Ruby & Cocoa

© Udo Borkowski, 2009

email: training@udo-borkowski.de

Agenda

- Ruby: An Introduction
- RubyCocoa, MacRuby, HotCocoa, ...

Ruby: An Introduction

Ruby: An Introduction

"Ruby is designed to make programmers happy"

Ruby: An Introduction

"an interpreted scripting language
for quick and easy
object-oriented programming"

Everything is an Object

Tools

- ruby
- irb

General Syntax

- *newline*
- \
- ;

- # single line comment
- =begin

  ...

  ... Multi Line Comment

  ...

  =end
- __END__

General Syntax

```
  puts "Hello World!";
puts "Hello World!"
puts "Hello" \
" World!";
puts "Hello" +
" World!";

puts "Hello"
+ " World!";  # SYNTAX ERR !
```

General Syntax

```
# OK, but not nice
a = 42; b = 15; c = a + b
```

```
# better
a = 42
b = 15
c = a + b
```

Names

fred anObject _x three_two_one	local variable (start with lowercase)
@name @_ @Size	instance variable (start with @)
@@name @@_ @@Size	class variable (start with @@)
`module` **Math** **PI** = 3.1415926 `end` `class` **BigBlob**	constant name (start with uppercase) *(Convention: constant variable only uppercase and underscore)*
$params $PROGRAM $! $_ $-a $-.	*global variables, and some special system variables (start with $)*
`def` **myMethod** `...` `end` `def` **SolutionProvider** `...` `end`	method names (start with lowercase or uppercase)

Reserved Words

__FILE__	and	def	end	in	or	self	unless
__LINE__	begin	defined?	ensure	module	redo	super	until
BEGIN	break	do	false	next	rescue	then	when
END	case	else	for	nil	retry	true	while
alias	class	elsif	if	not	return	undef	yield

Method Names

- Method names may end with ? or !
- Convention:
  - ? for boolean (Yes/No) methods. E.g. `Array#empty?`
  - ! for „dangerous“ methods
    - e.g. exit!
    - Signal object modifying methods.
      E.g.: `Array#reverse!` vs. `Array#reverse`
      (Attention: not all object modifying methods end with !)

Basic Types

- Numbers
- Strings
- Symbols
- Ranges
- Regexs
- Arrays
- Hashes

Numbers

Class Fixnum - Bignum - Float

Ruby 1.9

Class Complex

- 123
- -4_567
- 123_456_789_123_456_789
- 3.1415
- 5.678e-2
- 3 + 4.im

- 0x1234 (hex)
- 0b1001001001 (binary)
- 0777 (octal)
- ?c (char)
- ?\C-a (control a)
- ?\M-a (meta a)
- ?\M-\C-a (meta control a)

In Ruby 1.9 chars are strings!

Numbers

```
a = 2
b = 3
c = a + b          # 5
c = a - b          # -1
c = a / b          # 0
c = 2.0 / b        # 0.6666666666666667
c = a * b          # 6
c = a**b           # 2*2*2 = 8
a += 1             # a = 3
a -= 1             # a = 2
a++                # not supported
```

Numbers

```
1.upto(3) { |i| puts i }     # 1 2 3

3.downto(1) { |i| puts i }   # 3 2 1

0.step(10,2) { |i| puts i } # 0 2 4 6 8 10

3.times { puts "42" }        # 42 42 42
```

Strings

class String

- Multiple syntax forms
- Differ in:
  - Delimiter
  - Substitution Behaviour
  - (Shell) Command Interpretation

Strings

Name	Example	Substitution

Single Quoted	`'simple string'`
	
	`%q(simple string)`
		`\\` » `\`
		`\'` » `'`

Double Quoted	`"Hello #{name}!\nHow are you?"`
	
	`%Q(Hello #{name}!\nHow are you?)`
	
	`%(Hello #{name}!\nHow are you?)`
	
	`"The #{n+1}. member is null"`
		Backslashes
		`#{`*expr*`}`

Shell Commands	`` `ls -l #{dirpath}` ``
	
	`%x(ls -l #{dirpath})`
		Backslashes
		`#{`*expr*`}`
		Shell Command Execution

Strings

```
>> puts 'result: #{42.0/3} EUR'
=> result: #{42.0/3} EUR

>> puts "result: #{42.0/3} EUR"
=> result: 14.0 EUR
```

Backslashes

`\a`	bell
`\b`	backslash
`\e`	escape
`\f`	formfeed
`\n`	newline
`\r`	carriage return
`\s`	whitespace
`\t`	tab
`\v`	vertical tab

`\nnn`	octal
`\xnn`	hex
`\cx`	control x
`\C-x`	control x
`\M-x`	meta x
`\M-\C-x`	meta control x

`\x`	*x*

General Delimited Input

`%q(simple string)`	`%q`	Single Quoted String

`%Q(Hello #{name}!\nHow are you?)`
 	
`%(Hello #{name}!\nHow are you?)`
	`%Q`
	`%`
		Double Quoted String

`%x(ls -l #{dirpath})`	`%x`	Shell Command
`%w(...)`	`%w`	Array of tokens
`%W(...)`	`%W`	Array of tokens (with substitution)
`%r(...)`	`%r`	Regular Expression

May use %.[...], %.{...}, %.<...>
or any other chars %c...c
instead of %.(...)

E.g.:

```
%q/this is a string/
%q-string-
%q{a {nested} string}
```

Other String Details

- << or + for Concatenation

```
'Con' + "cat" + 'en' + "ate"
```

```
s = 'Con'
s << "cat"
s << 'en'
s << "ate"
```

- Auto String Concatenation

```
'Con' "cat" 'en' "ate"
```

```
Concatenate
```

- Multi Line Support

- == to compare Strings content, equals? for identify check

Strings

```
>> s = "concatenation"

>> s[3]
=> 99
>> s[3].chr
=> "c"
>> s[7,6]
=> "nation"
>> s[3..5]
=> "cat"

>> "="*20
=> "=========================="
```

Ruby 1.9 => "c"

Strings

```
>> s = "concatenation"

>> s.delete("a")
=> "conctention"
>> s.delete("aeiou")
=> "cnctntn"
>> s.gsub("catena","ven")
=> "convention"
>> s.gsub(/[aeiou]/,"$")
=> "c$nc$t$n$t$$n"
>> s.index('a')
=> 4
>> s.index('a',5)
=> 8
```

Strings

```
>> "123".to_i
=> 123
>> nil.to_i
=> 0
>> "123abc".to_i
=> 123
>> Integer("123")
=> 123
>> Integer(nil)
=> 0
>> Integer("123abc")
ArgumentError: ...
```

Strings

Here Documents

```
a = 123
print <<HERE
Double quoted \
here document.
Sum = #{a + 1}
HERE

print <<-'THERE'
    This is single quoted.
    The above used #{a + 1}
    THERE
```

```
Double quoted here document.
Sum = 124
    This is single quoted.
    The above used #{a + 1}
```

```
def create(project, user)
  ...
end

update(<<PROJECT_XML, current_user)
  <project>
    <owner>#{project.owner}</owner>
    <manager>#{project.manager}</manager>
  </project>
PROJECT_XML
```

Symbols

Class Symbol

```
:Object
:myVariable
:"Hello world"
```

```
>> :Hello.object_id == :Hello.object_id
=> true
```

```
>> "Hello".object_id == "Hello".object_id
=> false
```

```
colors = { :red   => 0xf00,
           :green => 0x0f0,
           :blue  => 0x00f
         }
```

Ranges

Class Range

- expr .. expr (inclusive range)
- expr ... expr (last element excluded)
- === to check for value in range

Ranges

```
inclusive = (1..10)     # 1, 2, 3, ..., 10
exclusive = (1...10)    # 1, 2, 3, ..., 9
letters = ('a'..'z')    # a, b, c, ..., z
words = ('aa'..'bz')    # aa, ab, ac, ...
                        #     bx, by, bz
values = -12.3..+12.3

letters === 'x'         # true
letters === '@'         # false

values === -1.23        # true
values === 123          # false
```

Ranges

```
for i in (1..10) do
  puts i
end
```

```
while c = gets
  case c
  when '0'..'9' then puts "digit"
  when 'a'..'z' then puts "letter"
  end
end
```

```
a = [1, 2, 3, 4, 5, 6]
puts a[1..3]                    # 2 3 4
puts a[1...3]                   # 2 3
```

Arrays

Class Array

```
arr = [ fred, 10, 3.14, "This is a string", barney("pebbles"), ]
```

(Trailing comma ignored)

```
arr = %w( fred wilma barney betty great\ gazoo )
```

```
["fred", "wilma", "barney", "betty", "great gazoo"]
```

```
arr = Array.new
arr << 123
arr << "PI"
arr << nil << 3.14159 << "Done"
```

```
[123, "PI", nil, 3.14159, "Done"]
```

```
arr = []
arr[0] = 13
arr[2] = 15
```

```
[13, nil, 15]
```

Arrays

```
numbers = [1, 7, 12, 42]
```

```
numbers[0]   # 1
numbers[3]   # 42
numbers[4]   # nil
numbers[-1] # 42
numbers[-2] # 12
```

```
numbers[0,2]   # [1, 7]
numbers[2,1]   # [12]
numbers[2,0]   # []
numbers[-2,2] # [12, 42]
```

```
numbers[0..1]    # [1, 7]
numbers[0..2]    # [1, 7, 12]
numbers[0..10]   # [1, 7, 12, 42]
numbers[-3..-1] # [7, 12, 42]
```

Arrays

```
a = [1,2,3,4,5,6]
```

```
a[2] = [3,2,1]
```

```
[1, 2, [3, 2, 1], 4, 5, 6]
```

```
a[1..4] = [0,0,0]
```

```
[1, 0, 0, 0, 6]
```

```
a[1..4] = 0
```

```
[1, 0, 6]
```

```
a[2,0] = [7,8]
```

```
[1, 2, 7, 8, 3, 4, 5, 6]
```

Hashes

Class Hash

```
colors = { "red"    => 0xf00,
           "green"  => 0x0f0,
           "blue"   => 0x00f
         }
```

```
colors = Hash.new
colors[:red] = 0xf00
colors[:green] = 0x0f0
colors[:blue] = 0x00f
```

Ruby 1.9

```
colors = { red:0xf00, green:0x0f0, blue:0x00f}
```

Hashes

```
hash = { "x" => 199,
         123 => [1,2,3],
         :z => { "a"=>1, "b"=>2 }
       }
```

`hash["x"]`	`199`
`hash[123]`	`[1,2,3]`
`hash[:z]`	`{ "a"=>1, "b"=>2 }`
`hash[:z]["b"]`	`2`
`hash["a"]`	`nil`

Regular Expressions

Class Regexp

```
/pattern/
/pattern/options
%r{pattern}
%r{pattern}options
Regexp.new( 'pattern' [, options ] )
```

Options	
`i`	Case Insensitive
`o`	Substitute #{...} Once
`m`	Multiline Mode
`x`	Extended Mode

(For pattern syntax see documentation)

Conditions / Booleans

Everything is `true` except `false` and `nil`

&& (and) || (or) ! (not)

Lazy Evaluation of logical operators, last value returned

Example:

```
n = params[:name] || "unknown"
```

instead of

```
if params[:name]
  n = params[:name]
else
  n = "unknown"
end
```

if

```
if ...
   ...
elseif ...
   ...
else
   ...
end
```

```
if ... then ...
elseif ... then ...
else ...
end
```

```
if ... : ...
elseif ... : ...
else ...
end
```

```
x = 1 if x < 1
x = 10 if x > 10
```

```
mx = if x1 > x2 then x1; else x2; end
```

unless

```
unless ...
  ...
else
  ...
end
```

```
if !...
  ...
else
  ...
end
```

Also other syntax variations, as with `if`, e.g.:

```
x = 1 unless x >= 1
```

Ternary Operator

```
mx = x1 > x2 ? x1 : x2
```

case

```
case s
when "Hello" : puts "world"
when "Hallo" : puts "Welt"
else puts "mondo"
end
```

```
case c
when '0'..'9' then puts "digit"
when 'a'..'z' then puts "letter"
else puts "unknown character ${c}"
end
```

```
case c
when 'a'..'z', 'A'..`Z' then puts "letter"
else puts "unknown character ${c}"
end
```

case

```
x = case c
    when '0'..'9' : "digit"
    when 'a'..'z' : "letter"
    else "unknown character #{c}"
    end
```

Loops

- For
- While
- Until

for

```
for i in 1..10 do
  puts i
end
```

```
for i in [123, "Hello World", 1.89]
  puts i
end
```

```
for key,value in { :yes=> "Ja", :no=>"Nein"}
  puts "#{key} => #{value}"
end
```

while / until

```
while ... do
  ...
end
```

```
until ... do
  ...
end
```

```
i += 1 while i < 10
j -= 1 until i < 0
```

break etc.

`break`	exit loop
`next`	continue with next iteration
`redo`	redo current iteration
`retry`	restart all iteration (from the beginning)

Iteration

```
a_list.each do |item|
  # use item to access „current item“
  # in inner block
end
```

Some Enumerable Providers	
`each`	`find`
`collect`	`inject`
`map`	`sort`

Methods

```
def method_name(param1, param2,...)
  ...
  return x
end
```

```
def add x, y
  x + y
end
```

```
puts add(1, 2)
```

```
3
```

```
puts add 1, 2
# warning: parenthesize argument(s) for future version
```

```
3
```

Methods

```
def method1(a, b=1) ...
def method2(a, b=1, c=2) ...
def method3(a, b=1, c) ... # Error!

method1(1)
method1(1,2)
```

```
def foo(*args) ...
def bar(format,*args) ...

foo(1,2,3,4,5)
bar("...",1,2.34,"hello")
```

Methods

```
def pot(x)
  return x, x*x, x*x*x
end

x,x2,x3 = pot(4)
```

```
x == 4
x2 == 16
x3 == 64
```

```
h       = "12:34:56".split(':') # ["12", "34", "56"]
h,      = "12:34:56".split(':') # "12"
h, m    = "12:34:56".split(':') # "12", "34"
h, m, s = "12:34:56".split(':') # "12", "34", "56"
```

Methods

Blocks

```
method_call {
  ...
}
```

```
method_call do
  ...
end
```

```
def threeTimes
  yield
  yield
  yield
end
```

```
threeTimes { puts "Hello" }

threeTimes do
  puts "Hello"
end
```

```
Hello
Hello
Hello
```

```
def threeTimes
  yield 1
  yield 2
  yield 3
end
```

```
threeTimes { |i| puts "#{i}. Hello" }

threeTimes do |i|
  puts "#{i}. Hello"
end
```

```
1. Hello
2. Hello
3. Hello
```

Methods

Blocks

```
class Array
  def find
    for i in 0...size
      value = self[i]
      return value if yield(value)
    end
    return nil
  end
end
```

```
[1, 3, 5, 7, 9].find {|v| v*v > 30 }
```

7

```
def method(param,&block)
  ...
  block.call(...)
  ...
end
```

```
block = Proc.new { |s| puts s }
[1,2,3].each &block
```

```
block = lambda { |s| puts s }
[1,2,3].each &block
```

Ruby 1.9 (Lambda Shorthand)

```
p = -> a,b,c {a+b+c}
puts p.(1,2,3)
puts p[1,2,3]
```

Methods

block_given?

```
def dump(array, &block)
  unless block_given?
    block = Proc.new { |x| puts x }
  end
  array.each &block
end

a = [65, 66, 67]
dump(a)                          # 65 66 67
dump(a) { |x| puts x.chr }       # A B C
```

Classes

```
class MyOwnClass
  def initialize(param1,param2)
    ...
  end
  ...
  # other methods
  ...
end
```

```
obj = MyOwnClass.new(123, "hello")
```

Classes

Instance Variables & Methods

```
class InstanceVariableDemo
  @no_instance_var = 0 # !NO instance var

  def start
    # create instance var @start_time
    @start_time = Time::now
  end

  def stop
    @delay = Time::now - @start_time
  end
end
```

Classes

self

```
class SelfDemo
  def start
    self.@start_time = Time::now
  end

  def stop
    self.@delay = Time::now - self.@start_time
  end
end
```

Classes

Variable Access

```
class SampleClass
  def initialize
    @count = 0
    @name = ""
  end

  def name
    @name
  end

  def name=(n)
    @name = n
  end
end
```

```
class SampleClass
  attr_reader :name
  attr_writer :name

  ...
end
```

```
class SampleClass
  attr :name, true

  ...
end
```

```
class SampleClass
  attr_accessor :name

  ...
end
```

Classes
Visibility

- public (default)
- private (receiver is always self)
- protected (receiver is instance of defining class)

```
class MyClass
  def a_public_method; true; end

  private

  def a_private_method; true; end
  def another_private_method; false; end

  public

  def another_public_method; false; end
end
```

Classes

Visibility

```
class Project
  def method1
  end

  def method2
  end

  def method3
  end

  def method4
  end

  protected :method2, :method3
  private :method4
end
```

Classes

Class Variables & Methods

```
class ObjectCounter
  @@counter = 0

  def initialize(name)
    @@counter += 1
    @name = name
  end

  def ObjectCounter.counter
    @@counter
  end

  def self.log(s)
    puts "Log ObjectCounter: #{s}"
  end

  log "Started"
end
```

Classes

Inheritance

```
class MySuperClass
  def to_s
    "Super"
  end
end

class MyClass < MySuperClass
  def to_s
    s = super
    "MyClass < #{s}"
  end
end
```

Classes

Inheritance & Visibility

```
class MyClass < MySuperClass
   ...
  public :somePrivateMethod
end
```

Classes

Class Object

- `obj.object_id`
- `obj.class`
- `obj.instance_of`
- `obj.kind_of`
- `obj.to_s`
- `obj.inspect`
- `obj.==`
- `obj.eql?`
- `obj.equal?`
- `obj.nil?`
- `obj.===`

Classes

`method_missing`

```
def method_missing( id, *arguments )
  puts "The method #{id} was not found."
  puts "Arguments used: " +
    arguments.join(", ") + "."
end
```

Classes

send

```
1 + 2

# may also written as
1.+(2)

# or written as
1.send "+", 2
```

Classes

Extending Classes

```
class Fixnum
  def hours
    self * 3600  # number of seconds in an hour
  end
end

Time.mktime(2006, 01, 01) + 14.hours
#-> Sun Jan 01 14:00:00
```

Classes

Singleton Methods

```
class Car
  def inspect
    "cheap"
  end
end

porsche = Car.new
porsche.inspect  #-> cheap
```

```
def porsche.inspect
  "expensive"
end
porsche.inspect  #-> expensive
```

```
car = Car.new
car.inspect  #-> cheap
```

Classes
Override Operators

```
class Fixnum
  # You can, but please don't do this
  def +( other )
    self - other
  end
end
```

- array style access: [] and []=
- unary +, - (e.g. -x): +@, -@
- not all operators can be redefined
  - =, .., ..., !, not, &&, and, ||, or, !=, !~, ::
  - +=, *= etc. (abbreviations, redefine +, * etc.)

Modules

```
module StringUtil
  DEFAULT_SEPARATOR = ','

  def StringUtil.arrayToString(array)
    ...
  end
  ...
end

a = [1, 2, 3]
s = StringUtil.arrayToString(a) # => "[1,2,3]"

puts StringUtil::DEFAULT_SEPARATOR
```

Modules

Classes in Module

```
module XML
  class Document
  end

  class Tag
  end

  class Attribute
  end
end

t = XML::Tag.new("<person>")
a = XML::Attribute.new("name","John")
```

Modules

Mixin

```
module LogUtil
  def log(msg)
    ...
  end
end
```

```
class Person
  include LogUtil

  def initialize(name)
    log("Person #{name} created")
  end
end
```

Modules

Mixin

```
class Song
  include Comparable

  def <=>(other)
    self.duration <=> other.duration
  end
end
```

```
song1 = Song.new("My Way",  "Sinatra", 225)
song2 = Song.new("Bicylops", "Fleck",  260)

song1 <=> song2 # -1
song1  <  song2 # true
song1 ==  song1 # true
song1  >  song2 # false
```

Modules

Mixin

```
module Inject
  def inject(n)
    each do |value|
      n = yield(n, value)
    end
    n
  end

  def sum(initial = 0)
    inject(initial) {
      |n, value| n + value
    }
  end
  def product(initial = 1)
    inject(initial) {
      |n, value| n * value
    }
  end
end
```

```
class Array
  include Inject
end

[ 1, 2, 3, 4, 5 ].sum      # 15
[ 1, 2, 3, 4, 5 ].product  # 120
```

```
class Range
  include Inject
end

(1..5).sum          # 15
(1..5).product      # 120

('a'..'m').sum("Letters: ")
          # "Letters: abcdefghijklm"
```

Modules
module_function

```
module LogUtil
  def log(msg)
    ...
  end

  module_function :log
end
```

```
# called as instance method
include LogUtil
log("...")

# called as module method
LogUtil.log("...")
```

Exception

class Exception

```
raise IOError, "File not found"
raise "Too many recursions"  # use RuntimeError
```

```
begin
   ... # may raise here
rescue IOError => ex
   puts ex.message
   puts ex.backtrace
   raise ex
end
```

```
def some_method
   ...
rescue IOError => ex
   ...
end
```

```
123 / 0 rescue puts "div by zero demo"
```

Exception

```
begin
  ...
  rescue EOFError
    ...
  rescue IOError
    ...
  rescue
    ... # catch any remaining StandardError
end
```

```
def read_config
  f = nil
  begin
    f = File.open("config.cfg")
    ...
  rescue
    puts "Error: #{$!}"
  else
    ...
  ensure
    f.close if f
  end
end
```

Ruby Stuff Not Covered

- Processes and Thread
- Reflection / Metaprogramming
- IO
- ...

Ruby & Cocoa

Ruby & Cocoa

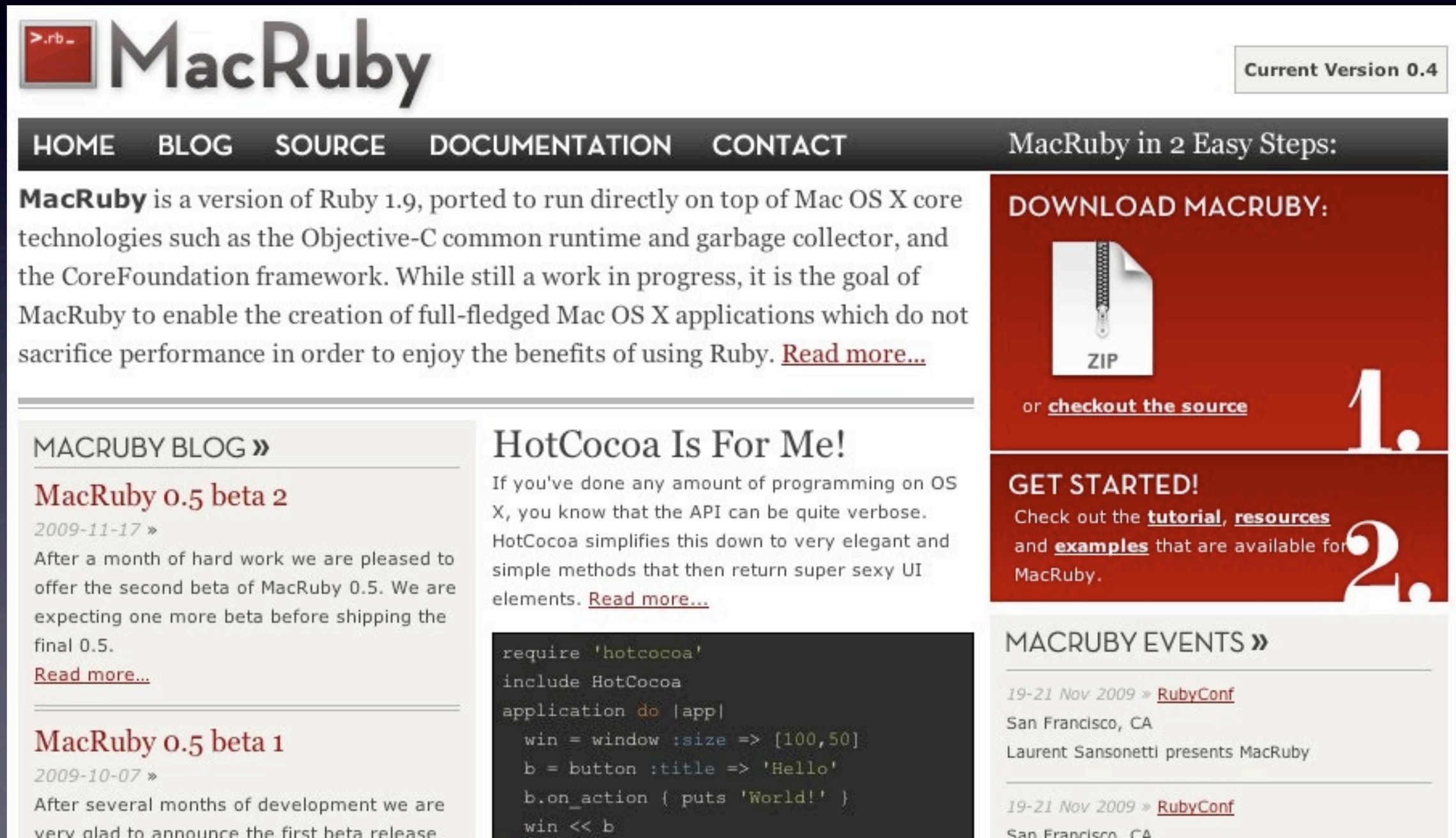
- RubyCocoa
- MacRuby
- HotCocoa

RubyCocoa

http://rubycocoa.sourceforge.net/

MacRuby

http://www.macruby.org/



MacRuby

Objective-C

String	NSMutableString
Number	NSNumber
Array	NSMutableArray
Hash	NSMutableDictionary
Object	NSObject

MacRuby

Objective-C

```
[text stringByReplacingOccurrencesOfString:@"Ruby"
                                withString:@"MacRuby"]
```

MacRuby

```
text.stringByReplacingOccurrencesOfString("Ruby",
                                withString:"MacRuby")
```

MacRuby

```
-(void) tableView: (NSTableView*)aTableView
  willDisplayCell: (id) aCell
   forTableColummn: (NSTableColumn*)aTableColumn
              row: (NSInteger)  rowIndex
```

MacRuby

```
      tableView:                    aTableView
willDisplayCell:      aCell
forTableColummn:                      aTableColumn
            row:                    rowIndex
```

MacRuby

```
def tableView(aTableView,
              willDisplayCell: aCell,
              forTableColummn: aTableColumn,
              row: rowIndex)
  # some code
end
```

Demo

HotCocoa

HotCocoa

```
require 'hotcocoa'

include HotCocoa

application do |app|
  win = window :size => [100,50]
  b = button :title => 'Hello'
  b.on_action { puts 'World!' }
  win << b
end
```

References

- Ruby Website - http://www.ruby-lang.org
- The Ruby Programming Language (Taschenbuch) von David Flanagan, Yukihiro Matsumoto
- Programming Ruby: The Pragmatic Programmer's Guide http://www.ruby-doc.org/docs/ProgrammingRuby/
- Ruby From Other Languages (C, C++, Java, Perl, PHP, Python) http://www.ruby-lang.org/de/documentation/ruby-from-other-languages/
- Ruby Grundlagen - http://www.b-simple.de/download/ruby.pdf
- MacRuby - http://www.macruby.org

Thank You

© Udo Borkowski, 2009

email: training@udo-borkowski.de

Creative Commons Attribution-Share Alike 3.0 Unported License