The R4X package

Convenient XML Manipulation for R

Romain François

Outline

E4X

Ecmascript (Javacript) for XML. Example from Wikipedia:
http://en.wikipedia.org/wiki/E4X

```
var sales = <sales vendor="John">
   <item type="peas" price="4" quantity="6"/>
   <item type="carrot" price="3" quantity="10"/>
   <item type="chips" price="5" quantity="3"/>
 </sales>;

alert( sales.item.(@type == "carrot").@quantity );
alert( sales.@vendor );
for each( var price in sales..@price )
  alert( price );
```

The XML Package

from the $\hat{\Omega}$ project. `http://www.omegahat.org/RSXML/`

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.

The XML Package in 3 slides

Creating XML content

```
> x <- xmlNode( "test",
      xmlNode( "bar", attrs = c( fruit = "mango") ),
      xmlNode( "bar", attrs = c( fruit = "apple" )),
    attrs = c(type="foo"))
> x

<test type="foo">
 <bar fruit="mango"/>
 <bar fruit="apple"/>
</test>

> class(x)

[1] "XMLNode"              "RXMLAbstractNode" "XMLAbstractNode"
[4] "oldClass"
```

The XML package in 3 slides

Append content to an XML structure, the `addChildren` function

```
> x
```

```
<test type="foo">
 <bar fruit="mango"/>
 <bar fruit="apple"/>
</test>
```

```
> addChildren( x,
    xmlNode( "bar", attrs =
      c( fruit = "pineapple" ) ) )
```

```
<test type="foo">
 <bar fruit="mango"/>
 <bar fruit="apple"/>
 <bar fruit="pineapple"/>
</test>
```

The XML package in 3 slides

Query content of an XML structure

```
> # The "fruit" attribute of the first child of x
> xmlAttrs( xmlChildren(x)[[1]], "fruit" )
  fruit
"mango"

> # The "fruit" attribute of each child of x
> xmlApply( x, xmlAttrs, "fruit" )
$bar
  fruit
"mango"

$bar
  fruit
"apple"
```

Create XML objects

The xml generic function

The default method tries to convert strings into XML nodes, including nested nodes. Remember: Strings can be multiline in R.

```
> y <- xml( '<test><foo blah="1"/><bar/></test>')
> y <- xml( '
    <test>
       <foo blah="1"/>
       <bar/>
    </test>
  ')
> y
```

```
<test>
 <foo blah="1"/>
 <bar/>
</test>
```

```
> class( y )
```

```
[1] "XMLNode"            "RXMLAbstractNode" "XMLAbstractNode"
[4] "oldClass"
```

Dynamic content with brew

The brew package provides a jsp-like templating framework for R. The <%= operator is used by R4X to add dynamic content without having to use paste or sprintf.

```
> f <- c("mango", "apple", "strawberry" )
> x <- xml( '
    <fruits>
      <fruit><%= f[1] %></fruit>
      <fruit><%= f[2] %></fruit>
      <fruit><%= f[3] %></fruit>
    </fruits>
  ')
> x <- xml( '
    <fruits>
      <%for( i in f) {%>
        <fruit><%= i %></fruit>
      <%}%>
    </fruits>
  ')
```

For stronger taste, `distill` rather than `brew`

The `distill` function generates `brew` templates giving a syntax closer to E4X than pure brew code.

```
> x <- xml( txt <- '
    <fruits>
      <fruit>{f[1]}</fruit>
      <fruit>{f[2]}</fruit>
      <fruit>{f[3]}</fruit>
    </fruits>
  ')
> x <- xml( txt <- '
    <fruits>
      <@fruit~f>
        <fruit>{ fruit }</fruit>
      </@>
    </fruits>
  ')
```

loop generators

Loop generators are special xml tags starting with @ that are used to generate for loop code.

Distilling Tag	Corresponding brew code
<@i|n>	<% for(i in 1:n){ %>
<@i~x>	<% for(i in x){ %>
<@i?y>	<% for(i in seq(along=y)){ %>
</@>	<%}%>

```
> cat( txt )
  <fruits>
    <@fruit~f>
      <fruit>{ fruit }</fruit>
    </@>
  </fruits>
> cat( distill( txt ) )
  <fruits>
    <% for( fruit in f){%>
      <fruit><%= fruit %></fruit>
    <%}%>
  </fruits>
```

Adding content to a node

The + and %+=% operators (see package operators for details on %+=%)

```
> ( y <- x + '<fruit>blueberry</fruit>' )
```

```
<fruits>
 <fruit>mango</fruit>
 <fruit>apple</fruit>
 <fruit>strawberry</fruit>
 <fruit>blueberry</fruit>
</fruits>
```

```
> a <- "raspberry"
> y %+=% '<fruit>{a}</fruit>'
> y
```

```
<fruits>
 <fruit>mango</fruit>
 <fruit>apple</fruit>
 <fruit>strawberry</fruit>
 <fruit>blueberry</fruit>
 <fruit>raspberry</fruit>
</fruits>
```

Example 1:
Generating a tag cloud in xHTML

Tag cloud

Generating a simple tag cloud. See the operators package for details. Generated with the following script from the words used in all descriptions of R packages.

```
> all <- casefold( readLines( "descriptions.txt" ) )
> all <- all %s~% "/[^\\w\\s]//pg" %/~% "\\s+"
> all <- all %without% commonWords
> tab <- rev( sort( table( all ) ) )[1:250]
> words <- names(tab)
> for( word in words ){
    if( ( plural <- sprintf("%ss", word) ) %in% words ) {
      tab[word] <- tab[word] + tab[plural]
      tab[plural] <- 0
    }
  }
> tab <- tab[ tab != 0 ]
> tab <- tab[ sort(names(tab)) ]
> ncuts <- 8
> sizes <- as.numeric( cut ( tab, ncuts ) )
> refs <- round( seq( 10,24, length=ncuts) )
> words <- names(tab)
```

Tag cloud

Generating a simple tag cloud. R4X code to write the html page.

```
> tags <- xml( '
    <html>
      <head>
      <style type="text/css">
      <@i|ncuts>
            .cl{i}{
               font-size:{refs[i]}pt;
            }
      </@>
      </style>
      </head>
      <body>
        <@i|length(tab)>
          <span class="cl{sizes[i]}">{words[i]}</span>
        </@>
      </body>
    </html>' )
>   tags %>% "tags.html"
```


1 2 al algorithm allows analyses analysis applications applied approach arbitrary association available basic bayesian binary book bootstrap c calculate calculation carlo censored chain class classes classification cluster clustering code collection common components computation computational compute computing conditional confidence control correlation count covariates create currently curves data database datasets density described design designed detection different discrete display distance distribution either engineering environment error estimate estimating estimation estimator et etc exact examples experiments features file finance financial first fit fitting framework function functionality gaussian gene general generalized genetic graph graphical graphics group gui hazard hierarchical if implementation implemented implements include included including independent inference information interface intervals its kernel large level library likelihood linear local logistic main manipulating map markov matrices matrix maximum may mean measures method microarray missing mixture model modeling modelling monte most multiple multivariate network nonlinear nonparametric normal number object observations order output package parameter parametric perform plot plotting point population possible power probability problems procedure process processes program programming proportional provide provided quantitative r random regression related response results risk robust routines s sample sampling selection series set simple simulation single smoothing so software spatial specified splus squares standard statistical statistics structure support survival system teaching test testing theory through time tools trees univariate useful user uses using utilities utility value variable variance various vector version very wavelet way weighted work written

Manipulate XML objects

Example XML Structure

We will use this simple XML structure to demonstrate the slicing of objects of class `XMLNode`.

```
<root>
 <child id="1">
  <subchild id="sub1">foo</subchild>
  <subchild id="sub2">bar</subchild>
 </child>
 <child id="2">
  <subchild id="a">blah</subchild>
  <subchild id="b">bob</subchild>
  <something id="c"/>
 </child>
 <fruits>
  <fruit>banana</fruit>
  <fruit>mango</fruit>
 </fruits>
</root>
```

XPATH-like syntax

R4X defines an XPATH-like syntax to manipulate XML structures with the usual R extractors `[` and `[[`

path expression	[	[[
"child"	list	XMLNode
"child/subchild"	list	XMLNode
"child/subchild/#"	vector	vector
"child/subchild/#n"	numeric vector	numeric vector
"child/@id"	vector	vector
"child//@id"	vector	vector
"child/~sub.*"	list	XMLNode
"fruits"	XMLNode	XMLNode

Table: Classes of result for various path expressions.

slicing with [

The *single* square bracket [gives an XMLNode or a list of XMLNode if the path matches more than one node

```
> x[ "child" ]

$child
<child id="1">
 <subchild id="sub1">foo</subchild>
 <subchild id="sub2">bar</subchild>
</child>

$child
<child id="2">
 <subchild id="a">blah</subchild>
 <subchild id="b">bob</subchild>
 <something id="c"/>
</child>

> x[ "child/subchild[1]/@id" ]

 child  child
"sub1"    "a"
```

Appending content with `[<-.XMLNode`

The `[` extractor also works to add content to an XML structure using the XPath-like expressions.

```
> ( y <- xml( '<test/>' ) )

<test/>

> type <- "foo-bar"
> y[ "foo/bar/test" ] <- '<test type="{type}" />'
> y

<test>
 <foo>
  <bar>
   <test type="foo-bar"/>
  </bar>
 </foo>
</test>
```

Example 2:
Fetch information from RSS feeds

RSS: Example

Example RSS feed from `http://www.w3schools.com/rss`.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">
<channel>
  <title>W3Schools Home Page</title>
  <link>http://www.w3schools.com</link>
  <description>
     Free web building tutorials
  </description>
  <item>
    <title>RSS Tutorial</title>
    <link>http://www.w3schools.com/rss</link>
    <description>
        New RSS tutorial on W3Schools
    </description>
  </item>
</channel>
</rss>
```

Update on the Olympics

Fetching data from the BBC Olympics RSS feed.

```
> sport <- xml( url(
    "http://newsrss.bbc.co.uk/[...]/olympics/rss.xml") )
> titles <- sport[ "channel/item/title/#" ]
> cat( titles, sep = "\n" )
```

```
Live - Olympics
Sprinter Thanou barred from Ga [..]
Phelps claims first Beijing go [..]
China defend women's diving ti [..]
Cooke grabs first GB gold meda [..]
Ronaldinho shines in Brazil wi [..]
Rice sees off Hoff for shock g [..]
Park secures 400m freestyle go [..]
South Korea clinch archery gol [..]
Blake dodges showers to progre [..]
```

References

XML references from W3C:

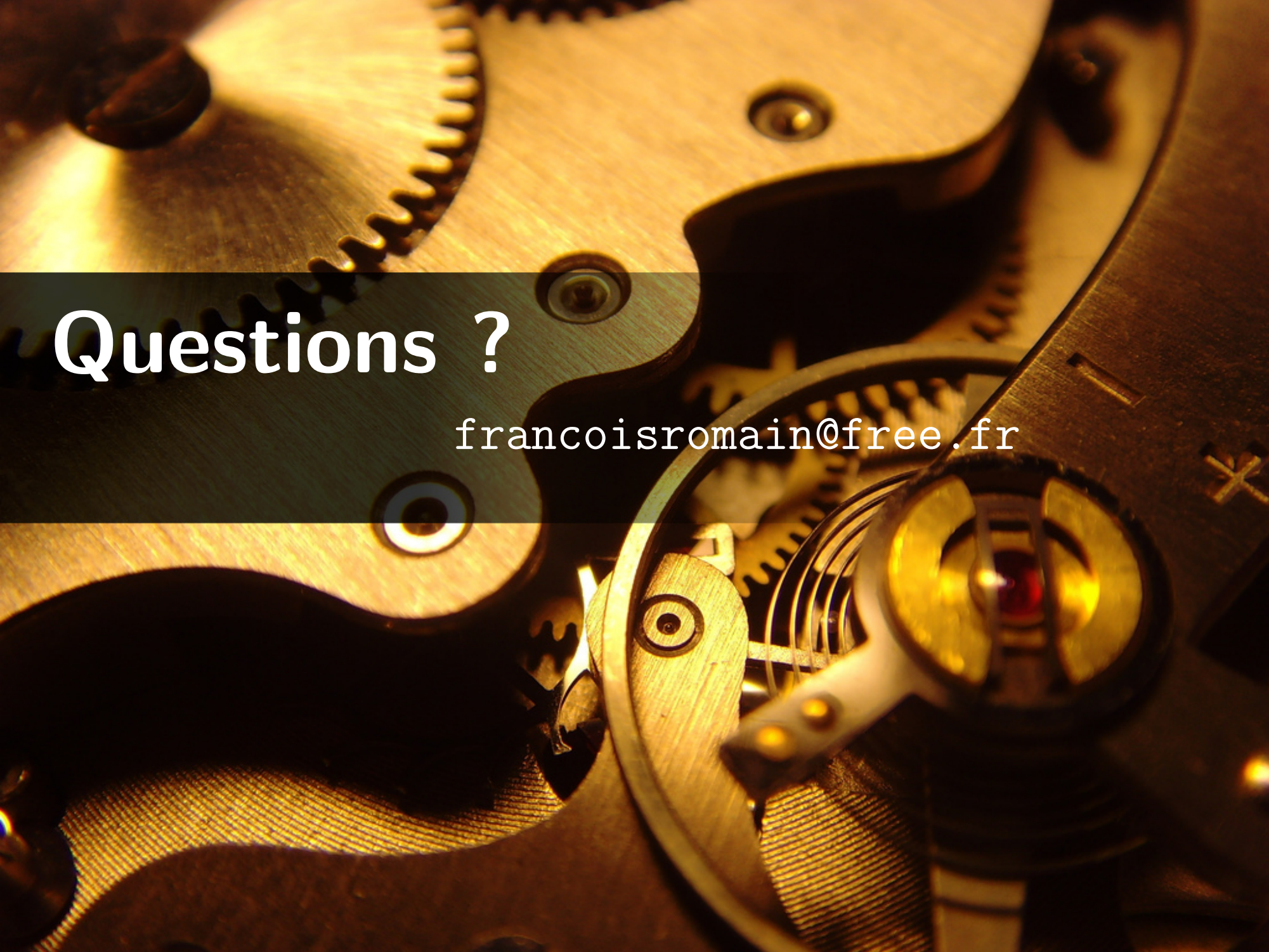
- E4X: `http://www.w3schools.com/e4x/default.asp`
- RSS: `http://www.w3schools.com/rss/default.asp`

R References

- `XML` ($\hat{\Omega}$). `http://www.omegahat.org/RSXML/`
- `brew`: `http://www.rforge.net/brew/`
- `operators`: `http://r-forge.r-project.org/projects/operators`

Pictures

- `http://www.flickr.com/photos/gamin/383003317/`
- `http://www.flickr.com/photos/27812866@N04/2748511595/`


Questions ?
francoisromain@free.fr