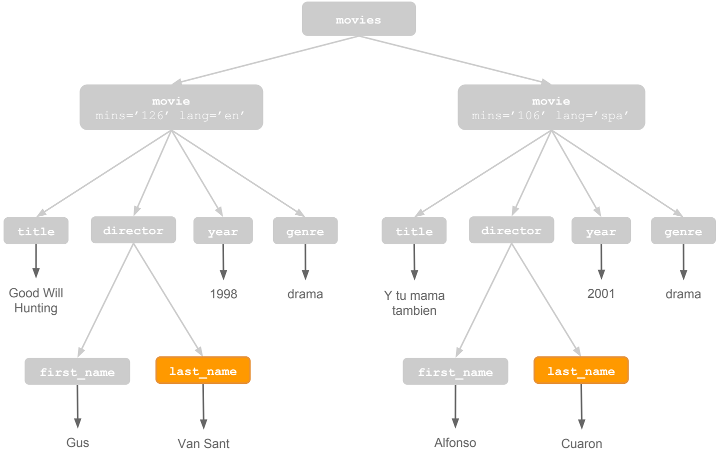
XPath: movie director nodes

`//movie/director`



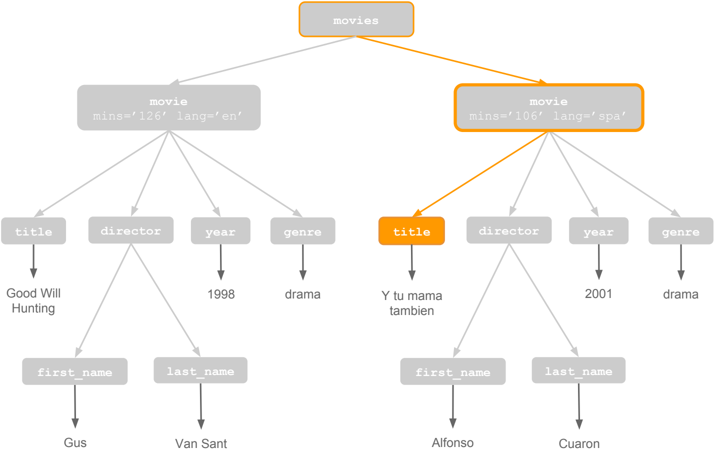
XPath: last name nodes

`//last_name`



XPath: title 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

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