Plug-in application for automatic generation of administration panel

Piotr Jakubowski

January 24, 2011

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Description

Along with the huge popularization of the Internet there appeared enormous market for all sorts of web applications. Every month big companies, as well as small startups, try to get as much of this market by bringing in new features and shocking the crowd with innovative functionalities. Surprisingly, the market seems to be far from saturation as brilliant ideas except for fulfilling known demand create new needs as well.

New tools started to emerge in order to meet the needs of Internet population. Dynamic languages started to supersede old and known. Python and Ruby became the new Java and C. The speed of development surpassed the speed of execution as technical limitations and price of servers are not the issue any more. Moreover, web development frameworks made it possible to code the application on the new level of abstraction - we can instantaneously start to implement our business logic, without the need to think about how we should persist our entities into database and how we should handle requests and respond to them. Java developers got Spring and Hibernate, but the noteworthy Ruby on Rails took web development to the whole new level. The threshold of making the ideas happen has been successfully shrunken.

As mentioned before, today's software development has been focused on the effectiveness and speed of development. A lot of work has been done to create developer-oriented tools that try to make developing business ideas effortless.

Of course, there is plenty of room for improvements. One of them is the idea of automating process of creating the administration panel for the application. Very often administration part of the application is mundane

collection of CRUD[1] actions. It is the "must do" part.

User-oriented features is the target thing of development. This is what makes money and keeps us all employed. It seems like possibility to be able to automatically generate panel for administering our resources would make development of web application even more dynamic. It would bring plenty of benefits, such as:

- Whole team can focus on what really matters, which is creating functionality that would be an added-value for users of the application
- Adding new features and resources to the application would require less effort - as soon as the user-oriented functionality is ready the administration part is automatically generated.
- Business people connected with the project can instantly browse new resources in the application.

Thereby, the aim of this thesis is to create and demonstrate the process of creation of plugin that would enable automatic generation of administration part of the application. The plugin would be developed for Ruby on Rails framework mentioned before.

1.2 Existing Solutions

Thanks to its rapidly growing popularity, Ruby and Ruby on Rails got very broad community. Not only does it result in a huge number of places you can ask for help, but also in all kind of libraries that would make the life of web developer easier. And as it seems I am not alone stating the issue described above. There are already several implementations attacking the problem of admin interface. Some of them are more successful than others, there are different kinds of approaches and different outcomes.

In order to see what market has to offer I looked through available solutions and chose 4 most popular ones in order to see what kind of functionalities they provide and what flaws they suffer from.

1.2.1 Active Scaffold[2]

According to Ruby Toolbox[3] it is the most popular plugin for admin interface generation. Unfortunately, it seems like the days of glory for this gem are long

[1] Create Read Update Delete

[2] http://activescaffold.com/

[3] http://ruby-toolbox.com/categories/rails_admin_interfaces.html

gone. The plugin does not support Rails in version 3, which despite being released very recently already became a standard for new applications.

Technicalities aside, it seems to be quite functional. The almost out-of-the-box look is presented on figures 1.1 and 1.2

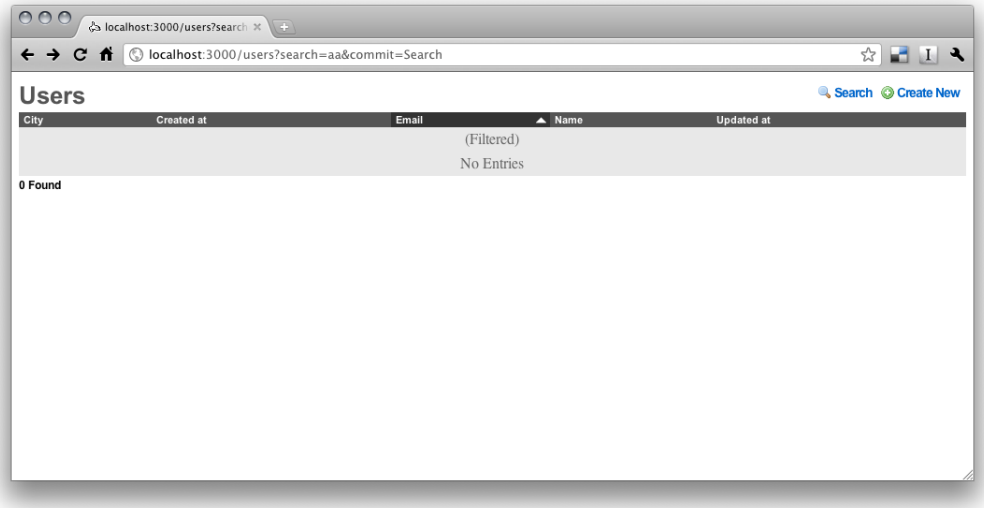

Figure 1.1: Active Scaffold

Active Scaffold ships with comprehensive API for customizing its behavior. User can change which columns are visible in which view or how the system behaves during particular actions. Moreover, it uses Asynchronous JavaScript and XML (AJAX) for interaction with user which gives nice and responsive user interface.

Unfortunately, it has big disadvantage. It is not entirely automatic. It requires developer to take following action to set it up:

- Add command to the layout of the application that will include styles and javascript for Active Scaffold.
- For every resource he wants to administer, he needs to create controller and add code that will set it up.
- Set up url for Active Scaffold actions for given controller by adding code to routes file.

This results in following inconveniences:

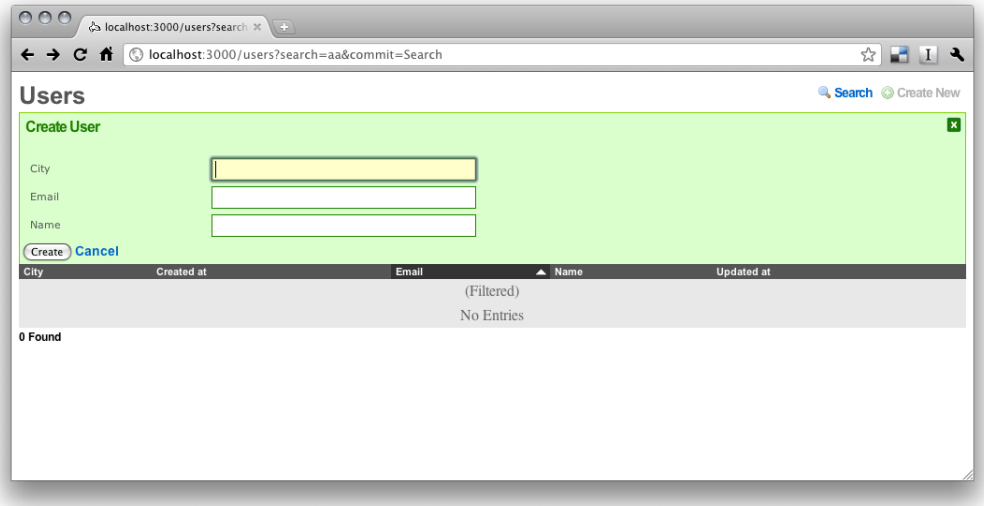

Figure 1.2: Automatically generated form in Active Scaffold

- Active Scaffold mixes with our application code.
- Every time we create some model we need to take care to set up Active Scaffold for it.

It seems like Active Scaffold instead of being "stand-alone" solution for administration is a set of commands, that speeds up set up of administration interface. It gives flexibility, but is not as hassle-free as could be and as developers would like it to be.

1.2.2 Typus[4]

Typus is reported to be the second most popular Rails admin interface generator. As opposed to Active Scaffold, its development team is still active. Thanks to that, it has been adapted to Rails 3 already.

Installation and set up is quick and effortless. We just need to install Typus and run one command. That way we get to the /admin path in our application. Its user interface is presented on figures 1.3 and 1.4.

Documentation provides thorough description of possibilities of Typus' customization. We can decide which columns will be visible, how admins can search our models and so on. Most of the configuration is done by specifying

[4]http://core.typuscms.com/

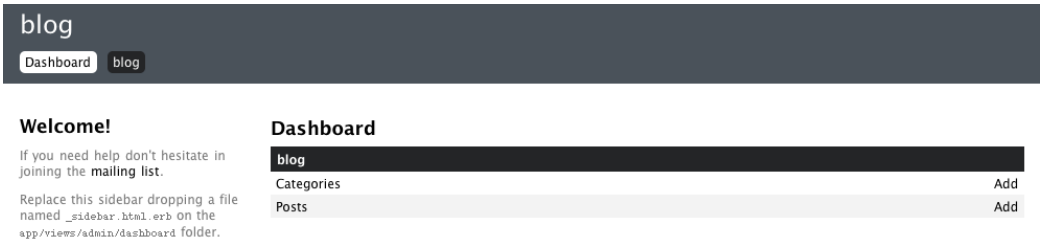

Figure 1.3: Typus main page

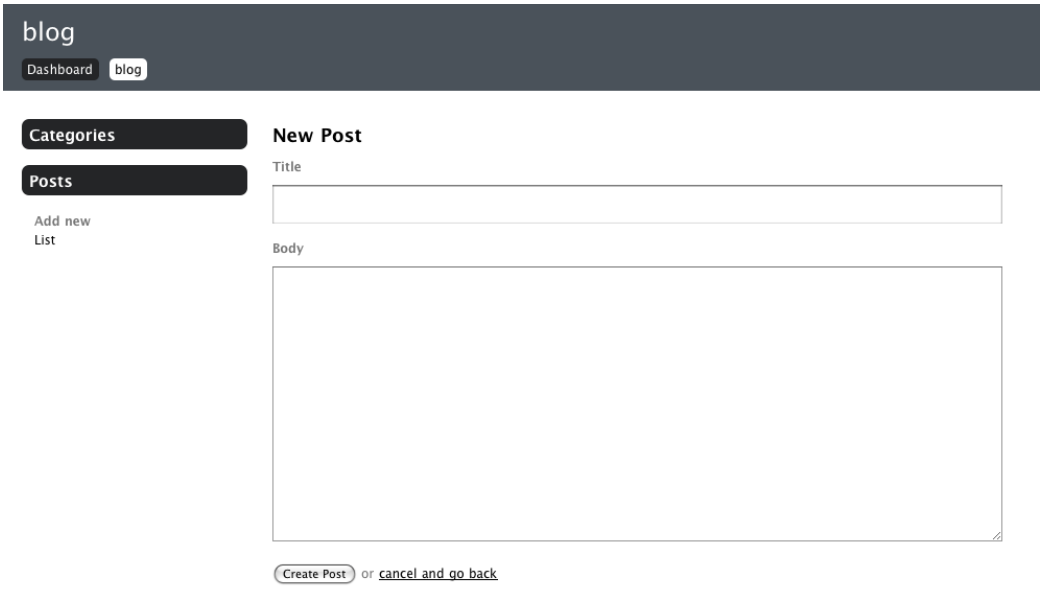

Figure 1.4: Automatically generated form in Typus

appropriate YAML files. In addition, Typus is supposed to handle file uploads in your application, which is very convenient. However, it will happen only if your application is using particular plugin for files upload called Paperclip.

But again, similiar as it was in Active Scaffold, Typus does not work entirely "on-the-fly". The generator we have to run in order to set Typus up, except for copying needed HTML, javascript and image files, creates controllers that serve particular Models of our application. As said before, this means that we can get full control on how given controller behaves - we

can override Typus implementation with our own - but such approach is not as dynamic in terms of adapting to upcoming resources in the application. Of course, now it is just the sake of running simple command that would take care of everything once we introduce some new Model into application, but surely it is not something we just install and can forget about.

1.2.3 admin_data

Admin_data is next one on the list. Again, the project seems to be still maintained and has been upgraded to Rails 3 already. Installation is very much alike to the one of Typus. But, after installation we do not need to run any generators or create any kind of files. We instantly get access to /admin_data path and figure 1.5 presents what more or less we would see after trying it out.

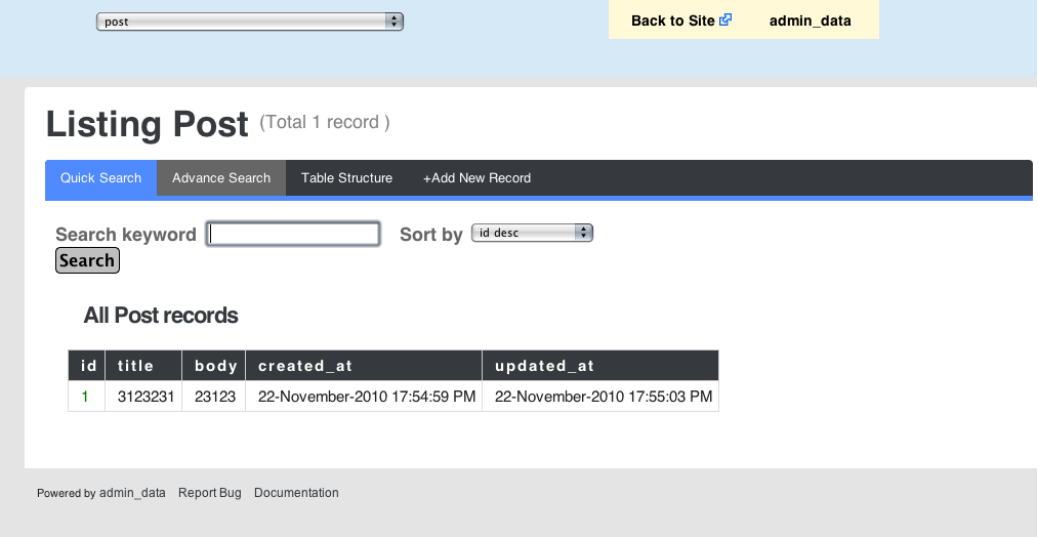

Figure 1.5: Admin data

Of course, it provides us with some level of customizability, though it is not as wide as in two other projects. But, admin_data has the level of dynamism I have been looking for. It creates the list of models on the fly, so while developing our application we will constantly have the most current version of our models lists, no question asked. Moreover, we can also see table's structure for given model, which may come in handy at times.

On the other hand, you can clearly see that admin_data has been created with programmers in mind. The user interface can be very hard to comprehend for people that do not know what is SQL or do not care how developers

organized tables in the database. Therefore, it could be very hard to convince our business people that it is something they can use.

1.2.4 rails_admin

Although rails_admin is not present on the ruby-toolbox list it has been very popular lately. It has been developed during Ruby Summer of Code[5]. It has been developed strictly for Rails 3 so it uses all the best features and the best practices from the newest version of the framework.

Installation is very quick, although we need to go through few steps. Moreover, it depends on devise[6] gem, which takes care of users authentication, as rails_admin's default option is to make people to log in to the administration panel.

What we get after installation is very nice and user-friendly panel(figure **??**). In addition, it tracks changes in records, what gives us functionality of seeing whole history of activities on records as well as people that made them. Additionally, we have that data presented in form of nice graph that gives us an idea of what was happening in the system throughout the month or year.

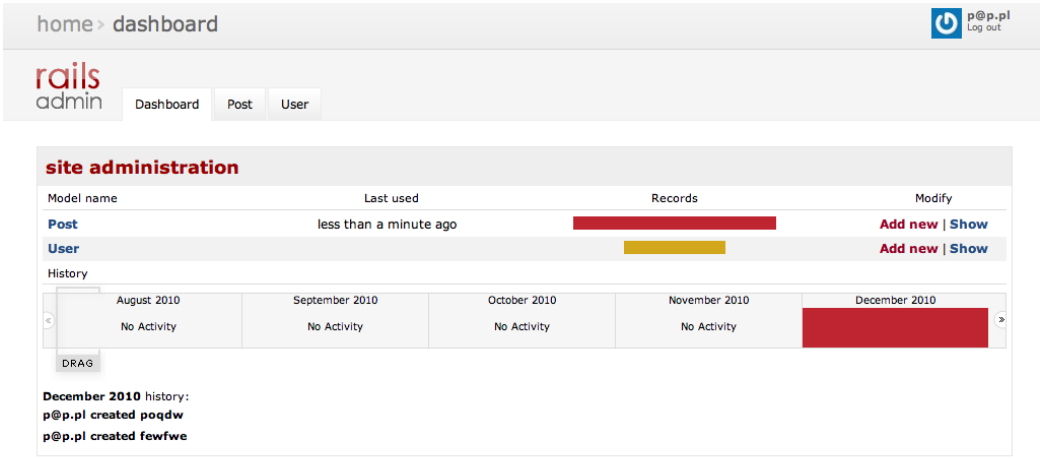

Figure 1.6: rails_admin dashboard

Rails_admin seems to have real potential and although I have not seen it in use in production yet, soon it maybe become real power of Rails applications.

[5]http://rubysoc.org/
[6]https://github.com/plataformatec/devise

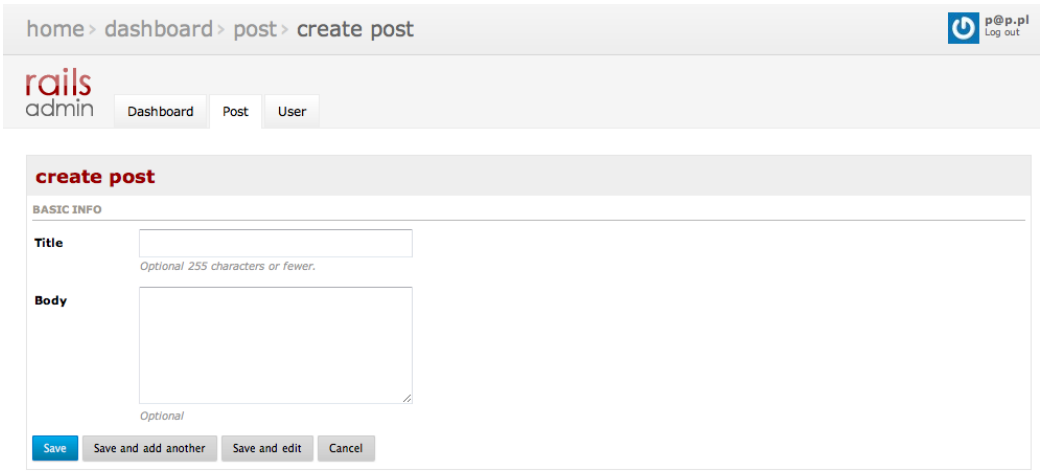

Figure 1.7: Form generated in rails_admin

Definitely, ideas used in that project are very interesting and may be very useful.

Chapter 2

Project definition

This chapter aims to state the scope of the project. Moreover, I would like to draw an outline of expected output and point out challenges that I would have to face during the development.

2.1 Scope of the project

As stated in the introduction, generation of administration panel seems to be an actual problem. Therefore, my aim is to develop a plugin or rather a gem[1] for Ruby language that would provide functionality of administration panel generation for Ruby on Rails application.

One of the first assumptions was that the installation of the administration panel should be effortless and should not require any changes to the code of application. Moreover, the administration panel should detect changes to business models "on-the-fly" and should adjust to reflect those changes in the panel without need of running any generators or other scripts.

The functionality of the administration panel would consist of:

- Listing all of business models in the application.
- For particular model listing all of the records.
- Creation and edition of records using automatically generated forms.
- Deletion of records.
- Proper handling of associations between models.
- Configuration of the panel regarding which models or model fields should be visible.

[1] TODO: More on gems in chapter bla bla bla

2.2 Project outline

The project would be developed for test application that would be a simple blog system. The functionality of the test application would be a subject of adjustments in order to check and prove appropriate functioning of the plugin.

To start with, test application would allow following actions:

- Add, edit, list and delete posts
- Add, edit, list and delete categories
- Assign posts to appropriate categories

Target functionality of plugin has been stated in previous section. Mock-ups of user interface of the plugin are depicted in figures **??**, **??** and **??**

Figure 2.1: Mockup of dashboard page of the plugin

2.3 Anticipated challenges

The challenge that is most striking while approaching this project is the fact that the solution has to be universal and totally independent of any

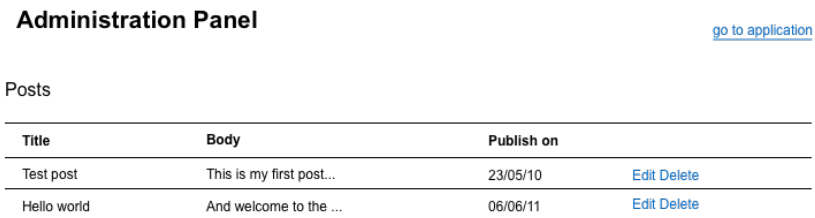

Figure 2.2: Mockup of page for listing records of given model

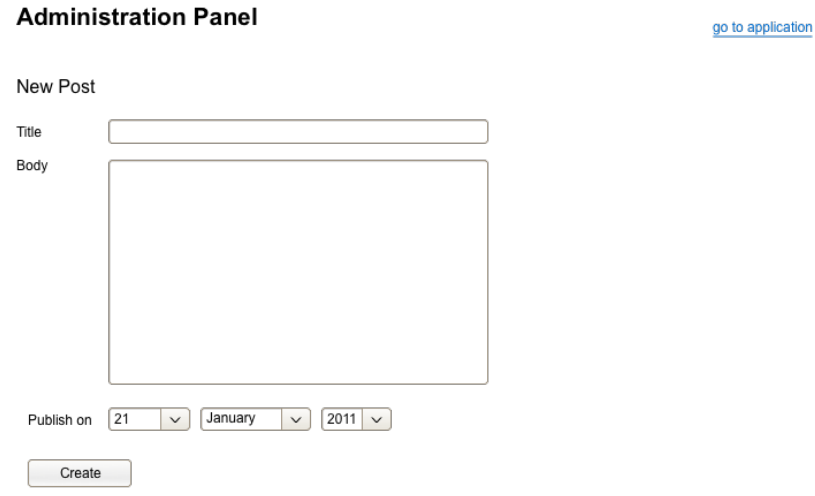

Figure 2.3: Example page for creation or edition of model record

domain specific workarounds. As a plugin, it has to be usable for all or at least majority of web applications. Therefore, it is necessary to take into consideration all kinds of association between models and all types of fields that may appear in those models. Moreover, I would have to watch out not to violate any business logic that some developer put in his models or not to introduce data inconsistency. Fortunately, well-design systems would rather fail raisin an exception than allow to perform some illegal action on models.

Furthermore, it seems like a lot of code in this project would need to figure out actions itself during the runtime basing on what kind of code would be in the application. In order to make that plugin functional, it would be necessary to dive into and make use of meta-programming, which may not be as easy to use, clear and readable. Fortunately, Ruby from its nature of being a dynamic language allows many meta-programming actions and provides developers with very pleasant reflection API.

Hopefully, those challenges would get overcome during design and development of the project in order to produce plugin that would make life of web programmers slightly easier.

Chapter 3

Research

The aim of this chapter is to present the environment in which the plugin will be developed. I will present, firstly, the features of Ruby language and then what is Ruby on Rails, how it is constructed and how we can enrich it with plugins. In the end I will take a look into possibilities of metaprogramming in Ruby and Rails.

3.1 Ruby

Here you will find information abut Ruby language and especially those features that create the power of Ruby and distinguish it from other programming languages.

3.1.1 Basic information

History

Ruby language has been created by Yukihiro Matsumoto also known as "Matz" for english speaking programmers. It firstly appeared publicly in 1995 with version 0.95 in order to reach version 1.0 a year later. As of the time of writing this document, the latest release of Ruby is of branch 1.9 (specifically 1.9.2), but the branch 2.0 with brand new exciting features is emerging on the horizon.

While creating Ruby Matz stated that he focused on developing a language that would be programmer-friendly and that would make the barrier between idea and putting this idea in code as low as possible. The most famous quote of Matz fully describes the philosophy of the language:

> Ruby is designed to make programmers happy

That explains the idea behind the project.

Implementation

The official implementation of Ruby has been written in C language. As no particular standard has been developed for the language, the official implementation is a reference for all other vendors. Nonetheless, there is plenty of other implementations including one operating in JVM(JRuby[1]), .NET framework(IronRuby[2]) and Objective-C runtime(MacRuby[3]).

3.1.2 Features

Basic features

It is worth mentioning few basic features of Ruby language that would make the further part of this chapter clearer.

- Ruby is scripting language - statements are executed as provided and there is no special `main` function
- Ruby is dynamically typed
- Method invocations do not need to include parentheses

Everything is an object

Ruby is an object oriented language. In fact, every value in Ruby is a object. Even such "primitive" values as integers, true, false or nil (which is Ruby's NULL). Thanks to that, Ruby promotes and enforce object oriented programming.

Moreover, you can have methods on integers or other primitive values which makes the code much more readable, shorter and enjoyable. For example instead of having Java-like:

```
NumbersConverter.arabicToRoman(1);
```

You can have much more readable:

```
1.to_roman
```

Ruby standard library does not include `to_roman` function for integers, but we could easily add one by using next described feature.

[1] http://www.jruby.org/
[2] http://ironruby.codeplex.com/
[3] http://www.macruby.org/

All classes are open

Ruby gives us freedom to change already declared classes. Moreover, it does not mean that we can change only the classes that we defined. We can change even classes defined by Ruby standard library. Therefore we can open for example class Integer that represents all integer values in Ruby code and add method `to_roman`:

```
class Integer
  def to_roman
    # Code that would change
    # arabic number to roman one
    # and return it as string
  end
end
```

Now, we would be able to have following line in our code:

```
4.to_roman #returns "IV"
```

Power of blocks

This is one of the most exciting features of Ruby language. Block is a fragment of code that can be passed to a function and the function may call this code anywhere inside its body.

Block in ruby can be denoted in two ways:

```
do
  #here code
end

#or

{
  #here code
}
```

Usually the first way is used for blocks that span throughout multiple lines, while the second way is used for single line blocks.

Blocks can be used in number of ways. The most popular are iterators:

```
a = [1, 2, 3]
a.each do |i|
  puts i
end
```

Above code would result in printing 1, 2, 3 to the output. Statement `do |i|` means that this block expects one argument called `i` just like regular functions do.

In order to understand how blocks operate, let me present simple example:

```
class Array
  def each_nested(&block)
    for i in 0...self.size do
      if self[i].is_a? Array
        self[i].each_nested &block
      else
        yield self[i]
      end
    end
  end
end

multidimensional_array = [
  [11, 12, 13],
  [21, 22],
  [31, 32, 33, 34]
]

multidimensional_array.each_nested { |element| puts element }

# Execution Result:
# 11
# 12
# 13
# 21
# 22
# 31
# 32
# 33
# 34
```

The above example adds the `each_nested` method to an array that allows us to perform operations on every element of multidimensional matrix. It iterates over every element of an array. If given element is again an array then it recursively calls `each_nested` on it and if it is other element then it passes the control to a block along with the element as a parameter.

As we can see appropriate use of blocks may be very useful and can make

code much shorter and more readable

3.2 Summary

This chapter demonstrated basic features of Ruby language. In order to get more information on this language I forward the reader to the official website[4] or to some books from bibliography[**?**].

[4] http://ruby-lang.org/