RUBY/XML, XSLT AND XPATH TUTORIAL

What is XML ?

The Extensible Markup Language *XML* is a markup language much like HTML or SGML. This is recommended by the World Wide Web Consortium and available as an open standard.

XML is a portable, open source language that allows programmers to develop applications that can be read by other applications, regardless of operating system and/or developmental language.

XML is extremely useful for keeping track of small to medium amounts of data without requiring a SQL-based backbone.

XML Parser Architectures and APIs:

There are two different flavors available for XML parsers:

- **SAX-like** ***Streaminterfaces*** **:** Here you register callbacks for events of interest and then let the parser proceed through the document. This is useful when your documents are large or you have memory limitations, it parses the file as it reads it from disk, and the entire file is never stored in memory.

- **DOM-like** ***Objecttreeinterfaces*** **:** This is World Wide Web Consortium recommendation wherein the entire file is read into memory and stored in a hierarchical *tree − based* form to represent all the features of an XML document.

SAX obviously can't process information as fast as DOM can when working with large files. On the other hand, using DOM exclusively can really kill your resources, especially if used on a lot of small files.

SAX is read-only, while DOM allows changes to the XML file. Since these two different APIs literally complement each other there is no reason why you can't use them both for large projects.

Parsing and Creating XML using Ruby:

The most common way to manipulate XML is with the REXML library by Sean Russell. Since 2002, REXML has been part of the standard Ruby distribution.

REXML is a pure-Ruby XML processor conforming to the XML 1.0 standard. It is a *nonvalidating* processor, passing all of the OASIS nonvalidating conformance tests.

REXML parser has the following advantages over other available parsers:

- It is written 100 percent in Ruby.
- It can be used for both SAX and DOM parsing.
- It is lightweight . less than 2000 lines of code.
- Methods and classes are really easy-to-understand.
- SAX2-based API and Full XPath support.
- Shipped with Ruby installation and no separate installation is required.

For all our XML code examples, let's use a simple XML file as an input:

```
<collection shelf="New Arrivals">
<movie title="Enemy Behind">
   <type>War, Thriller</type>
   <format>DVD</format>
   <year>2003</year>
   <rating>PG</rating>
   <stars>10</stars>
   <description>Talk about a US-Japan war</description>
</movie>
<movie title="Transformers">
   <type>Anime, Science Fiction</type>
   <format>DVD</format>
   <year>1989</year>
   <rating>R</rating>
   <stars>8</stars>
   <description>A schientific fiction</description>
</movie>
   <movie title="Trigun">
   <type>Anime, Action</type>
   <format>DVD</format>
   <episodes>4</episodes>
   <rating>PG</rating>
   <stars>10</stars>
   <description>Vash the Stampede!</description>
</movie>
<movie title="Ishtar">
   <type>Comedy</type>
   <format>VHS</format>
   <rating>PG</rating>
   <stars>2</stars>
   <description>Viewable boredom</description>
</movie>
</collection>
```

DOM-like Parsing:

Let's first parse our XML data in *tree fashion*. We begin by requiring the **rexml/document** library; often we do an include REXML to import into the top-level namespace for convenience.

```
#!/usr/bin/ruby -w

require 'rexml/document'
include REXML

xmlfile = File.new("movies.xml")
xmldoc = Document.new(xmlfile)

# Now get the root element
root = xmldoc.root
puts "Root element : " + root.attributes["shelf"]

# This will output all the movie titles.
xmldoc.elements.each("collection/movie"){
   |e| puts "Movie Title : " + e.attributes["title"]
}

# This will output all the movie types.
xmldoc.elements.each("collection/movie/type") {
   |e| puts "Movie Type : " + e.text
}

# This will output all the movie description.
xmldoc.elements.each("collection/movie/description") {
   |e| puts "Movie Description : " + e.text
}
```

This will produce the following result:

```
Root element : New Arrivals
Movie Title : Enemy Behind
Movie Title : Transformers
Movie Title : Trigun
Movie Title : Ishtar
Movie Type : War, Thriller
Movie Type : Anime, Science Fiction
Movie Type : Anime, Action
Movie Type : Comedy
Movie Description : Talk about a US-Japan war
Movie Description : A schientific fiction
Movie Description : Vash the Stampede!
Movie Description : Viewable boredom
```

SAX-like Parsing:

To process the same data, *movies.xml*, file in a *stream-oriented* way we will define a *listener* class whose methods will be the target of *callbacks* from the parser.

NOTE: It is not suggested to use SAX-like parsing for a small file, this is just for a demo example.

```ruby
#!/usr/bin/ruby -w

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


class MyListener
  include REXML::StreamListener
  def tag_start(*args)
    puts "tag_start: #{args.map {|x| x.inspect}.join(', ')}"
  end

  def text(data)
    return if data =~ /^\w*$/     # whitespace only
    abbrev = data[0..40] + (data.length > 40 ? "..." : "")
    puts "  text   :   #{abbrev.inspect}"
  end
end

list = MyListener.new
xmlfile = File.new("movies.xml")
Document.parse_stream(xmlfile, list)
```

This will produce the following result:

```
tag_start: "collection", {"shelf"=>"New Arrivals"}
tag_start: "movie", {"title"=>"Enemy Behind"}
tag_start: "type", {}
  text   :   "War, Thriller"
tag_start: "format", {}
tag_start: "year", {}
tag_start: "rating", {}
tag_start: "stars", {}
tag_start: "description", {}
  text   :   "Talk about a US-Japan war"
tag_start: "movie", {"title"=>"Transformers"}
tag_start: "type", {}
  text   :   "Anime, Science Fiction"
tag_start: "format", {}
tag_start: "year", {}
tag_start: "rating", {}
tag_start: "stars", {}
tag_start: "description", {}
  text   :   "A schientific fiction"
tag_start: "movie", {"title"=>"Trigun"}
tag_start: "type", {}
  text   :   "Anime, Action"
tag_start: "format", {}
tag_start: "episodes", {}
tag_start: "rating", {}
tag_start: "stars", {}
tag_start: "description", {}
```

```
  text   :   "Vash the Stampede!"
tag_start: "movie", {"title"=>"Ishtar"}
tag_start: "type", {}
tag_start: "format", {}
tag_start: "rating", {}
tag_start: "stars", {}
tag_start: "description", {}
  text   :   "Viewable boredom"
```

XPath and Ruby:

An alternative way to view XML is XPath. This is a kind of pseudo-language that describes how to locate specific elements and attributes in an XML document, treating that document as a logical ordered tree.

REXML has XPath support via the *XPath* class. It assumes tree-based parsing *documentobjectmodel* as we have seen above.

```
#!/usr/bin/ruby -w

require 'rexml/document'
include REXML

xmlfile = File.new("movies.xml")
xmldoc = Document.new(xmlfile)

# Info for the first movie found
movie = XPath.first(xmldoc, "//movie")
p movie

# Print out all the movie types
XPath.each(xmldoc, "//type") { |e| puts e.text }

# Get an array of all of the movie formats.
names = XPath.match(xmldoc, "//format").map {|x| x.text }
p names
```

This will produce the following result:

```
<movie title='Enemy Behind'> ... </>
War, Thriller
Anime, Science Fiction
Anime, Action
Comedy
["DVD", "DVD", "DVD", "VHS"]
```

XSLT and Ruby:

There are two XSLT parsers available that Ruby can use. A brief description of each is given here:

Ruby-Sablotron:

This parser is written and maintained by Masayoshi Takahashi. This is written primarily for Linux OS and requires the following libraries:

- Sablot
- Iconv
- Expat

You can find this module at Ruby-Sablotron.

XSLT4R:

XSLT4R is written by Michael Neumann and can be found at the RAA in the Library section under

XML. XSLT4R uses a simple commandline interface, though it can alternatively be used within a third-party application to transform an XML document.

XSLT4R needs XMLScan to operate, which is included within the XSLT4R archive and which is also a 100 percent Ruby module. These modules can be installed using standard Ruby installation method $i.e., rubyinstall.rb$.

XSLT4R has the following syntax:

```
ruby xslt.rb stylesheet.xsl document.xml [arguments]
```

If you want to use XSLT4R from within an application, you can include XSLT and input the parameters you need. Here is the example:

```
require "xslt"

stylesheet = File.readlines("stylesheet.xsl").to_s
xml_doc = File.readlines("document.xml").to_s
arguments = { 'image_dir' => '/....' }

sheet = XSLT::Stylesheet.new( stylesheet, arguments )

# output to StdOut
sheet.apply( xml_doc )

# output to 'str'
str = ""
sheet.output = [ str ]
sheet.apply( xml_doc )
```

Further Reading:

- For a complete detail on REXML Parser, please refer to standard documentation for REXML Parser Documentation.
- You can download XSLT4R from RAA Repository.