REXML: Ruby Electric XML

¿Que necesitamos?
Nada, viene integrado en la mayoria de las instalaciones (version 1.8.7 la tiene).

Formato basico

```
require 'rexml/document'
include REXML
file = [File.new("archPrueba.xml") | <string> ]
doc = Document.new(file)
puts doc
```

Linea 1: Incluimos la librería REXML.
Linea 2: Incluimos el nombre REXML, con lo que ahora no es necesario que hagamos *REXML::Document* todo el rato.
Linea 3: Abrimos el fichero XML, o lo recibimos mediante una cadena String. En nuestro caso recibiremos el XML
Linea 4: Generamos un objeto *Document,* que será con el que trabajemos mas adelante.

Accediendo a elementos y atributos

Supongamos que tenemos el siguiente fichero "bibliografia.xml":

```
<bibliography id="philosophy">
  <biblioentry id="ISBN0-19-285423-2">
     <author>
       <firstname>Bertrand</firstname>
       <surname>Russell</surname>
     </author>
     <title>The Problems of Philosophy</title>
     <publisher>
       <publishername>Oxford University Press</publishername>
     </publisher>
     <pubdate>1912</pubdate>
  </biblioentry>
  <biblioentry id='FHIW13C-1260' editorial='ESBN'>
     <author>
       <firstname>Sydney</firstname>
       <surname>Shoemaker</surname>
     </author>
     <pubdate>1984</pubdate>
  </biblioentry>
</bibliography>
```

Para las pruebas, usaremos *irb* (por comodidad).
Tras abrirlo (como en el primer caso), hacemos:

```
root = doc.root
```

Ahora, podemos decir que estamos "apuntando" al inicio del fichero XML. Cada objeto del XML, puede contener elementos, los cuales estan almacenados en un hash *elements* (**OJO**, va de 1 a N. Es un elemento *XPATH*, y estos elementos indexan desde 1).

```
irb(main):001:0> puts root.elements[1]
<biblioentry id='ISBN0-19-285423-2'>
     <author>
          <firstname>Bertrand</firstname>
          <surname>Russell</surname>
     </author>
     <title>The Problems of Philosophy</title>
     <publisher>
          <publishername>Oxford University Press</publishername>
     </publisher>
     <pubdate>1912</pubdate>
</biblioentry>
```

Estos elementos, pueden tener otros tantos objetos *Element*:

```
irb(main):001:0> puts root.elements[1].elements[2]
<title>The Problems of Philosophy</title>
```

Al ser un Hash, podemos acceder a ellos mediante su clave:

```
irb(main):001:0> puts  root.elements[1].elements["title"]
<title>The Problems of Philosophy</title>
```

Y de una manera mas simplificada, como si fuera una ruta de datos:

```
irb(main):001:0> puts  root.elements["biblioentry/title"]
<title>The Problems of Philosophy</title>
```

Luego, cada una de las tags de XML pueden contener unos atributos, a los que podemos acceder mediante el hash *attributes* (OJO: aquí no accedemos por numero, solo por clave):

```
irb(main):001:0> puts root.elements["biblioentry"].attributes["id"]
ISBN0-19-285423-2
```

Si queremos acceder al campo, lo llamamos mediante *text*

```
irb(main):001:0> puts root.elements["biblioentry/author/firstname"].text
Bertrand
```

Sacar por pantalla el nombre de la tag, mediante *name*:

```
irb(main):001:0> puts root.elements["biblioentry"].name
biblioentry
```

Si tenemos dos campos con el mismo tag (en nuestro caso *biblioentry*), podemos llamarlos como si fuera un hash: mediante un indice o usando la clave como si fuera una variable:

```
irb(main):001:0> puts root.elements["biblioentry[2]"]
<biblioentry id='FHIW13C-1260' editorial='ESBN'>
          <author>
               <firstname>Sydney</firstname>
               <surname>Shoemaker</surname>
          </author>
          <pubdate>1984</pubdate>
</biblioentry>

irb(main):002:0> puts root.elements["biblioentry[@id='FHIW13C-1260']"]
<biblioentry id='FHIW13C-1260' editorial='ESBN'>
          <author>
               <firstname>Sydney</firstname>
               <surname>Shoemaker</surname>
          </author>
          <pubdate>1984</pubdate>
</biblioentry>

irb(main):003:0> puts root.elements["biblioentry[@editorial='ESBN']"]
<biblioentry id='FHIW13C-1260' editorial='ESBN'>
          <author>
               <firstname>Sydney</firstname>
               <surname>Shoemaker</surname>
          </author>
          <pubdate>1984</pubdate>
</biblioentry>
```

Tambien tenemos la posibilidad de acceder de manera iterativa a los tags con mismo nombre, mediante *each_element*:

```
irb(main):001:0> root.each_element('//author') {|author| puts author}
<author>
     <firstname>Bertrand</firstname>
     <surname>Russell</surname>
</author>
<author>
     <firstname>Sydney</firstname>
     <surname>Shoemaker</surname>
</author>
```

Poniendo "//" delante del nombre del tag, Ruby buscará el tag a cualquier profundidad. Sin "//", buscará en el primer nivel unicamente:

```
irb(main):001:0> root.each_element('author') {|author| puts author}
=> []
```

Nota: *Element.each_element* es un atajo que crea Ruby de *Element.elements.each*.

Creando XML's e insertando datos

Lo primero de todo, necesitamos crear un documento:

```
irb(main):001:0> doc2 = Document.new
```

La raíz del XML será un tag *bibliography* y un atributo *id=philosophy* y *anyo=1927*. Lo crearemos mediante la funcion *add_element*:

```
irb(main):001:0> doc2.add_element("bibliography", {"id" => "philosophy", "anyo"=>"1927"})
=> <bibliography id='philosophy' anyo='1927'/>
```

Dentro creamos un nuevo tag, *biblioentry*:

```
irb(main):001:0> doc2.root.add_element("biblioentry")
=> <biblioentry/>
irb(main):002:0> puts doc2
<bibliography id='philosophy' anyo='1927'><biblioentry/></bibliography>
```

Ahora, dentro de *biblioentry* añadiremos un atributo, *ISBN*, y dos nuevos tag's, *author* y *title*:

```
irb(main):001:0> entrada = doc2.root.elements[1]
irb(main):002:0> entrada.add_attribute("ISBN", "12-345678-9")
irb(main):003:0> puts doc2
<bibliography id='philosophy' anyo='1927'>
    <biblioentry ISBN='12-345678-9'/>
</bibliography>
irb(main):004:0> autor = Element.new("author")
irb(main):005:0> autor.add_element("name")
irb(main):006:0> autor.add_element("surname")
irb(main):007:0> autor.elements["name"].text="Bertrand"
irb(main):008:0> autor.elements["surname"].text="Russell"
irb(main):009:0> entrada.elements << Element.new("title")
irb(main):010:0> entrada.elements["title"].text = "The problems of philosophy"
irb(main):011:0> entrada.elements << autor
irb(main):012:0> puts doc2
<bibliography id='philosophy' anyo='1927'>
  <biblioentry ISBN='12-345678-9'>
    <title>
        The problems of philosophy
    </title>
   <author>
      <name>Bertrand</name>
      <surname>Russell</surname>
   </author>
  </biblioentry>
</bibliography>
```

Para insertar elementos antes o despues de ciertos campos, podemos usar las funciones *insert_before* e *insert_after* respectivamente:

```
irb(main):001:0> fecha = Element.new("bth")
irb(main):002:0> dia = Element.new("day")
irb(main):003:0> dia.add_text("16")
irb(main):004:0> fecha << dia
irb(main):006:0> mes = Element.new("month")
irb(main):007:0> mes.add_text("August")
irb(main):008:0> fecha.insert_before("//day",mes)
irb(main):009:0> puts fecha
<bth>
  <month>August</month>
  <day>16</day>
</bth>
irb(main):010:0> doc2.root.insert_after("//surname",fecha)
irb(main):011:0> puts doc2
<bibliography id='philosophy' anyo='1927'>
  <biblioentry ISBN='12-345678-9'>
    <title>The problems of philosophy</title>
    <author>
      <name>Bertrand</name>
      <surname>Russell</surname>
      <bth>
        <month>August</month>
        <day>16</day>
      </bth>
    </author>
  </biblioentry>
</bibliography>
```

Como vemos, existen los atajos *add_element* y *add_attribute*. Tambien tenemos sus contrarios, *delete_element* y *delete_attribute*:

```
irb(main):167:0> doc2.root.elements[1].delete_attribute('ISBN')
irb(main):169:0> doc2.delete_element('//bth')
irb(main):170:0> doc2.root.elements[1].delete_element(1)
irb(main):171:0> puts doc2
<bibliography id='philosophy' anyo='1927'>
  <biblioentry>
    <author>
      <name>Bertrand</name>
      <surname>Russell</surname>
    </author>
  </biblioentry>
</bibliography>
```

Como vemos, podemos usar tanto una expresion XPATH (mediante "//") o el indice de su posición. Tambien, decir que las funciones devuelven los elementos que se hayan eliminado (tambien *delete_attribute* devuelve el *Element*).

Procesado de entidades

Cuando REXML parsea un documento, procesa las Definiciones de Tipo de Documento (DTD) y crea una tabla con cada uno de sus valores. Si esas entidades aparecen, REXML las sustituye por su valor:

```
irb(main):001:0> doc3 = Document.new('<!DOCTYPE testentity [
irb(main):002:1' <!ENTITY entity "test">]>
irb(main):003:1' <testentity>&entity; the entity</testentity>')
irb(main):004:0> puts doc3
<!DOCTYPE testentity [
<!ENTITY entity "test">
]>
<testentity>&entity; the entity</testentity>
irb(main):005:0> doc3.root.text
=> "test the entity"

irb(main):008:0> doc3.root.text = "test the &entity;"
irb(main):009:0> puts doc3
<!DOCTYPE testentity [
<!ENTITY entity "test">
]>
<testentity>&entity; the &amp;entity;</testentity>
irb(main):010:0> doc3.root.text
=> "test the test"
```

Stream parsing

El *stream parsing* es mas rapido que lo visto hasta ahora, *tree parsing*. La razón es que mientras en el segundo guardamos el XML en memoria y despues parseamos, en el *stream parsing* el parse es "al vuelo". Sin embargo, ciertas caracteristicas (como XPATH) no estan disponibles. Debemos implementar una clase *Listener* y cada vez que REXML detecte un evento (*start_tag*, *end_tag*, *text*), el oyente será notificado. Un ejemplo de implementacion:

```
require 'rexml/document'
require 'rexml/streamlistener'
include REXML

class Listener
  include StreamListener
  def tag_start(name, attributes)
    puts "Start #{name}"
  end
  def tag_end(name)
    puts "End #{name}"
  end
end

listener = Listener.new
parser = Parsers::StreamParser.new(File.new("bibliography2.xml"),
listener)
parser.parse
```

Problemas tipicos de uso

- *Extraer todos los datos de un documento XML.*

Tenemos un documento *orders.xml* con los siguientes datos:

```
<orders>
  <order>
    <number>105</number>
    <date>02/10/2006</date>
    <customer>Corner Store</customer>
    <items>
      <item upc="404100" desc="Red Roses" qty="240" />
      <item upc="412002" desc="Candy hearts" qty="160" />
    </items>
  </order>
</orders>
```

Nuestro script tendría que tener la siguiente forma:

```
require 'rexml/document'
include REXML
orders=Document.new(File.new("orders.xml"))
orders.root.each_element do |order|
  #cada <order> en <orders>
  order.each_element do |node|
    #<customer>,<item>, etc en <order>
    if node.has_elements?
      Node.each_element do |child|
        #Cada <item> en <items>
        puts "#{child.name}: #{child.attributes['desc']}"
      end
    else
      #El contenido de <number, <date>, etc.
      puts "#{node.name}: #{node.text}"
    end
  end
end
```

Y obtendriamos como salida:

```
number:105
date:02/10/2006
customer:Corner Store
item:Red Roses
item:Candy Hearts
```

Mas informacion: Rdoc → REXML::Element class

- *Stream Parsing*

Tenemos un XML event.xml de un tamaño considerable:

```
<events>
  <clean system=”dev” start=”01:35” end=”01:55”  area=”build” error=”1” />
  <backup system=”prod” start=”02:00” end=”02:35” size=”230014” error=”0” />
  <backup system=”dev” start=”02:00” end=”02:01” size=”0” error=”2” />
  <backup system=”test” start=”02:00” end=”02:47” size=”327450” error=”0” />
</events>
```

Deseamos recorrerlo lo mas rapido posible. Aquí nuestro script:

```
require 'rexml/document'
require 'rexml/streamlistener'

class ErrorListener
  include REXML::StreamListener
  def tag_start(name, attrs)
    if attrs[“error”] != nil and attrs[“error”] != 0
      puts %{Event “#{name}” failed for system “#{attrs[“system”]}” } +
      %{with code “#{attrs[“error”]}}
    end
  end
end
```

REXML::Document.parse_stream(File.new(“event.xml”),ErrorListener.new)

y obtendriamos como salida:

```
Event “clean” failed for system “dev” with code 1
Event “backup” failed for system “dev” with code 2
```

Mas informacion: Rdoc → REXML::StreamParser class

- *Navegar en el documento mediante XPATH*

Supongamos que tenemos el siguiente *doc.xml*:

```
<aquarium>
  <fish color=”blue” size=”small” />
  <fish color=”orange” size=”large”>
    <fish color=”green” size=”small”>
      <fish color=”red” size=”tiny” />
    </fish>
  </fish>
  <decoration type=”castle” style=”gaudy” >
    <algae color=”green” />
  </decoration>
</aquarium>
```

```
require 'rexml/document'
include REXML
doc = Document.new(File.new("doc.xml"))
```

Ahora, queremos buscar el primer objeto con tag <fish>:

```
XPath.first(doc,'//fish')
=> <fish color="blue" size="small" />
```

Un array con todos los elementos de color verde:

```
Xpath.match(doc,'//[@color="green"]')
=>[<fish color='green' size='small' > … </>, <algae color='green' />]
```

Mas informacion: Rdoc → REXML::XPath class

Bibliografía:

- *Ruby cookbook Ed. O'Reilly.*
- *Processing XML in Ruby*
- *DTD: Wikipedia*