Graphical User Interface Plugin

Development Documentation

Contents

List of Figures

1 Introduction

The development of the web-based Earth application by Rising Sun Pictures (RSP) over the Ruby-On-Rails (ROR) framework further illustrates the benefits of software reuse in terms of enhancing the versatility and extensibility of software applications. Instead of building the application from the ground up, the Earth application is developed atop the base code that is auto-generated by the relevant components of the underlying ROR framework. This development strategy means that the design of the Earth application is leaning significantly towards the Model-View-Controller (MVC) template of ROR-based software applications. The other application components of the Earth application such as the daemon and the built-in web server is then designed around the auto-generated base code. Presently, having been developed with a RSP design philosophy in mind, these additional codes (around the ROR auto-generated base code) are relatively static and requires significant efforts in order to accommodate new or later requirements.

As part of the project, the possibility of further developing the application in order to accommodate the reconfiguration of the existing design at a later stage was investigated. Such a reconfiguration option involves compartmentalising the various aspects of the existing web-based Graphical User Interface (GUI). A further goal of this initiative is to simplify the process with which the users of the Earth application could reconfigure the current GUI according to their own designs or requirements specifications. This exploratory task uncovered the ROR provisions for the plugin-type reconfiguration of the base code.

The following section will present how the existing ROR plugin provisions are activated and included as part of the reconfiguration option for the web-based Earth application.

2 Development Updates

The actual implementation of the GUI plugin was reached through a number of exploratory and discovery stages namely, the investigation of the existing plugin provisions in the ROR framework, the preparation by design of the demarcated features and functionalities, and the actual implementation in code of the approved GUI plugin design.

2.1 Investigation

Upon closer examination, it was established that the ROR plugin provisions can effectively enable the implementation of independent functional modules for the Earth application web-based GUI. This revelation means that there is no real need to devise or create an entirely different plugin manager for the GUI plugins as the existing ROR framework has adequate facility for such feature. However, since the various component for each GUI would have to be placed in different parts of the code in conformance to the MVC design aesthetics, figuring out each parts and where it should be placed imposes some challenges for the user or developed tasked with creating the GUI plugins. Nevertheless, the ROR framework provides a neat way of dealing with challenge and simplifies the development process a great deal still.

A ROR plugin is either an extension or a modification of the core framework. Such plugins can do almost anything that a ROR application can. The provided plugin generator script is able to copy the respective files into their respective application sub-directories. In addition, the generator script can process an ERB file and have it copied into the migration folder through the use of the following migration templates:

- **Models** Put a model in the plugins `lib` folder and use `generator` to copy it to `app/models` folder.

- **View Helpers** A helper method can be included, or mixed, into the rest of the application. (This is also known as "mixin".)
- **Controllers** Copies a controller into the `app/controllers` directory.
- **rake Tasks** Creates a `rake` file into the tasks folder.
- **Images, Stylesheets, Javascripts** The `generator` copies these into the public directory.
- **Test assertions** To be mixed-in to the tests cases.
- **Unit and Functional tests** Can be generated like controllers.

Ultimately, this investigative process confirmed that the ROR plugin framework can be used to create GUI plugins for the Earth application project.

2.2 Preparation

Creating a GUI plugin using the ROR framework requires the satisfactory completion of a number of preparatory tasks. First, a clear description and conceptual design of the proposed plugin-able feature should be documented and determined as being logically sound. All aspects of this proposed feature should be then be identified such as the relevant source files that needs to be created or modified. Information about where these files will reside within the codebase should also be established before the relevant generator script, which will process and copy the files into the app subdirectory of Earth application codebase, is invoked. A detailed description of the actual implementation in code is provided next.

3 Implementation

The GUI plugins are implemented using the ROR plugin framework. However, some code changes had to be made to the original 'VIEW' codes. This is to allow the seamless addition of extra features such as a new tag or search field. The structure of the ROR plugin framework is shown in Figure 1 below.

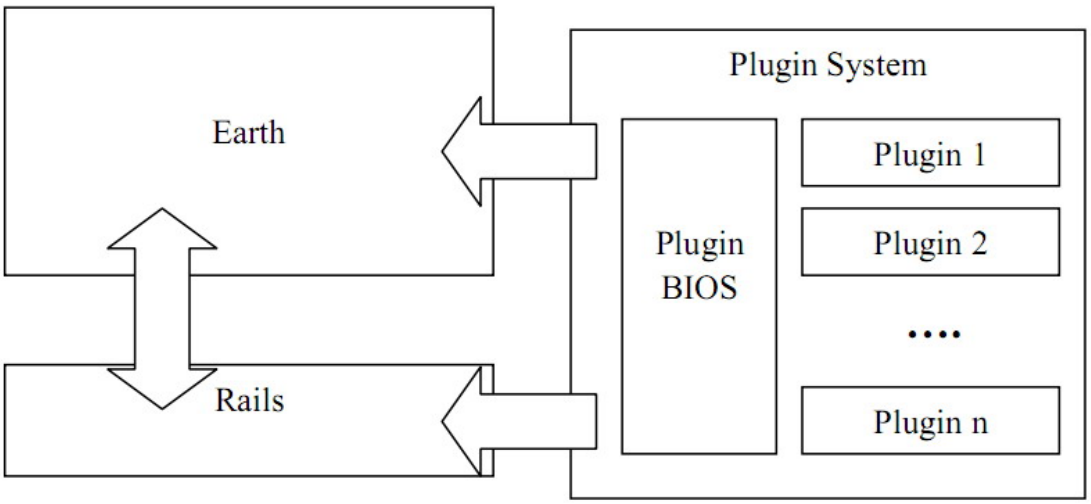

Figure 1: The structure of the GUI Plugin framework.

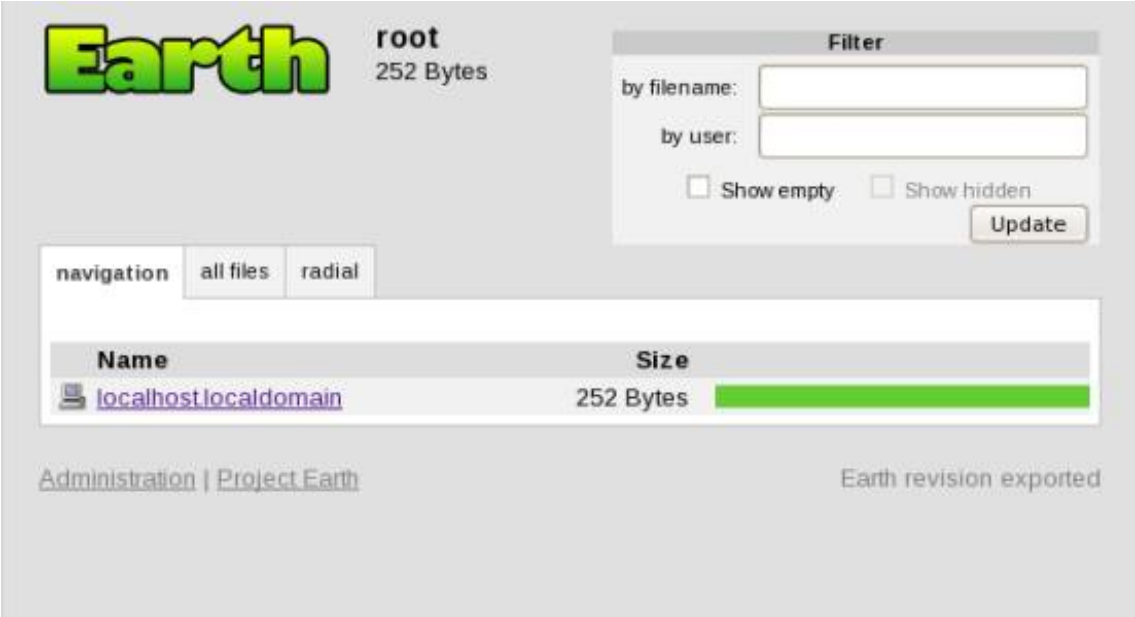

Figure 2: Original Earth front-end.

The ROR plugin framework is being adopted to extend the existing features and functionality of the Earth application. This resulted in the creation of an intermediate conceptul layer that is designed to facilitate the communication between the plugins, the Earth application daemon and the codes of the underlying ROR framework. This conceptual layer can be further expanded to include dependency checks and sub-classing feature. This also means that for each plugin, the configuration file 'plugin.cfg' is created for the purpose of retaining information about the corresponding plugin. This configuration file is then used as a manifest of the plugin package.

4 Instruction

This section will describe the steps on how to develop, install and uninstall a GUI plugin for the Earth application. For ease of explanation, a simple yet fully-featured plugin called 'mr_bogus' is used as an example. Figure 2 shows a screenshot of the web-based GUI for the Earth application before the 'mr_bogus' plugin is installed.

4.1 Development

As mentioned before, the generate script is first executed to create some basic necessary structures and files. To do so, the following command can be used:

```
./script/generate plugin mr_bogus --with generator
```

A blank plugin in Rails have been created after the previous command was executed successfully. At this stage, one can safely edit the affected files to have the final results displayed. Views can be added by creating new directories and files in the `./vendor/plugins/mr_bogus/generators/mr_bogus/templates` directory. For example, the action controller needs the directory called "controllers". Finally, update the file called "mr_bogus_generator.rb", which can be executed to create particular file wherever one wants. More instructions can be found in this web-site `http://wiki.rubyonrails.com/rails/pages/HowTosPlugins`.

Here an action controller and a webpage were created for views:

```
[keane@localhost Earth]$ ./script/generate plugin mr_bogus --with-generator
      create  vendor/plugins/mr_bogus/lib
      create  vendor/plugins/mr_bogus/tasks
      create  vendor/plugins/mr_bogus/test
      create  vendor/plugins/mr_bogus/README
      create  vendor/plugins/mr_bogus/MIT-LICENSE
      create  vendor/plugins/mr_bogus/Rakefile
      create  vendor/plugins/mr_bogus/init.rb
      create  vendor/plugins/mr_bogus/install.rb
      create  vendor/plugins/mr_bogus/uninstall.rb
      create  vendor/plugins/mr_bogus/lib/mr_bogus.rb
      create  vendor/plugins/mr_bogus/tasks/mr_bogus_tasks.rake
      create  vendor/plugins/mr_bogus/test/mr_bogus_test.rb
      create  vendor/plugins/mr_bogus/generators
      create  vendor/plugins/mr_bogus/generators/mr_bogus
      create  vendor/plugins/mr_bogus/generators/mr_bogus/templates
      create  vendor/plugins/mr_bogus/generators/mr_bogus/mr_bogus_generator.rb
      create  vendor/plugins/mr_bogus/generators/mr_bogus/USAGE
```

Figure 3: Running the `generate` script to create mr_bogus plugin.

```
class BogusController < ApplicationController
        def bogus
                #empty action
        end
end
```

Figure 4: The `bogus_controller.rb` file.

```
./vendor/plugins/mr_bogus/generators/mr_bogus/templates/controllers/bogus_controller.rb
./vendor/plugins/mr_bogus/generators/mr_bogus/templates/views/bogus.rhtml
```

Figure 4 shows the content of the `bogus_controller.rb` file, and Figure 5 shows the content of the `bogus.rhtml` file.

The file named `mr_bogus_generator.rb` would be modified for preparation of moving files to particular folders, as shown in Figure 6.

Once done, the following command can be executed to have the plugin implemented into Rails:

```
./script/generator mr_bogus bogus
```

To test the plugin, simply point the web-browser of choice to the following URL:

```
http://localhost:3000/bogus/bogus
```

```
<html>
        <head>
        <title>Hello, Mr.Bogus!</title>
        </head>
        <body>
        <h1>Hello, Mr.Bogus!</h1>
        </body>
</html>
```

Figure 5: The `bogus.rhtml` file.

```
class MrBogusGenerator < Rails::Generator::NamedBase
  def manifest
    record do |m|
          m.directory "app/views/bogus"
          m.file 'views/bogus.rhtml', "app/views/bogus/bogus.rhtml"
          m.file 'controllers/bogus_controller.rb', "app/
controllers/bogus_controller.rb"
    end
  end
end
```

Figure 6: The `mr_bogus_generator.rb` file.

```
#plugin name: plugin_name
plugin_name,bogus
#require component,view string,controller name,action name
tab,Mr_Bogus,bogus,bogus
```

Figure 7: The `plugin_cfg` file.

However, the plugin should be implemented into the existing Earth system instead. To do so, a configuration file has to be created. The file is named "plugin_cfg" in the `./vendor/plugins/mr_bogus` directory. Figure 7 shows the content of the configuration file.

Now, point the web-browser to the homepage of Earth. The result is shown in Figure 8. Figure 9 shows the implementation of the plugin.

So far, the plugin had included a new tab in Earth. However, to add a new text field in the filter section, the following can statements can be added into the configuration file:

```
#require componenet, view string, parameter name
field_tag, by wine, wine
field_tag, by cake, cake
```

Figure 10 shows the refreshed the homepage of Earth.

The plugin can now be tested and debugged. Finally, a set of installation and uninstallation scripts can be created from the following steps:

1. Find files and folders created.
2. Modify the installation in root of plugin to have the appropriate files copied to their appropriate destination directories.
3. Build the plugin as a package including the installation file.

The following is the content of installation of "mr_bogus" named "install.rb":

```
require 'fileutils'
plugin_name = ARGV[1] # mr_bogus
earth_root = ARGV[0]
puts "Creating the plugin"
system "ruby #{earth_root}/script/generate plugin #{plugin_name} --with-generator"
RAILS_ROOT = earth_root
FileUtils.cp File.join(File.dirname(__FILE__), 'init.rb'),File.join
(RAILS_ROOT, 'vendor','plugins',plugin_name,'init.rb')
FileUtils.cp File.join(File.dirname(__FILE__), 'install.rb'),File.join
(RAILS_ROOT, 'vendor','plugins',plugin_name,'install.rb')
FileUtils.cp File.join(File.dirname(__FILE__), 'uninstall.rb'),File.join
```

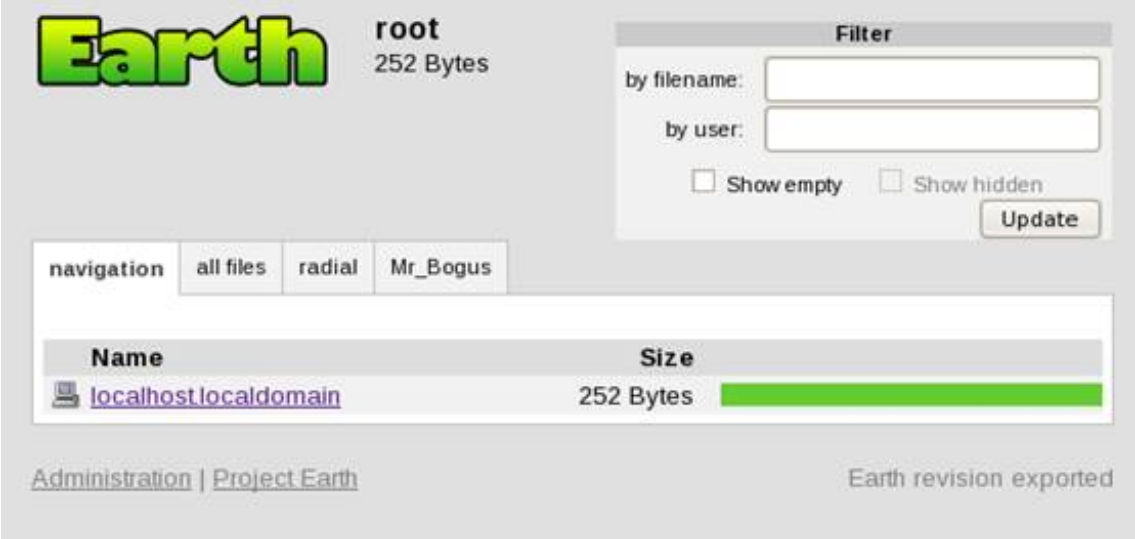

Figure 8: `mr_bogus` implemented in Earth.

Figure 9: The `mr_bogus` tab.

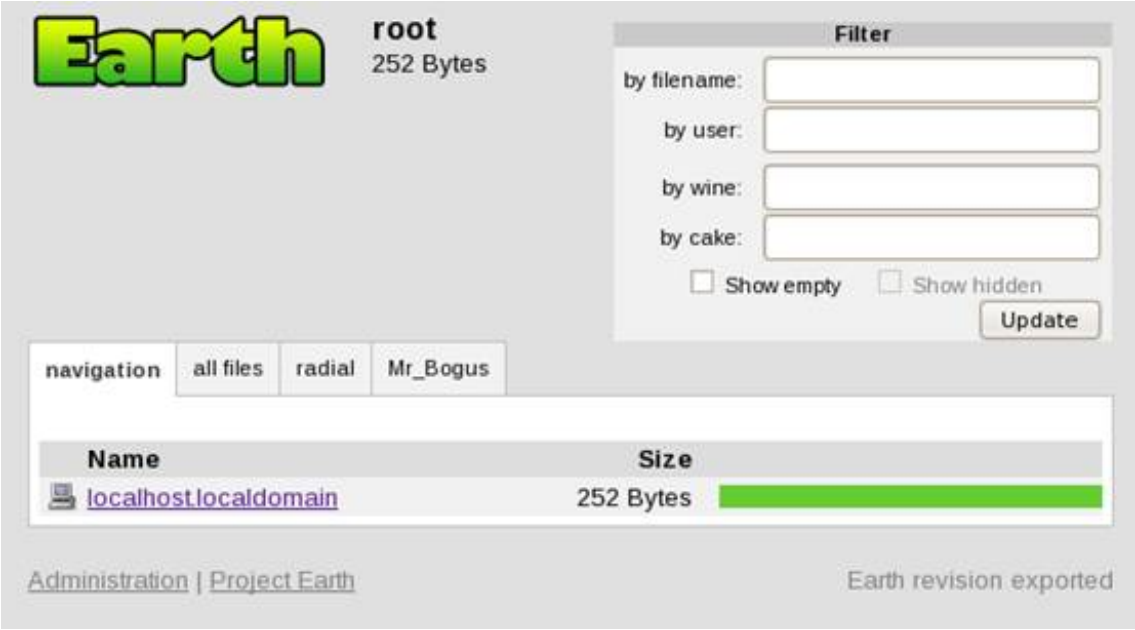

Figure 10: New text fields.

```
(RAILS_ROOT, 'vendor','plugins',plugin_name,'uninstall.rb')
FileUtils.cp File.join(File.dirname(__FILE__), 'plugin_cfg'),File.join
(RAILS_ROOT, 'vendor','plugins',plugin_name,'plugin_cfg')
FileUtils.cp File.join(File.dirname(__FILE__),'lib','mr_bogus.rb'),File.join
(RAILS_ROOT, 'vendor','plugins',plugin_name,'lib','mr_bogus.rb')
FileUtils.cp File.join(File.dirname(__FILE__),'generators',plugin_name,
'mr_bogus_generator.rb'),File.join(RAILS_ROOT,'vendor','plugins',plugin_name,
'generators',plugin_name,'mr_bogus_generator.rb')
Dir.mkdir("#{RAILS_ROOT}/vendor/plugins/mr_bogus/generators/mr_bogus
/templates/controllers") unless File.directory?("#{RAILS_ROOT}/vendor/plugins
/mr_bogus/generators/mr_bogus/templates/controllers")
Dir.mkdir("#{RAILS_ROOT}/vendor/plugins/mr_bogus/generators/mr_bogus
/templates/views") unless File.directory?("#{RAILS_ROOT}/vendor/plugins/mr_bogus/generators
/mr_bogus/templates/views")
FileUtils.cp File.join(File.dirname(__FILE__),'generators',plugin_name,'templates',
'controllers','bogus_controller.rb'),File.join(RAILS\_ROOT,'vendor','plugins',
plugin_name,'generators',plugin_name,'templates','controllers','bogus_controller.rb')
FileUtils.cp File.join(File.dirname(__FILE__),'generators',plugin_name,'templates','views',
'bogus.rhtml'),File.join(RAILS_ROOT,'vendor','plugins',plugin_name,'generators',
plugin_name,'templates','views','bogus.rhtml')
```

An uninstallation file named "uninstall.rb" for removing the plugin created. Here is an example:

```
require 'fileutils'
plugin_name = ARGV[1] #mr_bogus
earth_root = ARGV[0]
puts "Uninstalling the generator"
system "ruby #{earth_root}/script/destroy #{plugin_name} #{plugin_name}"
puts "Uninstalling the plugin"
system "ruby #{earth_root}/script/destroy plugin #{plugin_name} --with-generator"
```

```
system "rm -r #{earth_root}/vendor/plugins/#{plugin_name}"
```

4.2 Installation

To install and activate the plugin, simply execute the following commands:

```
ruby install.rb <root_earth> <plugin_name>
ruby <root_earth>/script/generate <plugin_name> <plugin_name>
```

4.3 Uninstallation

To uninstall, execute the following command:

```
ruby uninstall.rb <root_earth> <plugin_name>
```