Ruby XML Mapping

Marc Seeger (20488)
mail@marc-seeger.de
Computer Science and Media,
HdM Stuttgart, WS 08/09
Agenda

- Ruby
- XML Mapping Libraries
- YAML
A language that doesn't affect the way you think about programming, is not worth knowing

Alan Perlis
History

- Origin:
  - Yukihiro "Matz" Matsumoto
  - Japan 1993

- 1st english book: 2000
- Ruby on Rails: 2004
Variable Declaration:

```ruby
text = "Hallo Welt"  # String
zahl = 3.5           # Float
bla = 3              # Fixnum
blubb = 23252345863465364564564563 # Bignum
```

Function Declaration:

```ruby
def do_something(text, number)
  puts text * 3
  puts number * 3
end
```

```
do_something("Marc", 4)
→ MarcMarcMarc
12
```
Language Basics

- ClassNames
- method_names and variable_names
- methods_asking_a_question?
- slightly_dangerous_methods!
- @instance_variables
- $global_variables
- SOME_CONSTANTS or OtherConstants
Language Basics: Hashes

1. \( h = \{ \text{dog} \rightarrow \text{wuff}, \text{cat} \rightarrow \text{miau}, \text{donkey} \rightarrow \text{ihah} \} \)
2. \( h.\text{length} \) ➞ 3
3. \( h[\text{dog}] \) ➞ \text{wuff}
4. \( h[\text{cow}] = \text{muh} \)
5. \( h[\text{cat}] = 7 \)
6. \( h = \{ \text{cow} \rightarrow \text{muh}, \text{cat} \rightarrow 7, \text{donkey} \rightarrow \text{ihah}, \text{dog} \rightarrow \text{wuff} \} \)
Ruby is a syntactic sugar factory

Principle of Least Surprise
Language Basics: Arrays

1. `a = [ 3.14159, "pie", 99, "Blubb" ]`
2. `a.type` ➞ **Array**
3. `a.length` ➞ 3
4. `a[1]` ➞ "pie"
5. `a[4]` ➞ **nil**
6. `a[-1]` ➞ "Blubb"
7. `a[-2]` ➞ 99
8. `a[1, 3]` ➞ ["pie", 99, "Blubb"]
9. `a[0..2]` ➞ [3.14159, "pie", 99]

10. `b = Array.new`
11. `b.type` ➞ **Array**
12. `b.length` ➞ 0
13. `b[0] = "second"`
14. `b[1] = "array"`
15. `b` ➞ ["second", "array"]
people = Array.new
people << "Marc" << "Christian" << "Jakob" << "Michael"
people = ["Marc", "Christian", "Jakob", "Michael"]
people.push("Marc", "Christian", "Jakob", "Michael")
people = %w("Marc", "Christian", "Jakob", "Michael")
5.times { print "Hallo HdM!" }
Control Structures

```ruby
if expr [then]
  expr...
[elsif expr [then]
  expr...]]...
[else
  expr...]
end

until expr [do]
  ...
end

puts "Error!" if $debug

puts "Error!" unless $production_mode
```
Ruby for Java-Coders
Ruby for Java-Coders

- No Primitives, Integers and floats are objects!
  - -1.abs => 1
Ruby for Java-Coders

No Semi-colons
Ruby for Java-Coders

- nil, not null
- nil is an object!
  - nil.nil? => true
  - nil.class => NilClass
- nil and false are false
  - everything else, including 0, is true
Ruby for Java-Coders

- Expression oriented syntax.
  - Almost everything returns a value
  - Methods automatically return their last expression.
Ruby for Java-Coders

- Single Inheritance
  - But mixins are available (= Interface with implemented methods)
module BarModule
  def hello_world
    puts "Hello World"
  end
end

class BaseClass
  def class_method
    puts "In class method"
  end
end

class Foo < BaseClass
  include BarModule
end

f = Foo.new
f.class_method
f.hello_world

← This module implements the mixin

← A class that doesn’t do that much

← inheriting
← and mixing!

← We inherited that one
← And mixed in that one
Classes are always open (even built in classes)

```ruby
class String
  def foo
    "foo"
  end
end

puts "hdm test".foo #=> "foo"
```

Another Example from Rails:
1.hour.from_now
Features
Paradigms

- Procedural
- Object Oriented
- Functional
Everything is an object

ORLY?

YA RLY
Everything is an object

```
irb(main):001> 42.methods

irb(main):005> nil.class
=> NilClass

irb(main):002> nil.hallo_hdm
NoMethodError: undefined method `hallo_hdm' for nil:NilClass
  from (irb):2
```
Typing: strong / weak

- Strong typing
  - "5" / 2 → "NoMethodError"

- Weak typing
  - "16" / 2 → 8 (e.g. in Perl)

Ruby is strongly typed! (Java too)
Explicit: int a = 5
Implicit: a = 5

Ruby is implicitly typed! (Java explicitly)
Typing: static / dynamic

- **Static typing**
  - The compiler checks types during compilation

- **Dynamic typing**
  - The compiler doesn’t check types during compilation

Ruby uses dynamic typing (Java uses static typing)
Blocks

“Blocks are unnamed functions”
Blocks

Define:

```ruby
def foo &proc
  proc.call 2
  proc.call 4
  proc.call 6
end
```

```ruby
def foo
  yield 2
  yield 4
  yield 6
end
```

Call:

```ruby
foo{|some_number|
  puts some_number * 3
}
```

Result:

```
6
12
18
```
Blocks Example: Iterators

The current piece of the collection we are working with

```ruby
some_collection.each { |item| puts item }
```

What we are going to do with it

```ruby
some_collection.select { |item| item =~ /\[xz]/ }
some_collection.reject { |item| item =~ /\[xz]/ }
```
A closure object has:
• **code** to run (the executable)
• **state** around the code (the scope)

So you capture the environment, namely the local variables, in the closure. As a result, you can refer to the local variables inside a closure.

def method_that_returns_a_block( x )
    some_value = x * 12
    return Proc.new { puts "The value of X was *#{x}*, causing some_value to be #{some_value}" }
end

block = method_that_returns_a_block(5)
block.call
Idea: Function returning a function

```python
# Builds a function that returns true when 'f' returns false, and vice versa.
def complement(f):
    return lambda *args: not f(*args)
end
```

In action:

```python
is_even = lambda n: n % 2 == 0
is_odd = complement(is_even)

is_odd.call(1)  # true
is_odd.call(2)  # false
```
Parts of Ruby

- Libraries ("gems")
- Standard API
- Core API
- Ruby VM
Execution Environments

- Ruby VM (Ruby 1.8)
- YARV (aka Ruby 1.9)
- Rubinius
- MacRuby
- Jruby
- IronRuby
- Hotruby
Package Management

- Gem:
  - Search:
    ```
    C:\Users\Marc>gem search -r mapping
    *** REMOTE GEMS ***
    dm-mapping <0.7.0>
    xml-mapping <0.8.1>
    ```
  - Installing:
    ```
    C:\Users\Marc>gem install xml-mapping
    Successfully installed xml-mapping-0.8.1
    1 gem installed
    Installing ri documentation for xml-mapping-0.8.1...
    Installing RDoc documentation for xml-mapping-0.8.1...
    ```
Package Management

- **Gem:**
  
  - **Usage:**

```
irb(main):001:0> require "rubygems"
=> false
irb(main):002:0> require "hpricot"
=> true
irb(main):003:0> Hpricot.methods
=> ["private_class_method", "inspect", "name", "uxs", "tap", "ect", "clone", "public_methods", "__send__", "method_defined?", _e_variable_defined?", "equal?", "freez
```
Enough with the Jibber-Jabber
Mapping Libraries

- XML → Object
  - XML-Object
  - XmlSimple

- XML ↔ Object
  - ROXML
  - XML::MAPPING
  - HappyMapper
Tools like JSON or YAML are a much better fit for this kind of job, but one doesn’t always have that luxury."
... attempts to make the accessing of small, well-formed XML structures convenient, by providing a syntax that fits well in most Ruby programs.
<recipe name="bread" prep_time="5 mins" cook_time="3 hours">
  <title>Basic bread</title>
  <ingredient amount="8" unit="dL">Flour</ingredient>
  <ingredient amount="10" unit="grams">Yeast</ingredient>
  <ingredient amount="4" unit="dL" state="warm">Water</ingredient>
  <ingredient amount="1" unit="teaspoon">Salt</ingredient>
  <instructions easy="true" hard="false">
    <step>Mix all ingredients together.</step>
    <step>Knead thoroughly.</step>
    <step>Cover with a cloth, and leave for one hour in warm room.</step>
    <step>Knead again.</step>
    <step>Place in a bread baking tin.</step>
    <step>Cover with a cloth, and leave for one hour in warm room.</step>
    <step>Bake in the oven at 180(degrees)C for 30 minutes.</step>
  </instructions>
</recipe>

require 'xml-object'
recipe = XMLObject.new(File.open('recipe.xml'))

recipe.name => "bread"
recipe.title => "Basic bread"

recipe.ingredients.is_a?(Array) => true
recipe.ingredients.first.amount => "8" # Not a Fixnum. Too hard. :

recipe.instructions.easy? => true
recipe.instructions.first.upcase => "MIX ALL INGREDIENTS TOGETHER."
recipe.instructions.steps.size => 7
require "xml-object"

`<recepie name="bread" title="an awesome recepie for bread">
<title>Bread Recepie</title>
</recepie>`

puts "At first, Elements are checked: " + test.title
puts "You can get the Attributes though: " + test[:attr => "title"]
Features: Adapter

1. require 'xml-object'     # REXML
2. require 'xml-object/adapters/hpricot'  # hpricot
3. require 'xml-object/adapters/libxml'  # libxml
Features: Question notation

1. `<admin>true</admin>`

1. `XMLFile.admin? => true`
Features: Collection auto folding

```xml
<student>
  <name>Bob</name>
  <course>Math</course>
  <course>German</course>
  <course>Biology</course>
</student>
```

```ruby
student = XMLObject.new(xml_file)
student.course.is_a?(Array)  #=> true
student.course.first == 'Math'  #=> true
student.course.last == 'Biology'  #=> true
```
Features: Collection pluralization

```xml
<student>
  <name>Bob</name>
  <course>Math</course>
  <course>German</course>
  <course>Biology</course>
</student>
```

```ruby
student = XMLObject.new(xml_file)
student.courses.first == student.course.first #=> true
```
Features: Collection proxy

```xml
<author>
  <name>John</name>
  <publications>
    <book>Math 101</book>
    <book>Biology 101</book>
  </publications>
</author>
```

```ruby
author.publications == author.publications.books #=> true
author.publications.map { |b| b.downcase } #=> ['math 101', 'biology 101']
```
Recursive
The design of the adapters assumes parsing of the objects recursively. Deep files are bound to throw SystemStackError, but for the kinds of files I need to read, things are working fine so far. In any case, stream parsing is on the TODO list.
XmlSimple
http://xml-simple.rubyforge.org/

a Ruby translation of Grant McLean's Perl module XML::Simple
XmlSimple = Rxml + ...

- xml_in()
- xml_out()
<config logdir="/var/log/foo/" debugfile="/tmp/foo.debug">
  <server name="sahara" osname="solaris" osversion="2.6">
    <address>10.0.0.101</address>
    <address>10.0.1.101</address>
  </server>
  <server name="gobi" osname="irix" osversion="6.5">
    <address>10.0.0.102</address>
  </server>
  <server name="kalahari" osname="linux" osversion="2.0.34">
    <address>10.0.0.103</address>
    <address>10.0.1.103</address>
  </server>
</config>
require 'xmlsimple'
config = XmlSimple.xml_in('foo.xml', { 'KeyAttr' => 'name' })
Our result: Hash.new

```ruby
{
    'logdir' => '/var/log/foo/',
    'debugfile' => '/tmp/foo.debug',
    'server' => {
        'sahara' => {
            'osversion' => '2.6',
            'osname' => 'solaris',
            'address' => [ '10.0.0.101', '10.0.1.101' ]
        },
        'gobi' => {
            'osversion' => '6.5',
            'osname' => 'irix',
            'address' => [ '10.0.0.102' ]
        },
        'kalahari' => {
            'osversion' => '2.0.34',
            'osname' => 'linux',
            'address' => [ '10.0.0.103', '10.0.1.103' ]
        }
    }
}
```

Usage: `puts config['server']['kalahari']['address'][1]`
my personal opinion

ONE DAY IN THE LIFE OF A PERL PROGRAMER

WHAT THE HACK IS THIS?

09:45 AM
READING THE CODE FROM THE PREVIOUS DAY
ROXML
http://roxml.rubyforge.org/
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>roxml_1.0_beta</td>
<td>roxml-1.0_beta</td>
<td>June 28, 2006</td>
</tr>
<tr>
<td>ROXML 1.0</td>
<td>roxml-1.0.zip</td>
<td>July 1, 2006</td>
</tr>
<tr>
<td>ROXML 1.1 Beta</td>
<td>ROXML 1.1 Beta</td>
<td>September 24, 2006</td>
</tr>
<tr>
<td>ROXML 1.2</td>
<td>ROXML 1.2</td>
<td>November 10, 2007</td>
</tr>
<tr>
<td>ROXML</td>
<td>2.2.0</td>
<td>November 3, 2008</td>
</tr>
</tbody>
</table>
ROXML Features

- Read Ruby objects from XML
- Write Ruby objects to XML
- Annotation-style methods for XML mapping
- One-to-one (composition) Ruby to XML
- One-to-many (aggregation) Ruby with array to XML

Source: http://roxml.rubyforge.org
It's all about annotations

```xml
<library>
  <NAME><![CDATA[Favorite Books]]></NAME>
  <books>
    <book ISBN='0201710897'>
      <title>The PickAxe</title>
      <description><![CDATA[Best Ruby book out there!]]></description>
      <author>David Thomas, Andrew Hunt, Dave Thomas</author>
    </book>
  </books>
</library>
```

```ruby
book = Book.new()
book.isbn = "0201710897"
book.title = "The PickAxe"
book.description = "Best Ruby book out there!"
book.author = "David Thomas, Andrew Hunt, Dave Thomas"

lib = Library.new()
lib.name = "Favorite Books"
lib.books << book
```
I/O

```ruby
#SAVE
File.open("library.xml", "w") do |f|
  lib.to_xml.write(f, 0)
end

#LOAD
lib = Library.parse(File.read("library.xml"))
```
XML::MAPPING

http://xml-mapping.rubyforge.org/

≈ROXML
class Client
  include XML::Mapping

  text_node :name, "Name"
  object_node :home_address, "Address[@where='home']", :class=>Address
  object_node :work_address, "Address[@where='work']", :class=>Address, :default_value=>nil
end

XML::Mapping::ArrayNode
XML::Mapping::BooleanNode
XML::Mapping::HashNode
XML::Mapping::NumericNode
XML::Mapping::ObjectNode
XML::Mapping::SingleNode
XML::Mapping::SubObjectBaseNode
XML::Mapping::TextNode

XML::Mapping::ChoiceNode
class Publication
  include XML::Mapping

choice_node :if,  '@author', :then, (text_node :author, '@author'),
  :elsif, 'contr', :then, (array_node :contributors, 'contr', :class=>String)
end

## usage

p1 = Publication.load_from_xml(REXML::Document.new('<publication author="Jim"/>').root)
=> #<Publication:0xb7ad3f38 @author="Jim">

p2 = Publication.load_from_xml(REXML::Document.new('</publication>
  <contr>Chris</contr>
  <contr>Mel</contr>
  <contr>Toby</contr>
</publication>').root)
=> #<Publication:0xb7ac7ee0 @contributors=["Chris", "Mel", "Toby"]>
has many
typecasts

code, code, code

xml = <<-EOF
<products>
  <product>
    <title> A Title</title>
    <features_bullets>
      <feature>This is feature text</feature>
      <feature>This is feature text</feature>
    </features_bullets>
  </product>
</products>
EOF

class FeatureBullet
  include HappyMapper
  tag 'features_bullets'
  element :feature, String
end

class Product
  include HappyMapper
  element :title, String
  has_many :features_bullets, Feature Bullet
end

Product.parse(xml).each do |product|
  puts product.title
  product.features_bullets.each do |fb| puts " - #{fb.feature}" end
end

# outputs:
# A Title
# - This is feature text
Camel Case XML Tags to Ruby method names

```ruby
element :total_pages, Integer, :tag => 'TotalPages'
```
twitter

1. `<statuses type="array">
2.   <status>
3.     <created_at>Sat Aug 09 05:38:12 +0000 2008</created_at>
4.     <id>882281424</id>
5.     <text>I so just thought the guy lighting the Olympic flag on the wall. Wow that would have been catastrophic.</text>
6.     <source>web</source>
7.     <truncated>false</truncated>
8.     <in_reply_to_status_id>1234</in_reply_to_status_id>
9.     <in_reply_to_user_id>12345</in_reply_to_user_id>
10.    <favorited>false</favorited>
11. </user>
12. </status>
13. </statuses>`

```ruby
1. class User
2.   include HappyMapper
3. end
4. class Status
5.   include HappyMapper
6. end
7. statuses = Status.parse(xml_string)
8. statuses.each do |status|
9.   puts status.user.name, status.user.screen_name
10. end
```
YAML
(rhymes with “camel”)
Design Goals

1. easily readable by humans.
2. matches the native data structures of agile languages.
3. portable between programming languages.
4. consistent model to support generic tools.
5. supports one-pass processing.
6. expressive and extensible.
7. easy to implement and use.
JSON/YAML = human readable data interchange format

- JSON == simplicity + universality
  - trivial to generate and parse
  - reduced human readability
- YAML == human readability + serializing native data structures
  - harder to generate and parse
  - easy to read
JSON ↔ YAML

JSON.valid? → YAML.valid!
...is Sequences, Maps, Scalars

- Seq = Array
- Map = Hash
- Scalars = String, Integer, Float, Time, NilClass
Sequences

Sequence:

YAML:
- apple
- burrito
- egg salad sandwich

Array:

Ruby:
['apple', 'burrito', 'egg salad sandwich']
Maps

Map:

YAML:

```
event: RubyConf.new(2002)
location: Seattle, WA, U.S.A.
start: Nov. 1st, 2002
end: Nov. 3rd, 2002
```

Hash:

```
{ 'event' => 'RubyConf.new(2002)',
  'location' => 'Seattle, WA, U.S.A.',
  'start' => 'Nov. 1st, 2002',
  'end' => 'Nov. 3rd, 2002'
}
```
Map of Scalars:

```
YAML:
integer: 12
float: 766.05
date: 2002-11-01
time: 2002-11-01T15:30:00.00Z
string: Begins with an alphabetic or numeric character.
single-quoted: '12'
double-quoted: "12"
```

Hash of Objects:

```
Ruby:
{ 'integer' => 12,
  'float' => 766.05,
  'date' => Date.new( 2002, 11, 01 ),
  'time' => Time.utc( 2002, 11, 01, 15, 30, 00, 00 ),
  'string' => 'Begins with an alphabetic or numeric character.',
  'single-quoted' => '12',
  'double-quoted' => '12'
}
```

Native typing is implicitly determined in plain scalars.
Can i haz code sample?
Input/Output

- YAML output:

```ruby
irb(main):003:0> require "yaml"
=> true
irb(main):004:0> ["Goik", "Kriha", "Schmitz", "Maucher"].to_yaml
=> "---
- Goik
- Kriha
- Schmitz
- Maucher
---"
irb(main):005:0> puts ["Goik", "Kriha", "Schmitz", "Maucher"].to_yaml
=> nil
```

- YAML input:

```ruby
irb(main):001:0> require "yaml"
=> true
irb(main):002:0> profs = YAML::load( File.open( 'profs.yml' ) )
=> ["Kriha", "Goik", "Maucher", "Schmitz"]
irb(main):003:0> profs.class
=> Array
```
Ruby and YAML

- More than 1 document

```
---
from: Marc
to: Audience
message: >
    Hallo, ich hoffe ihr seid noch wach :) ?
---
from: Audience
to: Marc
message: >
    Klar, bei dem super Vortrag!
```

- Ruby code

```ruby
YAML::load_documents( File.open( 'message.yml' ) ) { |msg| puts "A message from #{msg['from']} to #{msg['to']}:"
    puts msg['message']
}
```

- Output

```
A message from Marc to Audience: Hallo, ich hoffe ihr seid noch wach :) ?
A message from Audience to Marc: Klar, bei dem super Vortrag!
```
Won’t somebody please think of the children objects!
Live Demo :D
Problem: The !ruby/object type is only understood by YAML.rb.

Solution:

```ruby
require 'yaml'
require 'bigdecimal'

#Marshal
class BigDecimal
  def to_yaml(opts={})
    YAML::quick_emit(object_id, opts) do |out|
      out.scalar("tag:induktiv.at,2007:BigDecimal", self.to_s)
    end
  end
end

#Unmarshal
YAML.add_domain_type("induktiv.at,2007", "BigDecimal") { |type, val|
  BigDecimal.new(val)
}
```
YAML and JSON are simple and nice format for structured data and easier for human to read and write than XML. But there have been no schema for YAML such as RelaxNG or DTD. Kwalify gets over this situation.
Fragen?

KTHXBYE
Each project’s website

Some useful closures in Ruby
http://www.randomhacks.net/articles/2007/02/01/some-useful-closures-in-ruby

Kai Jäger: Ajax in der Praxis Grundlagen, Konzepte, Lösungen
ISBN-10: 3-540-69333-5

Using Ruby - An Introduction to Ruby for Java Programmers
http://onestepback.org/articles/usingruby/index.html

Ruby for Java Programmers

Happy Mapper: Making XML fun again:

YAML Working draft 1.2
http://yaml.org/spec/1.2/

YAML Cookbook:
http://www.nt.ntnu.no/users/haugwarb/Programming/YAML/YAML_for_ruby.html