

# **Ruby Reports (rreport)**

**Making reporting as fun as it can be.**

**Presented by Gregory Brown  
as part of the GSoC Panel, RubyConf 2006**

**What is Reporting, Anyway?**

**Is reporting fun?**

**'... I urge you to at least download this report and peek inside. It's a tragic example of database output gone wrong.'**

**- JEG2**

## **Ruby Quiz #17: To Excel**

**' Sorry for sending in a quiz that "looked too much like real work". I promise, we're back to fun and games tomorrow...'**

**- JEG2**

**Without external motivation,  
writing reporting software  
simply SUCKS.**

**Programs like GSoC provide  
motivation and resources  
necessary to make**

*'stuff that looks too much like work'*

**come together.**

**But why is Rupert important?**

**Web** Results **1 - 10** of about **1,280,000** for "[Reporting Software](#)". (0.34 seconds)

**Reporting is a MASSIVE domain.**

**Web** Results **1 - 10** of about **2,530,000** for "[Reporting Tools](#)". (0.36 seconds)

## RAA - Search

3 projects found

<a href="#">datavision</a>	<a href="#">Application/Database</a>	0.6.0	<a href="#">jim Menard</a>	mature	2003-01-29T04:28:55Z
database reporting tool: [DataVision is now a Java project ( <a href="http://datavision.sourceforge.net">http://datavision.sourceforge.net</a> ) that will start using J..					
<a href="#">report-bug</a>	<a href="#">Application/Development</a>	0.2.0	<a href="#">MoonWolf</a>	alpha	2004-11-10T15:56:26Z
Bug Reporting Utility: Bug Report Utlity How to use: \$ ruby -report-bug <your-application.rb>					
<a href="#">rreport</a>	<a href="#">Library/Reporting</a>	0.5.4	<a href="#">Gregory Brown</a>	beta	<u>2006-09-25T03:54:57Z</u>
Ruby Report Generation Framework: Ruby Reports (Ruport) is a free software library and toolset that is designed to mak..					

**Ruby Reports is the only active generalized reporting project in Ruby.**

**PDF::Writer**

**RubyDBI**

**ActiveRecord**

**FasterCSV**

**RedCloth**

**XML::Builder**

# **Ruby has the raw materials**

**Scruffy**

**Gruff**

**Standard Library**

**Mechanize**

**Markaby**

**Sparklines**

# Database Interaction

```
DBI.connect('dbi:mysql:foo', 'root', '') do | dbh |  
  @results = dbh.select_all('select * from chunky_bacon')  
end
```

**Via Ruby DBI**

# CSV Generation

```
csv_output = FasterCSV.open("results.csv", "w") do |csv|  
  @results.each { |r| csv << r }  
end
```

Via FasterCSV

# HTML Generation

```
builder = Builder::XmlMarkup.new(:indent => 2)

html_output = builder.html do |html|
  html.table do |html_table|
    @results.each do |row|
      html_table.tr do |html_row|
        row.each { |e| html_row.td(e) }
      end
    end
  end
end
```

**Via XML Builder**

# Email Creation and Delivery

```
File.open("results.html","w") { |f| f << html_output }

mail = MailFactory.new
mail.to = "test@test.com"
mail.from = "sender@sender.com"
mail.subject = "Here are some files for you!"
mail.text = "Attached are the CSV and HTML files"
mail.attach("results.csv")
mail.attach("results.html")

Net::SMTP.start( 'smtp1.testmailer.com', 25, 'mail.from.domain',
                 mail.from, password, :cram_md5) do |smtp|
  smtp.send_message(mail.to_s(), mail.from, mail.to)
end
```

**Via MailFactory +  
Net::SMTP**

**The real problem  
is integration**

```
begin; require 'rubygems'; rescue LoadError; nil; end
%w[dbi builder fastercsv net/smtp mailfactory].each { |l| require l }

DBI.connect('dbi:mysql:foo', 'root', '') do |dbh|
  @results = dbh.select_all('select * from chunky_bacon')
end

csv_output = FasterCSV.open("results.csv","w") do |csv|
  @results.each { |r| csv << r }
end

builder = Builder::XmlMarkup.new(:indent => 2)

html_output = builder.html do |html|
  html.table do |html_table|
    @results.each do |row|
      html_table.tr do |html_row|
        row.each { |e| html_row.td(e) }
      end
    end
  end
end

File.open("results.html","w") { |f| f << html_output }

mail = MailFactory.new
mail.to = "test@test.com"
mail.from = "sender@sender.com"
mail.subject = "Here are some files for you!"
mail.text = "Attached are the CSV and HTML files"
mail.attach("results.csv")
mail.attach("results.html")

Net::SMTP.start( 'smtp1.testmailer.com', 25, 'mail.from.domain',
                 mail.from, password, :cram_md5) do |smtp|
  smtp.send_message(mail.to_s(), mail.from, mail.to)
end
```

```
class SimpleReport < Ruport::Report

  prepare do

    source :default,
      :dsn => "dbi:mysql:foo",
      :user => "root"

    mailer :default,
      :host => 'smtp1.testmailer.com',
      :address => 'sender@sender.com'

    @table = query('select * from chunky_bacon')

  end

  generate do
    write "results.html", @table.to_html
    write "results.csv", @table.to_csv
  end

end

SimpleReport.run do |report|
  report.send_to("test@test.com") do |mail|
    mail.subject = "Here are some file for you!"
    mail.text = "Attached are the CSV and HTML files"
    mail.attach("results.csv")
    mail.attach("results.html")
  end
end
```



**What are the  
core concepts of  
reporting?**

Fetching

Mu n g i n g

Presenting

Producing

# The Goal:

Provide a system that covers  
the core concepts of  
reporting while meeting our  
unique needs as Rubyists

# The Needs

**(1)**

**The ability to play nice  
with others**

**acts\_as\_reportable**

**Allow ActiveRecord  
to be used as a data  
source for Report**

```
require "ruport"

class SimpleReport < Ruport::Report

  prepare do

    source :default,
      :dsn => "dbi:mysql:foo",
      :user => "root",

    mailer :default,
      :host => 'smtp1.testmailer.com'
      :address => 'sender@sender.com'

    @table = query('select * from chunky_bacon')

  end

  generate do
    write "results.html", @table.to_html
    write "results.csv", @table.to_csv
  end

end

SimpleReport.run do |report|
  report.send_to("test@test.com") do |mail|
    mail.subject = "Here are some file for you!"
    mail.text = "Attached are the CSV and HTML files"
    mail.attach("results.csv")
    mail.attach("results.html")
  end
end
```

```
@table = query('select * from chunky_bacon')
```



```
class ChunkyBacon < ActiveRecord::Base
  acts_as_reportable
end

...

@table = ChunkyBacon.report_table(:all)
```

```
@table = query("select * from chunky_bacon where name = 'greg'")
@table2 = query("select foo,bar,baz from chunky_bacon")
```



```
@table = ChunkyBacon.report_table(:all, :conditions => ['name = ?', 'greg'])
@table2 = ChunkyBacon.report_table(:all, :columns = %w[foo bar baz])
```

**Allow Rails users to  
leverage Rupture in their  
Rails applications.**

```
# in config/environment.rb
require "ruport"

class ActiveRecord::Base
  def self.to_csv
    data_set = Ruport::DataSet.new(self.column_names)
    self.find_all.each do |row|
      data_set << row.attributes
    end
    data_set.to_csv
  end
end

#-----
# >> Contact.to_csv #=> A string of CSV data
```

Rails to Excel - QuirkeyBlog

```
class ActiveRecord::Base

  acts_as_reportable

  def self.to_csv
    report_table(:all).to_csv
  end

end
```

Using `acts_as_reportable`  
and `Ruport 0.5.x`

**(2)**

**Access to the  
sharpest knives in  
the drawer**

**The need for  
extension is  
inevitable**

# A Unique Formatting System

```
module Ruport
  class Format::Engine
    class Graph < Format::Engine

      attributes [:width, :height, :style, :title]

      renderer do
        super
        active_plugin.render_graph
      end

      alias_engine Graph, :graph_engine
      Ruport::Format.build_interface_for Graph, :graph

    end
  end
end
```

**Engines cover concepts and processes**

```

module Ruport
  class Format::Plugin
    class XmlSwfPlugin < Format::Plugin

      helper(:init_plugin) { |eng|
        @chart_type = eng.style.to_s || "bar"
      }

      renderer :graph do

        require_gem "builder"
        builder = Builder::XmlMarkup.new(:indent => 2)
        builder.chart do |b|
          b.chart_type(@chart_type)
          b.chart_data do |cd|
            cd.row { |first|
              first.null
              data.column_names.each { |c| first.string(c) }
            }
            data.each_with_index { |r,i|
              label = r.tags[0] || "Region #{i}"
              cd.row { |data_row|
                data_row.string(label)
                r.each { |e| data_row.number(e) }
              }
            }
          }
        end
      end
    end

    plugin_name :xml_swf
    register_on :graph_engine

  end
end

```

# Plugins cover presentation

```
# Start with a Ruport::Table object. This could easily come from
# activerecord or any of the other ways to build a Table. See the ruport
# recipes book for some ideas
data = [[5, 7, 9, 12, 14, 16, 18]].to_table(%w[jan feb mar apr may jun jul])

# initialize the graph with our table object
graph = Ruport::Format.graph_object :plugin => :svg, :data => data

# there are currently only a handful of options for customising the
# appearance of the graph. The ones listed here are all of them at
# the current time.
graph.width = 700
graph.height = 500
graph.title = "A Simple Line Graph"
graph.style = :line # other options: bar, smiles, area, stacked

# render the graph and print it to stdout. To save the output to a file, try
# ruby line_graph.rb > pirates.svg
puts graph.render
```

**You get a simple interface for free!**

(3)

A way to keep  
things DRY

```

class SimpleReport < Ruport::Report

  prepare do

    source :default,
      :dsn => "dbi:mysql:foo",
      :user => "root"

    mailer :default,
      :host => 'smtp1.testmailer.com',
      :address => 'sender@sender.com'

    @table = query('select * from chunky_bacon')

  end

  generate do
    write "results.html", @table.to_html
    write "results.csv", @table.to_csv
  end

end

SimpleReport.run do |report|
  report.send_to("test@test.com") do |mail|
    mail.subject = "Here are some file for you!"
    mail.text = "Attached are the CSV and HTML files"
    mail.attach("results.csv")
    mail.attach("results.html")
  end
end

```

```
class SimpleReportTextUI < SimpleReport
  generate do
    @table.to_text
  end
end

SimpleReportTextUI.run { |r| puts r }
```

**Write as little code as possible**

```
class SimpleReportTextUI < SimpleReport

  attr_accessor :header

  generate do
    header + @table.to_s
  end
end

a = SimpleReportTextUI.new
b = SimpleReportTextUI.new

a.header = "foo"
b.header = "bar"

SimpleReportTextUI.run(:reports => [a,b]) { |r| puts r }
```

**Reuse processes whenever possible**

**DEMO**

# Concrete products of these needs produced during GSoC

- `acts_as_reportable` provides hooks into `Ruport`  
→ from `ActiveRecord` and `Rails`.
- A powerful formatting system was developed,  
→ allowing people to extend `Ruport` for their own needs.
- A Simple DSL for producing Reports provided a DRY  
→ way to generate and deliver reports as needed.
- Datastructures tailored for tabular reporting provided a solid base  
→ for manipulations in `Ruport`.
- Community contributions for creating SVG Graphs and PDF Invoices

[report.infogami.com](http://report.infogami.com)

**QUESTIONS?**