

Introduction

This paper briefly describes the use of Ruby and Ruby on Rails (RoR) as a supplement to Oracle Application Express (Apex). It is a companion to a presentation that I gave at the Oracle User Group Norway (OUGN) spring seminar on Color Magic April 14 2010.

It is assumed that the reader is familiar with Apex and the Oracle database.

The following topics will be covered:

- Why do we need a supplement to Apex?
- Why Ruby and Rails (and friends)?
- Examples of Ruby in action
- Notes on deployment and integration

Why do we need a supplement to Apex?

Apex, is a great tool. It is a fully integrated development environment based on the robust Oracle stack and it uses PL/SQL in very efficient and DBA friendly manner. This makes Apex perfect for small development teams, giving them many of the same advantages as for example Java and PHP teams. That is: the same language is used for both back-end and front-end development. This facilitates Agile approaches and results in productivity that rivals that of popular and “sexy” web frameworks, such as:

- Ruby on Rails
- Zend (PHP)
- CodeIgniter / CakePHP (both PHP)
- ASP.NET

When is Apex not a good fit?

The general rule is always: Work with the tool, not against it. We have all heard of, and sometimes even experienced, how someone developed (old style) PHP in PL/SQL or heard someone say “I can do anything in FORTRAN”.

Now try this in Apex:

- Get a file from a ftp site, stream the contents into a blob in the database
- Take a file that someone uploaded to the database and stream it Amazon S3
- Periodically log on to a server and retrieve files
- For all files in a directory, create a SQL*Loader (sqlldr) control file on the fly, then execute sqlldr
- Retrieve emails from a mail server; for each email: parse the message, look for patterns to trigger text messages; store attachments in the database

Trying to to solve these type of challenges with Apex is like forcing a square peg into a round hole: a very bad idea. You will most likely end up with a hodgepodge of shell scripts, dbms_scheduler jobs and sqlplus scripts. The solution will be brittle and difficult to maintain. Finally, your code will be platform specific. This may not be a big deal if you represent a corporation that has standardized on a specific hardware and software combination. It is a big deal if you are a consultant.

Warning!

- Learning a new language and a new framework can be addictive
- You will get a different perspective on things
- You will probably become a better developer
 - Understanding of general concepts (what is really “templating”)
 - Different view of concepts in your “native” language (exception handling, C# vs PL/SQL)
 - Easier to evaluate new language features in you “native” language (closures in Ruby, soon in Java 7)
- Your respect for other languages will increase, making it easier to communicate with (for example) “those web dudes”
- It will most likely be fun

You can read more about this in [fowler].

I am assuming that you are familiar with a programming language, such as PL/SQL.

Why Ruby and Rails (and friends)?

Dynamic languages are on roll. Even in the Enterprise, more and more developers are turning to dynamic languages such as PHP and Perl. Based on [\[matt-asay\]](#) we can conclude that the demand for .Net and Java developers is fairly stable. At the same time, there is distinct growth in the demand for PHP and Perl developers. A simple search on [\[indeed.com\]](#) shows that the demand for Ruby and Rails expertise is growing much faster than for its dynamic brethren.

I chose Ruby for the following reasons:

- Strong database connectivity
- Solid platform support
- Great communications libraries: ftp, ssh, mail, etc
- Lots of extras (AKA gems): S3, ImageMagick, etc

Ruby

According to [\[wiki-ruby\]](#) “Ruby is a dynamic, reflective, general purpose object-oriented programming language that combines syntax inspired by Perl with Smalltalk-like features.”

Ruby was conceived in Japan during the 1990s and it was initially designed by Yukihiro “Matz” Matsumoto. Ruby has a C like syntax. The standard 1.8.7 implementation is written in C, as a single-pass interpreted language. It runs in a VM and is generally available on “all” platforms: Linux, Mac, Windows and Java.

A quick example of Ruby vs traditional scripting

The following code is taken from a wrapper script that I wrote to execute rman (Oracle Recovery Manager) commands. We use it on all the Oracle systems that we manage.

The few lines shown here identifies the directory portion of the calling script. This information is used to identify the log directory. If the binary is called c:\orabackup\bat\orabackup.bat, the directory will be c:\orabackup\bat and the variable LOG_DIR will be set to c:\orabackup\bat..\log, which of course is the same as c:\orabackup\log.

Note how we have two code bases to perform exactly the same job on the two main platforms (Windows and *nix). Also note how we have to account for differences of the various flavors of *nix. The net effect is that every time we make changes on one of the platforms, we have to add the same change to the script for the other platform; which of course will have to be followed by proper testing.

Rman wrapper on Windows:

```
set BIN_DIR=%~dp0
set LOG_DIR=%BIN_DIR%\..\log
```

Rman wrapper on *nix:

```
if [ `uname` = "SunOS" ]; then
  BIN_DIR=`/bin/dirname $0`
else
  BIN_DIR=`/usr/bin/dirname $0`
fi
LOG_DIR=${BIN_DIR}/../log
```

Finally, this is what it would look like in Ruby:

```
bin_dir=File.dirname(__FILE__)
log_dir=bin_dir+'../log'
```

Even though files and directories are referred to in *nix like syntax, the Ruby file IO library will translate to the platform specific syntax before the file is opened.

Anatomy of Ruby on Rails (RoR)

[\[wiki-rails\]](#) describes Rails as "... an open source web application framework for the Ruby programming language. It is intended to be used with an Agile development methodology that is used by web developers for rapid development."

Rails was extracted from the Basecamp product by David Heinemeier Hansson (AKA DHH) of 37signals.

RoR is a very elegant implementation of the Model View Controller (MVC) paradigm. Rails emphasizes convention over configuration and strongly encourages you to keep it DRY (don't repeat yourself).

Ruby on Rails is separated into the following packages:

- ActiveRecord (an object-relational mapping system for database access)
- ActionMailer
- ActiveSupport
- ActionPack
- ActiveResource

I will focus on ActiveRecord. It can be included in any Ruby program that you write. This discussion is therefore also relevant for other frameworks, such as Merb and Sinatra.

Convention over configuration

Convention over configuration really means two things to those with a “Enterprise” (read J2EE) perspective:

- There are just a few configuration files
- YML files are used instead of XML (more later)

Typical config directory in Rails

```
find config -name "*" -print | grep -v svn
config
config/boot.rb
config/database.yml
config/environment.rb
config/environments
config/environments/development.rb
config/environments/production.rb
config/environments/test.rb
config/initializers
config/initializers/backtrace_silencers.rb
config/initializers/inflections.rb
config/initializers/mime_types.rb
config/initializers/new_rails_defaults.rb
config/initializers/session_store.rb
config/locales
config/locales/en.yml
config/routes.rb
```

ActiveRecord, the gem on rails

ActiveRecord is based on the design pattern with the same name. It is a Object Relational Mapping (ORM) technique that greatly simplifies scripting of database access.

Basically ActiveRecord gives you objects that you can perform actions on without having to use sql, your code will work on any database supported by Ruby.

Simple rules:

- Table names in plural
- The primary key is always **id** (number)
- The foreign key is always **table-name-in-singular_id**

The implementation in Oracle requires a sequence:

- **table-name-in-singular_seq**

Now you can connect to the database and do operations on tables.

```
# db tables used in example:
# web_logs(id number, entry varchar2(4000))

# find all,put into list
wl_list = WebLog.find :all

# create new
wl=WebLog.new

wl.entry="Today I am presenting at OUGN"
wl.save
```

So what about that connection?

Remember, convention over configuration.

Configurations are done as yml files. If you use ActiveRecord directly, you have to pass a hash to the connect method.

```
ActiveRecord::Base.establish_connection :user=>"username",
:password=>"sekrit" .....
```

Not too elegant, let's use a config file.

```
# sample database.yml file
development:
  adapter: jdbc
  username: apex
  password: sekret
  url: jdbc:oracle:thin:@1.1.1.1:1521:apex_development
  driver: oracle.jdbc.driver.OracleDriver

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  adapter: jdbc
  username: apex
  password: sekret
  url: jdbc:oracle:thin:@1.1.1.1:1521:apex_test
  driver: oracle.jdbc.driver.OracleDriver

production:
  adapter: jdbc
  username: apex
  password: sekret
  url: jdbc:oracle:thin:@1.1.1.1:1521:apex_production
  driver: oracle.jdbc.driver.OracleDriver
```

Now, all we need is this code, which can easily be parameterized.

```
yml_filename="../../config/database.yml"
my_config=YAML::load(File.open(yml_filename))["development"]

ActiveRecord::Base.establish_connection my_config
```

Note how Ruby allows you to write method calls without parenthesis. This, combined with the dynamic nature of Ruby makes it easy to create domain specific languages (DSL). These are special purpose languages, typically based on general purpose languages. See [\[wiki-dsl\]](#) for more information.

ActiveRecord - Mapping objects to the database

Creating a reference to a table that you can use in your scripts is as easy this:

```
class WebLog < ActiveRecord::Base
end
```

Relations between tables are defined so:

```
class WebLog < ActiveRecord::Base
  has_many :comments
end

class Comment < ActiveRecord::Base
  belongs_to :web_log
end
```

The *has_many* and *belongs_to* keywords are Rails/ActiveRecord specific. They are examples of DSL.

Revisiting our Web log sample, we can now create some real working code.

```
# db tables used in example:
# web_logs(id number, entry varchar2(4000))
# comments(id number, the_comment varchar2(40000), web_log_id number)

# connect omitted for brevity

# set comment
wl_id = 33
my_comment="You rock!"

# find web log entry to comment on
wl=WebLog.find_by_id wl_id

# build comment as child of web log
wlc=wl.comment.build

# set fields
wlc.the_comment = my_comment

# Now save
wl.save
```

What about stored procedures?

There is a very comprehensive library for calling pl/sql, see [\[ruby-plsql\]](#). It was developed by Oracle ACE Raimonds Simanovskis, Oracle Developer of the Year 2009. His blog is at [\[rayapps\]](#).

At Senitel we do not use ruby-plsql because we have not been successful in running it under JRuby with JDBC. There seems to be bindings to the OCI based Oracle adapter. Look into ruby-plsql if this (dependency on OCI) is not a problem for you.

Examples of Ruby in action

In the following section I will give some examples of Ruby in action. The examples are based on real code that we have in production at customer sites or in our own hosting environment.

I have chosen three examples:

1. Dynamic Execution of SQL*Loader
2. Generic communications library for S3, ftp and ssh
3. Calling Java code

Dynamic execution of SQL*Loader

While loading a database with external data, we needed to be able to rebuild the controlfile each time. We had the same need in two different projects; one on Windows, the other on Linux.

We built a template system, modeled after the View in Model View Controller (MVC) in Rails. For each run we create a new controlfile and use that file for executing sqlldr. Everything is controlled from a Ruby script.

A central component is the embedded Ruby interpreter: ERb. This allows us to take a string and pass it to a Ruby interpreter, thus evaluating all occurrences of Ruby code in that string. The code is inserted within tags/markers as in `<%= your ruby code here %>`.

In the following example, I am adding the values from the variables `sc_filename`, `sc_insby` and `sc_status` for every run.

afile.dat:

```
load data
append
into table my_table
fields terminated by ';'
trailing nullcols
(
  prod char
, kunde char "trim(:kunde)"
, dekning char
, navn char
, personnr char
, gateadresse char
, postnr char
, poststed char "trim(:poststed)"
, gateadresseobj char "trim(:gateadresseobj)"
, postnrobj char "trim(:postnrobj)"
, poststedobj char "trim(:poststedobj)"
, fradato char "to_date(substr(:fradato,1,10), 'dd.mm.yyyy')"
, tildato char "to_date(substr(:tildato,1,10), 'dd.mm.yyyy')"
, oppsigelse char "to_date(substr(:oppsigelse,1,10), 'dd.mm.yyyy')"
, bygger char "decode(length(trim(:bygger)), 4, trim(:bygger), '1899')"
, bygning char "trim(:bygning)"
, krypkjeller char "trim(:krypkjeller)"
, byggemte char
, premie char
, sc_insdate sysdate
, sc_insby constant '<%= sc_insby %>'
, sc_filename constant '<%= File.basename(sc_filename) %>'
, sc_status constant '<%= sc_status.capitalize %>'
)
```

Ruby code to evaluate string:

```
sc_filename='afile.dat'  
sc_insby='Espen'  
sc_status='Imported'  
template_file_name="afile.erb"  
template=File.read(template_file_name)  
  
controlfile=ERB.new(template).result(binding)
```

Generic communications library for S3, ftp and ssh

To manage distribution of the files in the previous example we have developed a generic library that operates on remote files and folders.

The most common operations (methods) are:

- `ls`
- `save_file` (put)
- `download_file` (get to local file)
- `get_file` (get into memory object)

Since the library supports operations on files as well as in-memory objects, we can stream from a remote file directly into a lob/clob in the database.

The code required to “store” ftp files in memory illustrates the dynamic and flexible nature of Ruby. We have extended the base `Net::FTP` class to achieve this effect by adding four methods.

- `getbinaryfile_as_stream` (based on `getbinaryfile`)
- `gettextfile_as_stream` (based on `gettextfile`)
- `putbinaryfile_from_object` (based on `putbinaryfile`)
- `puttextfile_from_object` (based on `puttextfile`)

Ruby on Rails (RoR) as a back-end processor for Apex

The listing below contains all the code needed to enable this effect.

```
# 1st, overload FTP so that we can download as stream
module Net

  class FTP

    # note that line breaks, comma signals continuation
    def getbinaryfile_as_stream(remotefile, blocksize = DEFAULT_BLOCKSIZE,
      &block) # :yield: data
      raise "Need to provide a block" unless block

      # no retry, but can be checked in @resume .. check source
      rest_offset=nil

      retrbinary("RETR " + remotefile, blocksize, rest_offset) do |data|
        yield(data)
      end
    end

    def gettextfile_as_stream(remotefile, &block) # :yield: line
      raise "Need to provide a block" unless block

      retrlines("RETR " + remotefile) do |line|
        yield(line)
      end
    end

    # note that line breaks, comma signals continuation
    def putbinaryfile_from_object(remotefile, data,
      blocksize = DEFAULT_BLOCKSIZE)
      rest_offset=nil
      f=StringIO.new(data, 'r')
      storbinary("STOR " + remotefile, f, blocksize, rest_offset)
    end

    def puttextfile_from_object(remotefile, data)
      f=StringIO.new(data, 'r')
      storlines("STOR " + remotefile, f)
    end

  end
end
```

In the case of `getbinaryfile_as_stream` our class library for ftp uses this new method in the following way.

```
def get_file(file_name, folder_name=nil)
  connect
  clob=nil

  remote_name=get_remote_name(folder_name, file_name)
  @conn.getbinaryfile_as_stream(remote_name) do |data|
    clob=clob ? "#{clob}#{data}" : data
  end

  clob
end
```

In the scripts.

```
my_clob=conn.get_file remote_file, remote_directory
```

Calling Java code

The basic “backup” of Apex is externalized as Java class called APEXExport. We created the simple script below to run the export/backup.

This script will only work with JRuby and it can easily be extended to run in a J2EE container. You probably need to add some simple locking/sequencing to avoid collisions.

```
require 'java'

# Needs to be in CLASSPATH
import "oracle.apex.APEXExport"

# pass all params as String!
def export_application(db, user, password, applicationid)

  # get handle to Java objects
  my_exporter=APEXExport.new()
  my_export_class=my_exporter.class

  # build arg list, multi-line for readability
  argv=[]
  argv << "-db" << db
  argv << "-user" << user
  argv << "-password" << password
  argv << "-applicationid" << applicationid

  # to java
  jargs=argv.to_java(:string)

  # output is f<applicationid>.sql, in directory where code runs
  my_export_class.main(jargs)
end

def main(argv)
  es="Use: #{__FILE__} dbhost.senitel.no:1521:db user pass applicationid"
  raise es unless argv.size==4
  export_application(argv[0], argv[1], argv[2], argv[3])
end

#
# Launch main if not irb
#
if $0 == __FILE__
  begin
    main(ARGV)

  rescue Exception => e
    $stderr.puts "Error: #{e.to_s}"
    exit 1
  end
end
```

Notes on deployment and integration

Deployment

Traditionally, deploying Rails applications has been somewhat of a black art. The situation has improved immensely over last few years and it (the task) should be achievable for anyone familiar with web servers and databases.

In many cases you will end up using the ActiveRecord library in command line scripts. Depending on your preferences, you may wish to use the native (AKA CRuby) VM or opt for the JRuby interpreter.

Other times, you will need a full application server setup. In those cases we strongly recommend that you deploy using [\[warbler\]](#). This will package your Rails code as a J2EE application that can run in any J2EE compliant server. This will insulate you from all types of problems related to native libraries *and* give you a much larger audience for your rails application: Your Rails application can be viewed as just another J2EE component.

Integration

When we started out with Ruby and Rails, we had grand ideas of how to use the *http_util* PL/SQL package for interaction between Rails services and Apex.

We ended up using Ruby in pretty much the same way that we had been using scripts in the past. The difference is that our Ruby scripts run on both Linux and Windows, we have full database connectivity in the scripts and the scripts are written in a beautiful, easy-to-read, language.

References

[rayapps] <http://blog.rayapps.com>

[ruby-plsql] <http://github.com/rsim/ruby-plsql>

[matt-asay] http://news.cnet.com/8301-13505_3-10453213-16.html

[fowler] Fowler, Chad. 2009. *The Passionate Programmer*. Dallas: The Pragmatic Bookshelf

[indeed.com] <http://www.indeed.com/jobtrends?q=Ruby%2C+Rails%2C+PHP%2C+Perl&l=&relative=1>

[wiki-ruby] <http://en.wikipedia.org/wiki/Ruby%28programminglanguage%29>

[wiki-rails] <http://en.wikipedia.org/wiki/RubyonRails>

[wiki-ar] http://en.wikipedia.org/wiki/Active_record

[wiki-dsl] http://en.wikipedia.org/wiki/Domain-specific_language

[warbler] <http://caldersphere.rubyforge.org/warbler/Warbler.html>

About the author

Espen Brækken is the Consulting Director at Senitel Consulting AS in Oslo. In the past he has served as Director of Engineering at e-Travel and Practice Manager at Oracle.