Infopark CMS Fiona

Rails Connector for CMS Fiona

Infopark CMS Fiona

Rails Connector for CMS Fiona

Contents

1

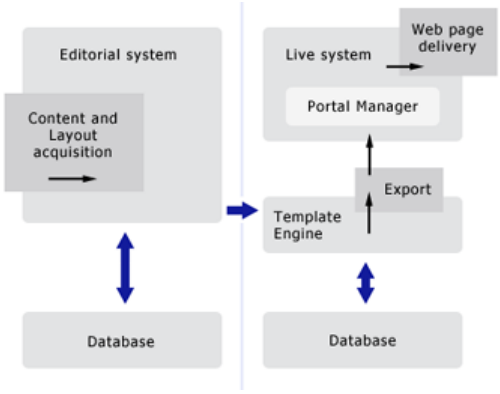
1 Concepts

1.1 Moving to Rails?

The following explanatory notes are meant to help you understand the differences between a classical CMS setup with Infopark CMS Fiona and one based on Infopark Rails Connector. Read on if you wish to evaluate whether to move to Rails or not.

1.1.1 Editorial and Live System

Infopark CMS Fiona consists of an editorial and a live system. The editorial system serves to create the content. Furthermore, administrators can use it to define data structures such as fields, file formats, etc. and adapt them according to the requirements of the website.



The task of the live system is to generate web pages from the content. In a classical setup, the layouts (templates) stored in the CMS are used for this. In layouts, the structures of the web pages are defined, independently of the content. By means of a special query language for layouts, the content is inserted into the layout during the so-called export. The result of this is the final product, the web pages.

1.1.2 Editorial Content and Layout are Handled Equally

Technically speaking, layouts are a special kind of content, namely one that is used to display the editorial content. Layouts as well as the editorial content are maintained in the editorial system. There you can edit the layouts, move them to a different position in the file hierarchy, enter a title or any other field value, etc.

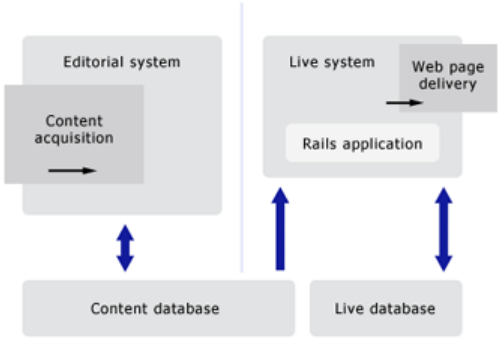
The unified handling of layout and editorial content is a central concept of Infopark CMS Fiona. Draft versions and released versions also exist for layouts, users can utilize the same tools for both content types. Layouts can be archived, are subject to the same workflow mechanisms as content, etc.

1.1.3 The Classical Live System is heterogeneous

The layout language of CMS Fiona makes it possible to use the functions of the live server in an easy and convenient way. Portlets, for example, can be included in web pages by means of a single layout instruction. The same applies if page access is to be restricted to particular user groups.

However, this approach requires several, partly disparate technologies on the live side. Updated content not only needs to be transferred to the live server for export. For implementing dynamic servers including personalization, access control, portlets, maybe also JSPs, the content needs to be delivered by the Portal Manager, processed by the Velocity Engine, the JSP Engine, and various filters. Using technologies as variable as these may quickly lead to unclear server setups on the live side.

1.1.4 Ruby on Rails is an Integrated Framework



In contrast to this, Ruby on Rails is an integrated framework. By means of Infopark Rails Connector, content stored in the CMS database can be processed and displayed. On the live side, a setup that basically consists of Ruby, Rails, Infopark Rails Connector, and the Search Engine, replaces the Template Engine, the Trifork application server, and the Portal Manager running in the latter.

The editorial system, on the other hand, remains unchanged. It is still required for comfortable content acquisition, for versioning, workflow control, etc.

Next to the database in which the editorial content is stored, Rails applications often utilize a live database for user-generated content such as comments, ratings, or personal preferences. For developers, access to the database content is transparent. The content display is elegantly encapsulated in views. This makes Ruby on Rails an ideal tool for efficiently developing modern Web 2.0 applications.

1.1.5 Moving to Ruby on Rails

Consider changing from a classical server setup with CMS Fiona alone to CMS Fiona plus Ruby on Rails if you plan to relaunch your website and make it more user-interactive. Ruby on Rails projects are handled just like software projects. Infopark offers to completely manage and supervise such projects.

1.1.6 Frequently Asked Questions About Moving to Rails

How complicated is moving to the Rails technology?

Before Infopark Rails Connector can be used, all layouts and custom functionality (such as PHP or Java code, portlets) need to be rewritten. The expenditure for this depends on the complexity of your system.

The content, on the other hand, can be taken over completely. In particular, editors can continue to use the system for creating content while the layouts and the functionality are adapted to Rails behind the scenes. As soon as the latter has been completed, the current version of the content can be easily delivered using the new Rails server.

Do I require a Template Engine if I use the Rails Connector?

No, the Template Engine can no longer be used in a sensible way.

Do I require Trifork or any other Java application server if I use the Rails Connector?

Not on the live server, but still on the editorial system since the GUI is implemented in Java and requires the Trifork or any other comparable application server to run.

Is Java still supported in general?

Yes, the Java Portal Manager is still expressedly supported. However, we recommend that new customers work with Rails and not with Java.

Can we continue to use our portlets?

If you wish to run your live server with Rails and the Rails Connector, portlets currently cannot be used.

Can we work both with Rails technology and Java?

You can – even if this should rarely be necessary – deliver the content of two Fiona instances using different technologies. However, it is not possible to combine these two technologies in one CMS instance.

When changing over to Rails, your editorial system and your website can stay in operation because the Rails application can be developed in parallel to running the existing (classic) Fiona system. During development, live data can be used for testing the rails application which then, finally, completely replaces the classic live system.

Do we still require Apache web server for delivering the web pages?

Yes. Apache web server ensures that all incoming HTTP requests are properly redirected. Apache web server forwards these requests to one or more Mongrel servers (possibly located on different machines). Mongrel is for your Rails application what Trifork is for Java applications. It processes requests and returns the result via the Apache server.

Which databases are supported in conjunction with the Rails Connector?

Currently, only MySQL databases are supported.

What is the role of the Search Server in conjunction with the Rails Connector?

The Rails Connector supports the use of the Search Server. Therefore, the latter can be installed and operated as usual.

Do I require a version control system for my Rails project? If so, which one is recommended?

Rails projects cannot be handled professionally without a version control system. Like numerous other companies, Infopark uses Git. Subversion is another often-used VCS.

Where can I find information about Ruby on Rails?

The home page of Ruby on Rails is http://www.rubyonrails.org/.

1.2 What Is Ruby on Rails?

Ruby on Rails is a framework with which dynamic websites based on content stored in databases can be developed in short time. The framework encapsulates access to the database and request processing. Thus, it saves a large amount of time required for developing the corresponding code in other environments.

In Ruby on Rails projects the functions offered by portals, for example, are considered more important than in classic projects focused on static content. This makes Ruby on Rails the ideal tool for Web 2.0 applications and their community functions such as blogs, rating, commenting, etc.

A Ruby on Rails runtime environment is a closed system composed of the Ruby interpreter and numerous scripts, helper programs and libraries. A Rails application consists of a directory tree containing the configuration, the views, the controllers, the database model, and other things.

Due to the Model-View-Controller architecture, i.e. the clear distinction between the data, its processing, and its display, the web applications are easy to maintain. In the following, the terms used in this architecture are briefly explained.

Model

The data model, i.e. the structure of the data and its interrelations, are mapped in Ruby on Rails to classes, methods and properties. A class normally corresponds to a database table, a row in it to an object, and an object's properties to fields. Due to this object-relational mapping of data to the object-oriented model of the programming language, the handling of data is consistent with the handling of the built-in and programmed classes and objects.

View

A view is what you use to make your data visible. Typically, a view contains HTML text, into which Ruby code is embedded by means of processing instructions, analogously to PHP, for example. The contents of a shopping cart could be a view, a particular website section as well.

Controller

A controller is a Ruby file that is executed if a request whose URL contains a particular component is to be handled. If, for example, a customer opens his shopping cart on a website, the controller recognizes this by identifying the corresponding URL component (`cart`, for example), fetches the associated data from the database and prepares it for the view. Then, the view associated with this controller is processed and the HTML page generated in this process is delivered.

1.3 Functions of Infopark Rails Connector

Infopark Rails Connector makes content that is maintained with Infopark CMS Fiona available to Rails applications.

- Infopark Rails Connector is delivered as a gem that can be integrated into an existing Rails application. It includes model classes for accessing CMS content and offers support for implementing controllers, helpers, and views for content delivery.
- From version 6.6 of Infopark CMS Fiona, your Rails application can be viewed in the preview of the editorial system so that editors can view their content in the current layout. Editing elements (for editing CMS content directly) can be integrated into the layout.
- As web pages are delivered, the Rails Connector is able to take account of the access permissions granted in the editorial system. If the website visitor is not logged-in to the site, an error page can be delivered which can be designed as required.
- Furthermore, Rails Connector supports deploying your Rails application to different systems (editorial, staging, live system). For this, a script for Capistrano is supplied.
- Marker menus enable edit markers and action markers for the preview to be organized into logical groups, thus simplifying the editing process. Marker menus expand and collapse so as not to disrupt the layout in the preview.
- Rails Connector makes it easy to add business functionality to a website and to combine editorial content with user-generated content.

For information about the website features of the Rails Connector, please refer to the description of the *Playland* demo application. Some features need to be activated explicitly. For details about how to do this, and how to use the APIs, please refer to the Rails Connector RDocs.

1.4 Usage Scenarios for the Rails Connector

For using Infopark Rails Connector, several scenarios should be considered. They are briefly presented in the following:

- **Preview in the CMS**
  - For the preview in the editorial system you require a Ruby on Rails installation and the Rails application that delivers the preview pages. This application should be taken from a versioning system.
  - For the preview to work on the user side, you require a web server that redirects the GUI and the preview URLs to the respective server. This redirection needs to be configured in the webserver. Furthermore, the GUI needs to be configured so that it generates correct preview URLs.
- **Development and layout**

  The layout of the web pages is based on the views of the Rails application. Normally, the following steps need to be made to set up the environment for developing a Rails application.
  - Install Ruby on Rails on your machine. CMS Fiona is not required because you do not need to create content.
  - Create a new Rails application or take it from the versioning system.
  - In this application, install the Rails Connector, if you have not already done so.
  - Install a database on your machine and import a dump you made from the database on the editorial system. Alternatively, direct access to the database on the editorial system can be set up.

- Set up the database model of your application so that it can access the data in the database.
- In the application, write or modify the controllers and the views, i.e. add functionality (business logic) and layout (presentation logic) to the application.
- Update the repository of the version control system in regular intervals to make your code available to others working on the same project.

- **Testing the web application**

  Normally, professional Ruby on Rails web applications are accompanied by test programs to ensure that the controllers and views behave as expected. The test programs require their own database into which the test data are written. Setting up the tests is part of the development process. Normally, the tests are run on the development systems.

- **Live (production) system**

  On the live system you require the ruby on rails runtime environment, the web application including the Rails Connector for Fiona, and the database containing the CMS content. The CMS itself is not required on the live system. On the live system, the content can be updated by means of replication mechanisms, if keeping the live side up-to-date matters. However, it is also possible to transfer the content in the form of a database dump from the editorial system to the live server and restore it there. For updating the web application itself, deployment tools such as Capistrano are available. You might also want to set up another database for storing user-generated content.

Environments

By default, every Rails application has three so-called environments that support the use of the application in different settings.

An environment is a named configuration set. One of the names available can be passed to the Rails server application. The configuration set specifies the database to use and provides global configuration parameters for controlling particular aspects of the application's behaviour. The environments available as a default are `development`, `test`, and `production` (live system).

When using the Rails application for previewing CMS content, usually another environment is added to the application, the `preview` environment. Next to the database to use, the `preview` environment specifies that the draft versions of the CMS files should be delivered instead of the released versions.

Database use

Rails applications can be operated with several databases – for the Rails Connector this is the rule rather than the exception because both the editorial content and the visitor-generated data (comments, customer data and the like) are used on the live server. Since the editorial content is never modified on the live side, as opposed to the data originating from the visitors, two databases are used. This is done for clarity as well as for stability and data security reasons.

All databases can be set up in the `config/database.yml` file of the Rails application. Typically, a Rails application uses the database connection named after the environment. The Rails Connector uses the connection named `cms`. You do not need to modify existing connections, just add the new section named `cms`.

1.5 The Playland Demo Application

For demonstrating some of the possibilities of a Rails application in conjunction with Infopark CMS Fiona, Infopark provides the *Playland* Rails application.

Unless otherwise stated, the functions described below are provided by the Rails Connector itself, not by third-party software.

To install Playland, please proceed as described in the instructions part of this Rails Connector documentation.

In Playland, Infopark has exemplary implemented several modern features: Access control (in the *Investor Relations* section), rating, voting, commenting, a wiki (based on third-party software), among others. These features help to make Playland livelier because they are oriented towards interaction with the website visitors. The implementation also demonstrates how user-generated content, e.g. comments, can be processed and displayed.

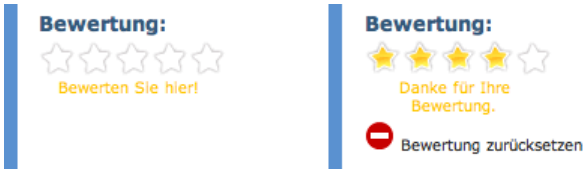
Website functions such as rating, voting , and commenting are not solely meant to provide feedback to the company operating the website (e.g. for the purpose of improving the site or the company's products). For this, much more precise instruments exist. Instead, these functions mainly have a community effect. From the number of ratings given by visitors, from the number of votes in polls, or from the postings in a forum (not included in Playland), a visitor can see how popular a site is. Visitors new to the site recognize that a community has formed around the company operating the website. The existence of a community makes a company more attractive because the company must have been accepted first. To non-expert first-time visitors a company with a large community is more likely to be helpful with issues related to the company's business.

If such features are used properly, new visitors or those just happening to be passing may feel inclined to spend a little more time on the company's offerings. Generally speaking, the community features

available to visitors serve as a platform for exchanging ideas or criticism, for example, or for the provision of information.

Rating

A rating function enables the visitor to rate the entire site or individual pages. To do this, the visitor clicks an icon to communcate the mark the page deserves. Sometimes, the website operator asks a question that narrows down the aspect to rate: "How do you like the way the content is presented on this page?", "How do you like this article?", "How helpful is the information on this page?". From the ratings the average mark is calculated and displayed. The option to reset the mark is only available to logged-in administrators.



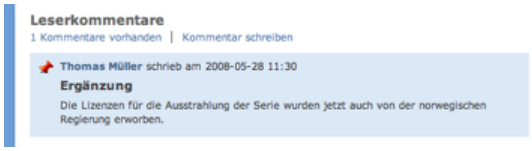
The more restrictive the rating function is handled the more credible is the operator's implicit interest in the visitors' feedback. For this reason, the rating function should only be available to registered visitors. Also, multiple uses of this function by the same visitor should only correct the mark given before, instead of counting as new marks.

In general, the average marks are not differentiated enough to conclude from them how visitors rate the quality of a web page. However, if the marks are linked to individual visitors (this can be achieved by means of the OMC Connector), targeted measures can be initiated.

In Playland, ratings are user-generated content. The marks as well as the CMS objects to which they refer are stored in the live database. The rating element can be individually switched on or off for every CMS object. To process the ratings, e.g. to associate them with the OMC customer accounts concerned, additional code is required.

Commenting

The commenting function enables website visitors to attach notes, critcism, enhancement proposals, questions, etc., to a page. Usually, the comments are visible to all visitors to encourage discussions about the content offered on a page or website. Mostly, comments are displayed below the content:



To reduce the risk of abuse as regards content, appropriate protection measures, e.g. moderation, should be implemented. In principle, it is advisable to have a notification e-mail sent to a moderator after a new comment has been posted. Also, spam should be avoided. Additional functions like these are not implemented in Playland.

The website operator should analyze the contributions at regular intervals. For one thing, contributions sometimes provide valuable information that can be used to improve a web page, the website as a whole, or the products or services offered. Furthermore, escalating discussions can be detected and stopped early.

The number of comments a visitor may post is not limited. The comments are stored in the database in the order in which they are posted. Every comment is associated with the CMS object to which it refers as well as the required user data.

Personalization via OMC Integration

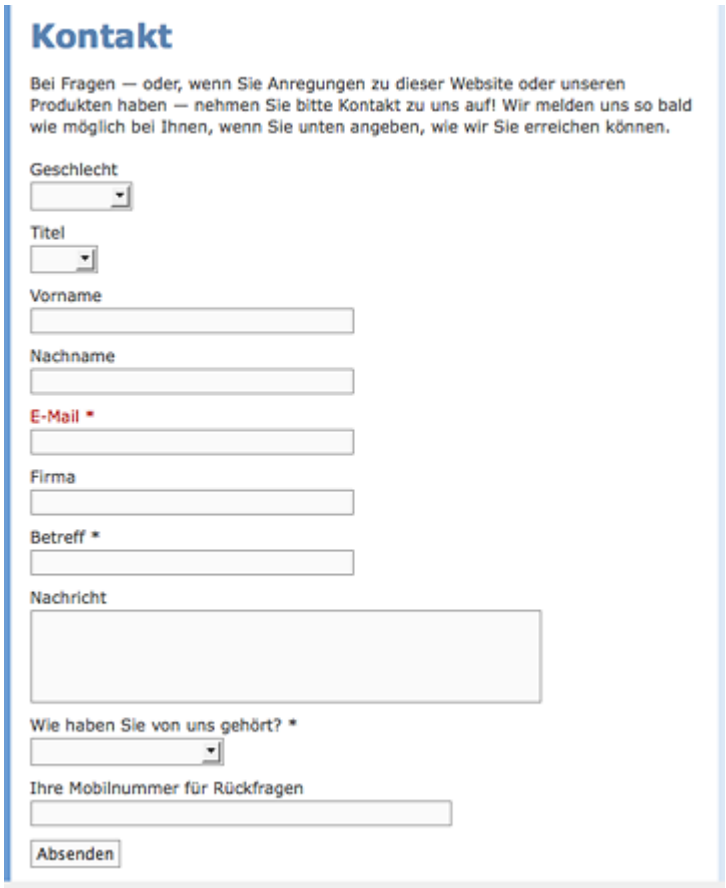
Next to the Rails Connector, this functionality requires the `infopark_omc_connector` *gem.*

More and more websites offer personalization options to strengthen customer loyality, for example. There are several other benefits for website operators: Premium customers can be given access to premium content. Users are enabled to keep their account data up-to-date. Using a suitable backend for storing the account data and other information helps to keep the data consistent.

Personalization requires an infrastructure that enables users to register, log in and out, and to change or reset their respective password.

Infopark Rails Connector provides this infrastructure. As the backend, Infopark Online Marketing Cockpit (OMC) is used. Playland utilizes the functions and views supplied by the Rails Connector to connect the Rails Connector with the OMC. User roles defined in the OMC control the visibility of (premium) content and the permission to delete comments.

Based on activity types in the OMC, you can have the Rails Connector create forms. For every field of the desired activity type (or a selection of these fields), a corresponding input field as a part of the form is created:

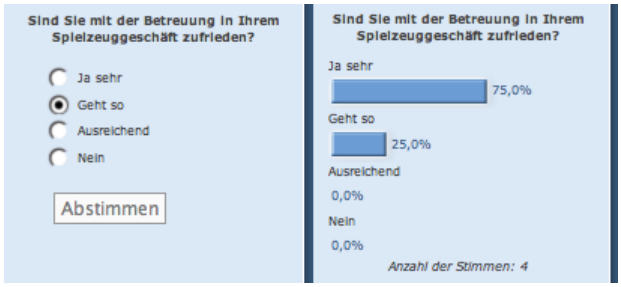


When a logged-in user submits such a form, the Rails Connector can be used to create an activity in the OMC. This activity includes the field values of the form. Thus, numerous procedures (event participation, registration, support cases, etc.) can be handled on the website. The data involved can be transferred to the OMC for further manual or automatic processing. Additionally, this data can also be used for event management purposes, mailings, etc.

Polls

This functionality is not supplied by the Rails Connector but has been directly implemented in Playland.

By means of polls, website operators can get at the website visitor's opinions. Usually, a poll consists of a question and several answers to it of which the visitor chooses one.



If votes have been casted, a chart illustrates the percentages of the votes for the individual answers. Administrators can easily create, edit or delete polls.

With clear questions, a poll might produce useful results. However, the voting results are usually not representative and can easily be misinterpreted.

Technically, the casted votes are handled and processed like the marks given in ratings, meaning that the visitors' input is stored in the live database for later analysis.

Wikis

This functionality is not supplied by the Rails Connector but has been directly implemented in Playland.

A wiki is a platform on which simple-structured web documents can be created and maintained by several authors. The attraction of a wiki lies in its ease-of-use. It can be opened to a large group of users who are enabled to produce a large amount of information in short time. Wikipedia is the best example of this. A wiki's openness, though, may cause content quality to vary and inconsistencies to sneak in. However, a wiki is a fast publishing medium most suitable for closed user groups (such as intranets) and for internal documentation purposes.

In Playland, wikis can be inserted into the structure of the CMS content using placeholder objects. Thus, a wiki can show up in navigations, it can act as link destinations, etc. In The *Playland* Rails application, wikis are made available by means of a plugin.

Other Website Features

Search
A core aspect of any website is the navigation. An integrated search provides a site with extra navigational functionality, allowing users to quickly find the information they need. Requires Infopark Search Server.

PDF-Generator
By means of the PDF Generator, PDF documents can be created on the fly from web content so that the visitors can be provided with offline content.

Time Machine
Many websites change regularly. The Time Machine makes it possible to inspect the state of a website at an arbitrary date in the future.

RSS feeds and podcasts
RSS feeds and podcasts help your visitors to automatically stay informed about news on your website.

Search Engine Optimization (SEO)
By providing metadata exclusively for search engines, the search engine rating of a website and its visibility can be noticeably improved. Additionally, a sitemap XML document can be generated automatically to aid search engines in the indexing of your site.

Tracking
Google Analytics provides a powerful tool for collecting various data based on your users' site visits. The tracking functionality offered in conjunction with Infopark Online Marketing Cockpit enables you to also track known users on your site, which offers a wide range of website optimization and content personalization options.

1.6 The Layout of a Rails Application

Rails applications are not based on the layout files stored in the CMS but on views which are an essential part of the Rails application itself. Views are HTML files that contain code with which the content can be dynamically included in the web pages to be displayed.

In a Rails project, a designer creates the layout of the web pages. The layout is based on Rails layout files that contain the HTML framework of the web pages to be delivered, i.e. templates. These templates contain placeholders that the Rails server replaces with content later on. What is inserted into the templates depends on the views used, and the view is determined by the URL the visitor has opened.

The placeholders in the layout include a main section (mostly for displaying the main content) as well as other named sections which are optional. The view supplies the content for all sections. Thus, it also contains a main section as well as the optional named sections. If a layout contains a named section for which no corresponding section exists in the view, the section remains empty in the generated web page.

To create or modify the design, the designer adapts the layout and the views to make them generate pages that meet the design specification. Typically, the layout references style sheets whose classes are used both in the layout and the views. The data for which the view is executed is rendered by means of Ruby code the designer embeds into the layout and the views.

From version 6.7.1, Infopark Rails Connector supports two template languages for writing views and layouts: ERb (Embedded Ruby) and Liquid.

The ERb template language is part of Ruby's standard library. When using ERb, Ruby code is embedded directly into views and layouts. CMS content within ERb templates can be displayed using Infopark Rails Connector. The file name extension of ERb templates is `.html.erb`.

Liquid is a template language implemented in Ruby with a focus on security and robustness. Liquid templates have a simple syntax and are therefore simpler to write with less danger of making errors. Infopark Rails Connector expands upon Liquid's basic syntax in order to facilitate displaying CMS content. The file name extension of Liquid templates is `.html.liquid`.

In a Rails application, views and layouts can be written interchangeably in Liquid or ERb.

1.7 Deployment

When developing and deploying Rails projects, it is recommended to use versioning software (e.g. Subversion or Git) plus the Rails deployment program Capistrano.

Especially with large projects, versioning software makes it much easier to maintain the rails application. Since all application components that are subject to the developing process are stored centrally in the repository of the versioning application, a clear definition exists of what the current version is and how it can be retrieved.

For the individual designer or programmer, the developing process consists of a sequence of identical cycles: checking out, editing, testing, checking in.

Capistrano deploys your web application to the target systems such as the staging or live server. For this, it uses the current version of the web application located in the repository of the versioning application and a couple of standard Unix programs such as ssh.

Capistrano uses a configuration file to determine the actions to perform. The initial configuration for a Rails project can be generated with Capistrano itself. The generated configuration is suitable for the classical development situations and can easily be adapted or extended by an administrator.

1.8 CMS Layouts as Extensible Page Types

Infopark Rails Connector makes it possible to access CMS data from within a Rails application. For this purpose, the Rails Connector provides a variable, `@obj` that represents the file to be delivered. You can, for example, access the value of the `abstract` field using `@obj.abstract`.

`@obj` is of the `Obj` class. Up to and including version 6.7.2, the Rails Connector automatically creates subclasses of `Obj` based on the file format name so that, for example, a CMS file that has the `NewsArticle` format automatically becomes an instance of the `NewsArticle` class.

From Version 6.7.3, subclasses of `Obj` are no longer created automatically. Thus, all CMS files are instances of the `Obj` class. If, however, the application includes a class that is based on `Obj` and has the same name as a file format, all files with this format automatically are instances of this class.

Layout-specific classes can be used to extend CMS files to which a particular file format was assigned with new properties and methods. If, for example, all CMS files with the Product format should be ratable, you can define the Product model as follows:

```
# This Model describes the behavior of all CMS objects of the Product class.
class Product < Obj

  has_many :ratings, :dependent => :delete_all, :foreign_key => :obj_id

  def rate(score)
    rating = ratings.find_by_score(score) || ratings.build(:score => score)
    rating.count += 1
    rating.save
  end

  def count_for_score(score)
    rating = ratings.find_by_score(score)
    rating ? rating.count : 0
  end

  def rated?
    !ratings.empty?
  end

  def average_rating
    raise TypeError unless rated?
    sum, count = ratings.inject([0, 0]) do |(sum, count), rating|
      [sum + rating.score * rating.count, count + rating.count]
    end
    sum.to_f / count.to_f
  end

  def average_rating_in_percent
    if rated?
      (100 * average_rating / Rating::MAXIMUM).to_i
    else
      0
    end
  end

  def reset_rating
    ratings.clear
  end
end
```

Ruby requires that class names follow the CamelCase notation, meaning that they must start with a capital letter. We strongly recommend to rename file formats in the CMS so that they follow this convention. The affected file formats can be determined using the `check:obj_classes` Rake task (included in the Infopark Rails Connector plugin). Afterwards, the formats can be renamed using the renameObjClass Tcl command.

1.9 Dedicated Controllers for Specific CMS File Formats

With Rails Connector (version 6.7.3 or higher) you can use dedicated controllers for every CMS file format. Creating a controller for a specific file format only requires that the controller class is created.

Common use cases are:

- Organizing a large number of views into folders according to file formats
- Providing multiple switchable views for a given CMS file
- Processing session data, cookies, or URL parameters for specific CMS file formats
- Using Rails' autoload mechanism for the helper modules needed in the current context (without `helper :all`)
- Handling `POST` requests to implement, for example, web forms via CMS files
- Delivering CMS data in several different formats, such as XML, HTML, JSON

For every CMS file to be loaded, the Rails Connector looks for a controller whose name matches the name of the current CMS format. For example, a file with the `Publication` format would be delivered by a controller named `PublicationController` that descends from `RailsConnector::DefaultCmsController`. If no such controller exists, `CmsController` is used as a default.

The `index` action is still used as the default action to deliver CMS data. While custom actions can be defined, they can only be evoked by means of custom routes. These routes need to be used explicitly in order to generate links to CMS files using other actions than `index`.

The Rails Connector provides a special method that is useful when writing tests for CMS related controllers. Use `request.for_cms_object` to define which CMS object should be loaded in your test. The following example uses the `rspec` test framework:

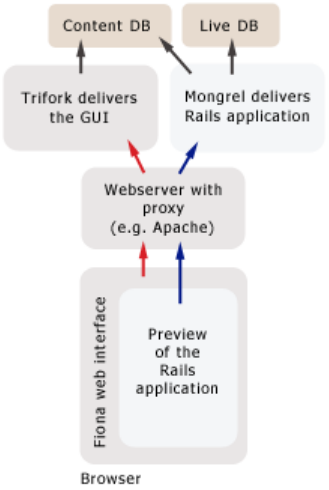
```
describe CmsController, "request for HTML document" do
  before do
    controller.stub!(:ensure_object_is_permitted).and_return(true)
    controller.stub!(:ensure_object_is_active).and_return(true)
    controller.stub!(:set_google_expire_header).and_return(true)
    @obj = mock_model(Obj, :mime_type => 'text/html', :permalink => 'dummy')
    request.for_cms_object(@obj)
  end

  it "should render the view" do
    get 'index'
    response.should be_success
  end
end
```

1.10 Using the Rails Application as a Preview Server

In the standard installation of Infopark CMS Fiona, the preview is delivered by the Infopark Portal Manager. From version 6.6 of the CMS you can alternatively integrate a Rails application as the preview, so that the editors can view the content they create in the layout of the Rails application..

With websites realized as rails applications, the web pages are delivered dynamically. The Rails server generates the requested pages on-the-fly and uses for this the content currently present in its database. If the databases of the editorial and the live system are synchronized, your website will always be highly up-to-date. As an option, caching mechanisms might be used to reduce the system load.

Releasing a document in the editorial system causes this document to be published immediately. The export facilities of the CMS are not required if the content is delivered by the Rails server. Accordingly, layout files are not required in the CMS.



If a Rails application is used for previewing the content, two servers are used, one for the GUI pages (Trifork) and one for the preview pages (Mongrel). The servers communicate with each other. To the client, the servers need to appear as one because for security reasons most browsers do not permit XSS.

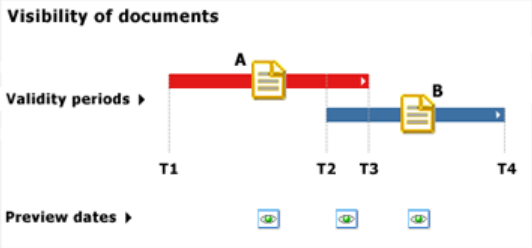
For this reason an Apache HTTP server is placed as a reverse proxy in front of the servers mentioned above.

Section Integrating the Rails preview into the GUI explains how the Rails preview can be enabled.

1.11 Previewing Content in the Future (Using the Time Machine)

For editors, the Rails Connector's "Time Machine" offers the possibility to see what the content would look like in the future. This is convenient if you have documents with different validity periods and if you would like to check whether your site contains the desired pages (and only these) at a given point in time.

If, for example, a document, "A", is valid from T1 to T3, and another document, "B", from T2 to T4 (the periods overlap), then you can use the time machine to check what is displayed between T1 and T2 (only "A" is visible), between T2 and T3 (both documents are visible), and between T3 and T4 (only "B" is visible):



You can select the preview date by means of a link on the preview page, provided that the link was integrated into the preview page. The link opens a window which contains a slider and a calendar:

Both controls can be used to select the desired preview date. If you let go of the slider or if you click the desired date, the selected preview date is set and the page is refreshed. Afterwards, the current page is displayed in the context of the selected point in time.

1.12 What Are Permalinks?

Permalinks are persistent (non-changing) identifiers for CMS files. On websites delivered with a Rails application that includes the Rails Connector, web pages can be opened by specifying the associated identifier in the URL, independently of its actual storage location in the CMS. Examples:

```
http://www.myserver.com/features
http://www.myserver.com/news/2011-07-27/current-release-is-available-for-download
```

Permalinks are file fields, meaning that they are not part of the content. After enabling permalinks in the GUI, a permalink can be given to each file.

1.13 Generating PDF Files

Infopark Rails Connector includes a component, the PDF generator, for creating PDF files from within a Rails application. For this, a web service interface is used.

The PDF generator makes use of Apache FOP (Formatting Objects Processor), a Java application that translates an XML file into a PDF file. The rules for generating the layout are taken from an XSL style sheet. A sample stylesheet is included.

The PDF generator has two modes, an external and an internal mode.

The External Mode

In the external mode, external sources can be specified for both the XML and the XSL file. This allows for using a different application (a PHP application, for example) als the source.

The Internal Mode

The internal mode, on the other hand, makes it possible for the programmer to generate PDF files from CMS files. A flexible and extendible mechanism is used to achieve this.

The PDF generator is deactivated by default and needs to be configured first. Afterwards, it can be used without further effort since all the required files are included.

1.14 Searching Documents Using the Search Server

Infopark Rails Connector installs a search controller for sending search queries to Infopark Search Server. By means of a view for this controller the search results pages are formatted and the hits are linked.

For being able to perform searches, Infopark Search Server as well as the indexes to be searched need to be made available to the rails application. For this, two approaches exist:

- The Search Server running on the editorial system is also used for searches on the live side. This solution is only recommended if the availability of the editorial system is not impaired by the additional load caused by searches on the live system.

  Even if the content database on the live system is kept up to date (for example by short replication cycles), discrepancies may arise between the indexes on the editorial system and the actual content on the live system (changed content might have been indexed before it shows up in the live database). If absolute synchronicity is required, this solution cannot be used.
- The live side is provided with a dedicated Search Server, possibly on a dedicated machine if high load is an issue.

  This solution requires that the search indexes are replicated to the live server in the same intervals as the content database.

1.15 Writing Integration Tests for a Rails Connector Project

Integration tests are an important tool used while developing a Rails application. They help to ensure that new functionality that is being developed does not inadvertently damage already existing functionality.

Ruby on Rails offers the possibility to write unit tests, functional tests and integration tests for the Rails application. The integration tests of a Rails application are normally located in the `test/integration` directory of the application.

Up to version 6.6.1 of Infopark Rails Connector, integration tests can only performed after the test environment including its database connection have been set up manually.

From version 6.7.0, integration tests are run in the `test` environment. Since all environments use the same database – the database which contains the productive data – real CMS data is used in the integration tests. Thus, the tests can prove whether the Rails application processes the data as planned.

To ensure that programming errors in an integration test do not modify or corrupt CMS data, the Rails Connector should connect to the database only as a user without write permission.

1.16 Recommended Literature About Ruby on Rails

To those who would like to know more about Ruby on Rails, MySQL, and related topics we recommend the following literature:

- Programming Ruby
  (ISBN-13: 978-0974514055
- Agile Web Development with Rails
  (ISBN-13: 978-0977616633)
- Rails Recipes
  (ISBN-13: 978-0977616602)
- Rails Cookbook
  (ISBN-13: 978-0596527310)
- Ruby Cookbook
  (ISBN-13: 978-0596523695)
- Ajax on Rails
  (ISBN-13: 978-0596527440)
- High Performance MySQL
  (ISBN-13: 978-0596527082)
- Building Scalable Web-Sites
  (ISBN-13: 978-0596102357)
- Ruby Pocket Reference
  (ISBN-13: 978-0596514815)
- Capistrano and the Rails Application Lifecycle (eBook / O'Reilly Shortcut)
  (ISBN-13: 978-0-596-52962-8)
- Mongrel (eBook / O'Reilly Shortcut)
  ISBN-13: 978-0-59-652854-6

2

2 Instructions

2.1 Conventions

The following conventions apply to all command lines shown in the Rails Connector instructions:

- `#` - An administrator's command promt character. Run this command as `root`.
- `$` - A general command prompt. You can run this command as any user.
- Required user input is inidcated by a blue font color:

```
# gem install mysql
```

- Command output appears in a black font:

```
Select which gem to install for your platform (i686-linux)
```

- Comments are displayed in red font color and are enclosed in round brackets:

```
(Please also enter all line breaks as shown.)
```

- Paths are always displayed using the Unix path delimiter `/` (slash). On Windows, please replace it by `\` (backslash).

2.2 Installation Requirements

2.2.1 Hardware

We recommend to use dedicated computers for application development, for the CMS server, the live server, and the staging server (if required). These computers should be based on the same hardware platform, and have the same operating software (such as the operating system and the database) installed. This makes it possible to recognize hardware and software problems in time, before live operation begins.

For development, testing, and staging, a standard PC is sufficient.

Normally, more powerful hardware is required for the live system than for the other systems mentioned. Rails servers can be scaled easily by improving the hardware.

2.2.2 Server-Side Software

For being able to use Infopark Rails Connector 6.7.3 you require the software listed below. Infopark recommends to always install gems as the same user, if possible. Otherwise, they will be placed in different file system locations which might cause Bundler to lose track of them.

1. For productive use, a standard Linux system is required (recommended: 64 bits). For development purposes, Mac OS X 10.6 (“Snow Leopard”) or 10.7 (“Lion”) can be used, too. However, Rails Connector versions up to and including 6.7.1 are not compatible with Mac OS X 10.6 or later.
2. Ruby 1.8.7, patch level 174 or later, including OpenSSL support and the shared ruby library
3. RubyGems 1.3.7. RubyGems will install required third-party software while Infopark Rails Connector is being installed. For this, a working internet connection is required.
4. Rails 3.0.5 and Bundler.
5. Infopark CMS Fiona, version 6.7.2 or later with a MySQL database. The database must have `storeBlobsInDatabase` enabled. The Content Manager (CM) must have been railsified.
6. Apache HTTP server, version 2.2 or later. Other web servers can also be used, provided they are able to work as a reverse proxy and as a load balancer. In conjunction with the Rails Connector, Infopark currently only supports the Apache server.

   The web server should support the XSendFile-Header. This is not an absolute necessity, but missing support for XSendFile will decrease performance.

   To enable XSendFile support for Apache HTTP server, please install the `mod_xsendfile` module.

   If applicable, activate the use of XSendFile in your Rails application.
7. If the Rails Connector is executed by Phusion Passenger, `mod_passenger`, version 2.2.9 or above, is required.

2.2.3 Client-Side Software

Editing elements (edit markers) are currently not supported in Internet Explorer 6. Marker menus, however, do work in this version of Internet Explorer.

2.2.4 Known Limitations

Rails or the Rails Connector for Infopark CMS Fiona currently have the following known limitations:

1. Mirror files are not supported.
2. If Rails and Infopark Rails Connector are operated on a 32 bit machine, date values from January 19, 2038 cause errors when the corresponding content is delivered. This is not the case on 64 bit systems.

2.3 Installing the Rails Demo Application Playland

This guide assumes that a working installation of Infopark CMS Fiona exists. Please also take account of the installation requirements. If you wish to run the supplied application tests you also require the libraries `libxml2-devel` and `libxslt-devel`.

1. Install the demo application and the components on which it depends

   - Install the Infopark Rails Connector gem, the MySQL blob streaming gem, and the OMC Connector gem as follows and in the order given:

     ```
     $ gem install path/to/mysql_blob_streaming-x.y.z.gem
     $ gem install path/to/infopark_omc_connector-x.y.z.gem
     $ gem install path/to/infopark_rails_connector-x.y.z.gem
     ```

     Other required software will be installed automatically.
   - Download the *Playland* demo application from the Downloads section of our website. From Fiona 6.7.3, *Playland* is part of the Rails Connector package.
   - Unpack the package into a directory of your choice. All path specifications below refer to this directory.
   - The package includes information about required third-party software. Change to the *Playland* directory and install this software:

     ```
     $ cd playland-x.y.z/

     # Installation including the above-mentioned libraries for testing
     $ bundle install

     # Installation without the above-mentioned libraries for testing
     $ bundle install --without=test
     ```

   - Copy your license file, `license.xml`, to the `config` directory.

2. Prepare OMC integration

   Please follow these steps if you wish to integrate the Playland demo application with Infopark Online Marketing Cockpit. Note that the login and logout functions of the Playland demo application can not be used without the OMC.

   - Register your website at reCaptcha. You will obtain a public/private key pair for use with the reCaptcha webservice. If you are already signed up for their service, you may skip this step.
   - Insert your reCaptcha key pair into the file `config/initializers/rails_connector.rb`:

     ```
     RCC_PUB="1331"  # public key
     RCC_PRIV="3113" # private key
     ```

   - Adapt the file `rails_connector.rb` to your OMC configuration:

     ```
     OmcConnector.configure do |config|
       port = ENV['OMC_PORT'] || 4000          # On which port is the OMC running?
       config.url = "http://localhost:#{port}" # On which host?
       config.login = "webservice"     # since 6.7.3
       config.api_key = 'geheim'       # prior to 6.7.3: password
     ```

```
   config.contact_roles_callback = lambda {|c| ['admins']}  # allows to remove
  comments
end
```

3. Prepare the database and the search for use with Infopark CMS Fiona

   It is recommendable to create a new CMS instance named `playland`, for example.

   For the Rails Connector to be able to connect to the database of the Content Manager, the CM database must be converted to MySQL.

   Afterwards, the database can be filled with content and optimized for the use with Infopark Rails Connector. For this, change into the directory of the CMS instance concerned and execute the following commands:

```
$ ./bin/CM -restore ../../share/demoContentDump
$ ./bin/CM -railsify
```

   In order to use the search functions of the Playland demo application, you will need to prepare the collections of the SES using these commands:

```
$ ./bin/CM -single < ../../share/configureInstanceForDemoContent.tcl
$ echo "indexAllObjects" | ./bin/CM -single
```

4. Create additional databases for the Playland rails application

   Open the MySQL command interface and create the required databases for live data such as ratings etc. For each environment (`development` and `test`) such a database is required:

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 5.0.41-community MySQL Community Edition (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> create database playland_development;
Query OK, 1 row affected (0.00 sec)

mysql> grant all on playland_development.* to fiona@localhost identified by 'fiona';
Query OK, 1 row affected (0.00 sec)

mysql> create database playland_test;
Query OK, 1 row affected (0.00 sec)

mysql> grant all on playland_test.* to fiona@localhost identified by 'fiona';
Query OK, 1 row affected (0.00 sec)

mysql> exit
Bye
```

5. Configure the database connections

   The Playland application uses two databases in each environment:

   - One database for the CMS content (`cms:`)
   - Another database for Rails-specific content (comments, page ratings, user session data etc.)

The configuration file for database connections, `database.yml`, is located in the `config` directory. Replace `database.yml` with a copy of `database.yml.template` and adapt the configurations to your setup.

For both databases, several configuration sets for different Rails environments exist. Which of them is used depends on the server's active environment. The database connection to the CMS content needs to be set up according to the general Fiona database configuration (*instanceName*`/config/cmdb.xml`).

6. Adapt the settings of the CMS instance

   In case you are not using the `default` instance, the following configuration settings need to be changed:

   - Specify the instance name in the `config/initializer/rails_connector.rb` file:

   ```
   RailsConnector::Configuration.instance_name = instanceName
   ```

   - Specify the HTTP port of the SES in the `config/initializer/rails_connector.rb` file:

   ```
   RailsConnector::Configuration.search_options = {
     ...
     :port => nnn5,
     ...
   }
   ```

7. Create the database structure in the Rails database

   Change into the root directory of your Rails application and initialize the database:

   ```
   $ rake db:migrate
   == CreateComments: migrating ==================================================
   -- create_table(:comments)
      -> 0.0461s
   == CreateComments: migrated (0.0467s) =========================================

   == CreateRatings: migrating ===================================================
   -- create_table(:ratings)
      -> 0.0064s
   == CreateRatings: migrated (0.0070s) ==========================================

   == AddSessions: migrating =====================================================
   -- create_table(:sessions)
      -> 0.0130s
   -- add_index(:sessions, :session_id)
      -> 0.0140s
   -- add_index(:sessions, :updated_at)
      -> 0.0156s
   == AddSessions: migrated (0.0442s) ============================================
   ```

   This `rake` command executes the ruby files located in the rails application directory `db/migrate` in order to create or modify the table structure of the databases of your rails application. Note that this does not alter your CMS installation or database.

8. Testing the application

   In `development` mode, start the Rails application server with the following command:

```
$ rails server
=> Booting WEBrick
=> Rails 3.0.5 application starting in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
```

Check if the application is running by opening the URL `http://localhost:3000/` in your web browser.

9. Integrate the Rails preview into the CMS GUI.

2.4 Configuring Database Connections

The database connection configuration is part of every environment. The configuration file `config/database.yml` holds the connection information:

```
my-project_environment:
  adapter: mysql
  database: rails-db
  username: rails-db-user
  password: rails-db-user-password
  socket: /var/run/mysqld/mysqld.sock
```

You can adjust the entries according to your needs. We strongly advise against using the same database for more than one environment.

The following commands can be used to create a MySQL database:

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 275
Server version: 5.0.37 MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database Rails-DB;
Query OK, 1 row affected (0.00 sec)

mysql> grant all on Rails-DB.* to 'Rails-DB-User'@'localhost' identified by 'Rails-DB-
User-Password';
Query OK, 0 rows affected (0.00 sec)

mysql> exit
Bye
```

2.5 Installing Infopark Rails Connector

Install Infopark Rails Connector in order to access and display CMS content directly from your rails application.

The following instructions refer to Infopark Rails Connector, version 6.7.3 upwards. Instructions for installing previous versions can be found in the corresponding manuals (PDF). For information about updating from an earlier version of Rails Connector to the current version, please contact Infopark Customer Support.

2.5.1 Prerequisites

1. At least version 6.7.2 of Infopark CMS Fiona is required.
2. Follow the instructions to install Infopark CMS Fiona. From version 6.7.3, Infopark's demo content (*Playland*) is only available in the Rails version which is part of the Rails Connector package.
3. Install the software required for the Rails Connector according to the instructions in the Installation Requirements section.
4. You will first need to install a MySQL database for your CMS Fiona instance, as SQLite databases are not supported by Infopark Rails Connector. To do this, please follow the instructions for integrating a different database.
5. Create a new Rails project and change to the project directory.

```
cd myproject
```

6. Copy your license file, `license.xml`, into the `config` directory below the project directory.

2.5.2 Installation

1. In addition to the database configuration for each environment, the `config/database.yml` file must contain the correct definition for the CMS Fiona database:

```
cms:
  adapter: mysql
  database: Fiona-DB
  username: Fiona-DB-Read-User
  password: Fiona-DB-Read-User-Password
  socket: /var/run/mysqld/mysqld.sock
```

   A database user with read-only access should be used for the connection to the Fiona database.
2. Get the Infopark Rails Connector package from the download section. Install the gems contained in it in the following order:

```
$ gem install infopark_omc_connector-x.y.z.gem
$ gem install mysql_blob_streaming-x.y.z.gem
$ gem install infopark_rails_connector-x.y.z.gem
```

   First, set up your application to use the above-mentioned gems by adding them to the `Gemfile`:

```
gem "infopark_rails_connector", "=x.y.z"
gem "infopark_rails_connector_addons", "=x.y.z"
gem "mysql_blob_streaming", "=x.y.z"
```

   As an alternative, you can also copy the `Gemfile` from the *Playland* demo application.
3. In addition to program code, the gems contain generators to enable the delivery of CMS-managed content, and to make additional features available. Execute the following command within your rails application:

```
$ rails generate rails_connector:install
```

This command copies several files to your application. The files are required, for example, for displaying edit markers, and for having the Time Machine and the Search available. Furthermore, a `rails_connector.rb` initializer file is created in the `config/initializers` directory, as well as the `configuration.rb` file in the `config/local` directory.

The Rails Connector uses the jQuery JavaScript library for creating and handling edit markers, comments, and ratings. If this library has not yet been installed into your application, the generator just executed will point to this fact. In this case, please install the library using the following command:

```
$ rails generate jquery:install
```

4. By means of the `rails_connector.rb` initializer file you can activate the add-ons you would like to use in your application, as well as specify the instance name of your CMS installation, for example.

   In case you are using a CMS instance other than `default`, provide the correct instance name by modifing the following line in the initializer:

   ```
   RailsConnector::Configuration.instance_name = 'default'
   ```

   By means of the `configuration.rb` file, the Rails project can be adapted to your local development environment.
5. Start the rails application server by issuing the following command:

   ```
   rails server
   ```

   The server will be available for browsing at `http://localhost:3000`.

2.5.3 Next Steps

1. To obtain detailed information about the Rails Connector's API, please read its RDoc. To do this, start an RDoc server with this command:

   ```
   $ gem server
   ```

   The command will start a gem server and print out a URL with which the documentation of all locally installed gems can be accessed. Open this URL in your browser (typically `http://localhost:8808/`) and navigate to `infopark_rails_connector` or `infopark_rails_connector_addons`. More information on running a gem server is available on the RubyGems site.
2. If you use the Rails application for previewing CMS content, configure the preview accordingly.
3. Please also configure the deployment for the different environments.

2.6 Updating a Rails 3 Application to Rails Connector 6.7.3

If you are running a Rails application with Infopark Rails Connector prior to version 6.7.3, please proceed as follows to upgrade your application to Rails Connector 6.7.3. The Rails application as such must have been upgraded to Rails 3 before the Rails Connector is upgraded to version 6.7.3.

As with previous versions, the Rails Connector includes a generator for installing the required files to a Rails application. This generator has been optimized for installing the Rails Connector to an empty Rails application. However, it can also be used for updating an existing application. The generator overwrites existing files only after the user has confirmed a corresponding query.

Please follow the steps below to upgrade Infopark Rails Connector to version 6.7.3:

1. Remove the following lines from the `config/initializers/rails_connector.rb` file:

```
config.inquiry = {
  :agent => 'name',
  :owner => 'name'
}
```

2. Remove the following call from the `config/routes.rb` file:

```
RailsConnector::Configuration.cms_routes(map)
```

3. Now, install the Rails Connector:

```
rails generate rails_connector:install
```

   The generator asks for confirmation before it overwrites existing files. Check the changes the generator intends to make by answering the confirmation question with ‘d’.

   The `public/javascripts/rails_connector/editmarker.js` file must be overwritten. The following files potentially contain application-specific changes that should not be overwritten without having looked at their contents first:

- `config/initializers/rails_connector.rb`
- `lib/obj_extensions.rb`
- `app/views/layouts/application.html.erb`

4. Install jQuery:

```
rails generate jquery:install
```

   The files `public/javascripts/jquery.js` and `public/javascripts/rails.js` should not contain application-specific code and need to be overwitten.

5. Comment out the following line in the `config/application.rb` file:

```
config.action_view.javascript_expansions[:defaults] = %w(jquery rails)
```

6. Remove the following files if they exist:

- `lib/tasks/rails_connector_addons.rake`
- `lib/tasks/rspec.rake`

Adapting Rails and the Rails Connector to the local development environment

Up to version 6.7.2 (inclusive), it was possible to override the Rails configuration settings in the `config/local/configuration.rb` file. The Rails Connector settings could be overridden in `config/local/initializer.rb`. The initialization process of the Rails frameworks required that the settings were located in different files.

This separation is no longer necessary. The files contained in the `config/local` directory may be merged or divided by individual aspects. All Ruby files in this directory are processed in the same initialization phase.

Localization files

From version 6.7.3, Ruby on Rails localization files are no longer included in Infopark Rails Connector packages. For running a Rails Connector application in a different language than English, localization files are required.

Session user attributes

From version 6.7.3, the attributes of a logged-in user to be temporarily stored in the session are no longer specified in the `UserController`. Instead, they need to be defined as follows in the `config/initializers/rails_connector.rb` file:

```
RailsConnector::Configuration.store_user_attrs_in_session =
  [:login, :first_name, :last_name, :email, :id]
```

The ID of the contact person concerned (`contact_id`) is always stored in the session, regardless of whether or not it is included in the list above.

2.7 Integrating the Rails Preview into the GUI

The instructions in this section apply to Infopark CMS Fiona 6.7.3. For other versions please refer to the corresponding manuals (PDF).

2.7.1 Prerequisites

To be able to preview pages generated by your Rails application in the editorial system, you first need to make sure that Infopark Rails Connector has been installed. Among other things, the connector provides a `PageController` responsible for generating the preview. Furthermore, a webserver able to work as a reverse proxy, is required, for example an Apache HTTP server.

The following steps illustrate by example how to configure the GUI, the Rails application, and Apache HTTP server in order to integrate the preview.

2.7.2 Configuring the GUI

For the GUI to generate preview URLs that match the route mapping of the Rails application's PageController, the property `railsMapping` of the bean `systemInfo` in the configuration file `WEB-INF/gui.xml` needs to be set as follows:

```
<property name="railsMapping" value="/:id/:name" />
```

After changing its configuration, the GUI it needs to be redeployed and the CMS must be restarted:

```
$ instancePath/bin/rc.npsd deploy GUI
$ instancePath/bin/rc.npsd restart
```

2.7.3 Configuring the Rails Application

Create a new environment, `preview`, by copying the file `config/environments/production.rb` to `config/environments/preview.rb`. Add the following line at the bottom of this file to make the Rails Connector display the draft versions of the CMS content:

```
RailsConnector::Configuration.mode = "editor"
```

Now create a new section, `preview`, in your database configuration and specify the database you intend to use for the preview environment.

Furthermore, it is essential that the web server passes the instance name to the Rails application server. If you use Apache webserver with Phusion Passenger, specify the instance name as the `RailsBaseURI` in the webserver configuration:

```
RailsBaseURI /instance-name
```

2.7.4 Configuring Apache Webserver

Communication between the browser and the GUI might look like cross-site scripting (XSS) to the browser. To avoid this, a webserver is placed as a proxy server between the client and the GUI/Rails. The server to which a request from the browser is directed, is determined by means of the URL prefix.

Please add the following entries to the Apache configuration file `httpd.conf`:

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so

# ..

ProxyRequests Off
ProxyVia On

# GUI requests
ProxyPass /instance-name/NPS http://localhost:8080/instance-name/NPS
ProxyPassReverse /instance-name/NPS http://localhost:8080/instance-name/NPS
```

Please make sure that the proxy modules are enabled as shown in the example above. Proxy definitions must be placed outside a container to take effect globally.

For the changes to take effect, the Apache server needs to be restarted. This will change the GUI URL to http://`gui-host`:`apache-port`/`instance-name`/NPS. In particular, the preview is no longer available on port 8080 (default setting).

2.8 Deploying a Rails Application

2.8.1 Prerequisites

To deploy your Rails application on a remote server you require the tool Capistrano. Therefore, please install the gem `capistrano` as follows:

```
# gem install capistrano --include-dependencies
```

The following instructions assume that you have installed Infopark Rails Connector for your Rails application.

2.8.2 Configuration

Configure Capistrano for your Rails application:

```
$ capify my-project
[add] writing `my-project/Capfile'
[add] writing `my-project/config/deploy.rb'
[done] capified!
```

Afterwards, please open the file *my-project*`/config/deploy.rb` and replace its content with the following lines:

```
set :application, "my-project"
set :repository, "subversion-URL"

set :deploy_via, :copy
set :copy_strategy, :export
set :user, "remote-login"
set :deploy_to, "target-path"
set :use_sudo, false

# Use the command line switch -S to preselect a stage.
# Example:
#   cap -S stage=production deploy
unless fetch(:stage, nil)
  set :stage do
    Capistrano::CLI.ui.choose do |menu|
      menu.header = "The following stages are available"
      menu.choice "preview"
      menu.choice "staging"
      menu.choice "production"
      menu.prompt = "Please choose: "
    end
  end
end

set :rails_env, stage

case rails_env
when "preview"
  role :web, "preview-web-server"
  role :app, "preview-app-server"
  role :db,  "preview-db-server", :primary => true
  set :prefix, "/cms-instance-name"
when "staging"
  role :web, "staging-web-server"
  role :app, "staging-app-server"
  role :db,  "staging-db-server", :primary => true
```

```
when "production"
  role :web, "production-web-server"
  role :app, "production-app-server"
  role :db,  "production-db-server", :primary => true
end

namespace :deploy do

  task :start do ; end
  task :stop do ; end
  desc 'Restart Rails App using Phusion Passenger'
  task :restart, :roles => :app do
    run "touch #{File.join(current_path,'tmp','restart.txt')}"
  end

  task :symlinks do
    %w(database.yml license.xml initializers/rails_connector.rb).each do |file|
      run "ln -nfs #{shared_path}/config/#{file} #{release_path}/config/#{file}"
    end
  end
end

before "deploy:setup", :db
after "deploy:update_code", "db:symlink"

set :keep_releases, 10
after "deploy:update", "deploy:cleanup"

require 'bundler/capistrano'
```

You can now adjust the configuration according to your needs:

- *my-project* is the folder name of your Rails application.
- *subversion-URL* is the subversion address of your Rails application.
- *remote-login* is the user name under which your Rails application will be deployed on the target host.
- *target-path* is the deploy directory on the target host.
- *CMS-instance-name* is the name of the CMS instance the preview server is designed for.
- *environment-role-server* is the target host for a specific environment (`preview`, `staging`, or `production`) and role (`:web`, `:app`, or `:db`).

Once a variable is set, it can be used as part of another variable. If you were to use the application name as part of the subversion address, you could write: `svn://svn/my-repository/#{application}`.

At the end of your adjustments, save the configuration file.

2.8.3 Initial Deployment

An environment needs to be set up before the Rails app can be deployed to it. This is achieved using the following command:

```
$ cap deploy:setup
1. preview
2. staging
3. production
Please choose: 2
  * executing `deploy:setup'
  * executing "umask 02 && mkdir -p /tmp/my-project /tmp/my-project/releases /tmp/my-
project/shared /tmp/my-project/shared/system /tmp/my-project/shared/log /tmp/my-project/
shared/pids"
```

```
    servers: ["ip20-127"]
    [ip20-127] executing command
    command finished
```

At the end of the configuration the software packages that need to be installed on the target host are listed. You can check if the required software is installed on the target host by executing the following command:

```
$ cd my-project
$ cap deploy:check
1. preview
2. staging
3. production
Please choose: 2
  * executing `deploy:check'
[...]
You appear to have all necessary dependencies installed
```

If all the requirements are met, you can deploy your Rails application:

```
$ cap deploy:cold
1. preview
2. staging
3. production
Please choose: 2
  * executing `deploy:cold'
[...]
    command finished
```

The command `deploy:cold` migrates the database before deploying.

2.8.4 Regular Deployment

Once your Rails app has been initially deployed, you can continue developing and deploy to the target host using:

```
$ cap deploy
1. preview
2. staging
3. production
Please choose: 2
  * executing `deploy'
[...]
    command finished
```

If your database schema changes in the course of an update, use `cap deploy:migrations` to bring the database up to date.

2.8.5 Commands for Other Administration Tasks

Capistrano also provides commands for further administration tasks along with the tasks described above. You can list all available commands using:

```
$ cap -T
1. preview
2. staging
3. production
```

```
Please choose: 1
cap deploy               # Deploys your project.
cap deploy:check         # Test deployment dependencies.
[...]
cap deploy:web:disable   # Present a maintenance page to visitors.
cap deploy:web:enable    # Makes the application web-accessible again.
cap invoke               # Invoke a single command on the remote servers.
cap shell                # Begin an interactive Capistrano session.
```

2.9 Cloning Database Contents (Using the Example of MySQL)

After having installed your development environment you need to fill the database with content for your Rails application to operate on. Please follow the instructions to import the live system's database content into a local MySQL database:

- Log on to the source server as a user who is permitted to access the database.
- Run the following command lines after having filled out the parameters (such as database username, password, target directory):

  ```
  $ mysqldump -h MySQL-host -u MySQL-login -p DB-name > dump.sql
  Enter password: MySQL-password
  ```

  In the preceding example the database dump is stored uncompressed as the file `dump.sql` in the current directory.
- Transfer the dump to the development host, for example by means of `scp` (secure copy).
- Import the dump into a new MySQL database. Here is an execution example:

  ```
  development-host:~ $ mysql -u root -p
  Enter password: MySQL-password
  Welcome to the MySQL monitor.  Commands end with ; or \g.
  Your MySQL connection id is 275
  Server version: 5.0.37 MySQL Community Server (GPL)

  Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

  mysql> create database railsdb;
  Query OK, 1 row affected (0.00 sec)

  mysql> grant all on railsdb.* to railsuser@localhost identified by 'railspassword';
  Query OK, 0 rows affected (0.00 sec)

  mysql> exit
  Bye


  development-host:~ $ mysql -u railsuser -prailspassword railsdb < dump.sql
  ```

2.9.1 Hints

The user and her password for database access should be the same as in the Rails database configuration file (`config/database.yml`).

2.10 Replicating the CMS Database

The following tables need to be included in the database replication process. The table prefix, such as `default`, equals the instance name.

From version 6.7.0

```
default_objs
default_blobs
default_permissions
default_news
```

From the `default_blobs` table only those entries are required in which `blob_name` equals `%.blobs` (`select * from default_blobs where blob_name like '%.blob'`).

Up to version 6.6.1

```
default_attributes
default_blobs
default_channels
default_contents
default_links
default_news
default_obj_class_attrs
default_obj_classes
default_objects
default_permissions
```

Steps required after replication

The replication of the tables `*_blobs`, `*_objs` (or `*_objects`, respectively), and `*_news` produces constraints that cannot be satisfied since they refer to tables that are not replicated. Therefore, it is required to remove these constraints immediately after each replication process:

```
ALTER TABLE instanceName_permissions DROP FOREIGN KEY internet_permfk_1;
ALTER TABLE instanceName_news DROP FOREIGN KEY internet_newsfk_1;
```

2.11 Delivering CMS Content Using ERb Templates

The CMS controller of Infopark Rails Connector uses views to deliver the contents of documents and folders. By means of URL routing, exactly one CMS file becomes the current context. In the view, this context can be accessed via the `@obj` instance variable of `Obj` class.

An ERb template is a view consisting of HTML code with embedded Ruby commands. The latter are used to generate content dynamically. To output, for example, the name of the current CMS file, you can use:

```
<%= @obj.name %>
```

The version fields of a file can be accessed in the same way. There are two variants for writing out a field such as the title of the released version (if no released version exists both commands lead to an error):

```
<%= @obj.title %>
<%= @obj[:title] %>
```

Use `@obj.title` if you wish to generate an error if the referenced field is missing. If missing fields are to be ignored, use `@obj[:title]`.

We recommend using the `display_field` helper for having all values displayed correctly (up to version 6.6.1 the helper `display` is used instead of `display_field` and `display_value`). Use it as follows:

```
<%= display_field @obj, :title %>
```

This command automatically places an edit marker for the referenced field into the preview. Use one of the following notations to disable this feature (up to version 6.6.1 the helper `display` is used instead of `display_field` and `display_value`):

```
<%= display_value @obj.title %>
<%= display_field @obj, :title, :marker => false %>
```

More complex functions such as creating a navigation are normally not implemented in the views directly. For clarity, better maintenance, and reusability, so-called helpers are used for this.

For HTML fragments to be reused, the Rails framework provides the concept of partials – small templates that can be stored separately.

2.12 Rendering CMS Content Using Liquid Templates

2.12.1 What is Liquid?

Liquid is a templating language written in Ruby with a focus on security and robustness. Developing templates with Liquid is simpler than with ERb, leading often to a quicker implementation with less danger of errors occurring. In contrast to ERb, Liquid silently handles errors caused by missing custom fields or nil values, thus ensuring a graceful degradation.

The following sites cover the basics for writing templates in Liquid:

- http://wiki.github.com/tobi/liquid/liquid-for-designers
- http://wiki.shopify.com/UsingLiquid

Infopark Rails Connector expands upon Liquid's basic syntax in order to handle the displaying of CMS content.

2.12.2 Displaying Content and Using Sub-Templates

The following Liquid code renders the title and main content for the current CMS object (represented by `obj`) and then invokes a sub-template:

```
<h1>{{ obj.title }}</h1>
{{ obj.body }}

{% template 'toclist' %}
```

`obj` is a Drop, which encapsulates an instance of the CMS object and controls access to the instance's methods and fields. The Drop automatically converts text to its HTML representation, resolves links, and inserts edit markers where applicable.

2.12.3 Context Lists, Link Lists, Links, and Images

The following template renders a list of the subfolders and subdocuments (`toclist`) of a CMS object, if it is a folder. Each list element is displayed with its thumbnail image and its title as a link:

```
<ul>
{% for object in obj.toclist %}
  <li>{{ object.thumbnail | image_tag }}
  {{ object.title | link_to: object }}</li>
{% endfor %}
</ul>
```

In this example, thumbnail images are referenced via a one-element link list named `thumbnail` and then displayed by the filter `image_tag`. The filter `link_to` is used to render an HTML link that points to the target CMS object.

Iterating over link lists can be done as follows:

```
<ul>
{% for link in obj.related_links %}
  <li>{{ link | link_to }}</li>
{% endfor %}
</ul>
```

If a link title has been specified, the `link_to` filter uses it as the text to be linked. If the link has no title, the filter uses the title of the target object if the link is internal. For external links, the URL is used.

Single links from a link list can also be accessed by their index:

```
{{ "A custom link text" | link_to: obj.related_links[2] }}
```

In the example above, the default link text is replaced with a custom link text.

2.12.4 Rendering Edit Markers Automatically

In general, editing elements are only displayed if the Rails application was configured for running in the `editor` mode, which should be the case if it is used as the CMS preview application. When using Liquid templates, the edit markers can be set either manually or automatically.

The standard practice is to enable edit markers by default. When the following code example is rendered in the preview, an edit marker appears automatically:

```
{{ obj.title }}
```

To provide for cases where for specific objects an edit marker is not desired, the filter `editmarker` can be used as follows:

```
{{ obj.title | editmarker: false }}
```

2.12.5 Rendering Edit Markers Manually

To deactivate edit markers globally for a Rails application, write the following in the application's configuration:

```
RailsConnector::Configuration.auto_liquid_editmarkers = false
```

This results in the following code no longer displaying an edit marker:

```
{{ obj.title }}
```

Yet again, the default behavior can be bypassed manually by using the `editmarker` filter as follows:

```
{{ obj.title | editmarker }}
```

2.12.6 Generating URLs for Content

The `url` filter can be used to retrieve the URL for a given CMS object.

```
<object data="external:{{ object | url }}" />
```

This is useful if a CMS object needs to be embedded into a page as Flash content or as a Java Applet.

2.12.7 Determining whether an Object Field has a Value

```
{% if obj.my_field != blank %}
  My field has the value {{ obj.my_field }}.
{% else %}
  My field has no value.
{% endif %}
```

In the above code, the `else` segment is evaluated if:

- `obj.my_field` contains an empty string
- `obj.my_field == nil`
- `obj` doesn't have a field named `my_field`

For this reason, `obj.my_field != blank` is the best way to protect against empty field values.

A similar syntax, `empty`, can be used to explicitly check for empty strings and nil values:

```
{% if obj.my_field == empty %}
  My field contains an empty string.
{% endif %}

{% if !obj.mein_feld %}
  My field doesn't exist (== nil).
{% endif %}
```

2.12.8 Rendering Formatted Dates and Times

Date and time values can be formatted using the `date` filter (see http://wiki.shopify.com/FilterReference#date.28input.2C_format.29):

```
{{ obj.ip_eventStart | date: "%d.%m.%Y" }}
```

2.12.9 Accessing Named Objects (Named Links)

The following Liquid statement retrieves the object referenced by the named link `my_name` and returns its title:

```
{{ named_object["my_name"].title }}
```

In the CMS, named links can be stored in the `relatedLinks` field of a CMS file to which the `NamedLink` file format was assigned. With Rails Connector, the target object of a link in this link list can be referenced by specifying the title of the link.)

2.12.10 Providing Action Markers

Action markers can be used to offer access to menu commands and wizards in the inline preview. Using Liquid, an action marker that opens an image editing wizard, for example, could be provided as follows:

```
{% actionmarker obj.some_image editImageWizard %}
  [edit image]
{% endactionmarker %}
```

2.13 How to Define Custom Liquid Filters

2.13.1 Liquid Filters

In Liquid, filters are used to output formatted text in templates. Liquid provides a couple of standard filters, such as `truncate` which limits the length of the output to the number of characters given:

```
The first 50 characters of the main content are:
{{ page.body | truncate: 50 }}
```

In this example, `page.body` is the input to the `truncate` filter to which a number is passed as an argument.

2.13.2 Creating Your Own Filters

As a developer of a Rails Connector application, you can easily create your own filters you can then use in Liquid templates. Basically, a filter is just a method that takes at least one parameter and returns a processed string.

Here is how you would implement a filter that wraps a text in `h1` tags:

```
module MyFilters

  def headline(input)
    "<h1>#{input}</h1>"
  end

end
```

In order for the Rails application to find your custom filters, you need to store them in the directory `/app/filters`. For the example above, the path would be `/app/filters/my_filters.rb`.

To use the above example in a template you would type the following:

```
{{ "Some Interesting Headline" | headline }}
```

This would produce:

```
<h1>Some Interesting Headline</h1>
```

2.13.3 Further Information

A good way to find inspiration in terms of how to implement custom filters is to read the source code of Liquid's standard filters.

2.14 Extending Views

The following instructions are addressed to developers who wish to add a view to an existing Rails application which already displays CMS content by means of Infopark Rails Connector. To do this, the following questions need to be answered.

- Where and how is the view determined that is to be used?
- Where and how can a new view be created?
- How can a view be filled with content?

Where and how is the view determined that is to be used?

Usually, it is not necessary to explicitly specify in a Rails application the view to be used for displaying content. You normally follow the convention by naming the view like the corresponding action of

the controller. In accordance with this convention, Rails would expect for the `CmsController#index` action that a view exists under `app/views/cms/index.html.erb` and use it.

You can follow this convention as long as all objects are to be displayed with an identical layout. However, typically, objects with different formats (`objClass`), field contents or paths exist. How these differences function in your web application and whether they affect, for example, how an object is displayed, depends on the individual case.

In cases where the format of an object serves as a criterion for layout selection, you can make use of dedicated controllers for specific CMS file formats provided by the Rails Connector.

Where and how can a new view be created?

If dedicated controllers for specific CMS file formats are used, views must be placed in a directory named like the controller and located below `app/views/`.

How can a view be filled with content?

In applications using the Rails Connector, the content of views is not conceptually different from the content in usual Rails applications. However, in every view an `@obj` instance is available that represents the object currently loaded. Depending on the format (`objClass`) assigned to the object, a particular set of fields can be accessed. The fields available include internal CMS fields such as `title`, `name`, `path`, `valid_from`, and `valid_until` as well as custom fields such as `abstract` or `showintoc`.

For displaying field values, helpers are available. For details, please refer to the API documentation of the Rails Connector.

2.15 Customize Rails Connector Views

The following components of Infopark Rails Connector are shipped with customizable views (basic knowledge of Ruby on Rails required):

- CMS Controller
- Search
- Time Machine
- Comments
- Ratings
- RSS feed
- SEO sitemap
- OMC integration

The view files are not located in your application directory, but are provided directly by the gem. The easiest way to change the output and appearance of a controller action is to create a view with a corresponding path and name in your application and thereby replace its previous definition. For example, for the action `SearchController#search`, you would need to create the file `app/views/search/search.html.erb`.

Predefined partials can be replaced, too. As a first step, it is recommended to get an overview of which partials exist in the Rails Connector gem and how they are used by other views or partials. To find out where the Rails Connector gem is installed on your machine, please run the following command:

```
$ gem which infopark_rails_connector
```

The views and partials for each controller are located in the subdirectory `app/views/controller-name`, for example `app/views/search` for the search views. Create a file in the corresponding subdirectory of the application to replace the packaged file of the same name.

2.16 Adapting Error Pages

Infopark Rails Connector offers the possibility to adapt the look of the error pages displayed for the status codes 403 (access denied) and 410 (page not found). Furthermore, the behavior of the Rails application in case one of these errors occurs can be changed.

To modify the displayed error page you can create an individual template and place it in the `app/views/errors/` directory.

To have your individual code executed as one of these errors occurs, you can extend the corresponding controller (for 403 errors) or the `CmsController` (for 410 errors) according to your needs. For details, please refer to the RDoc documentation on the `CmsAccessible` module part of Infopark Rails Connector.

2.17 Enable Permalinks

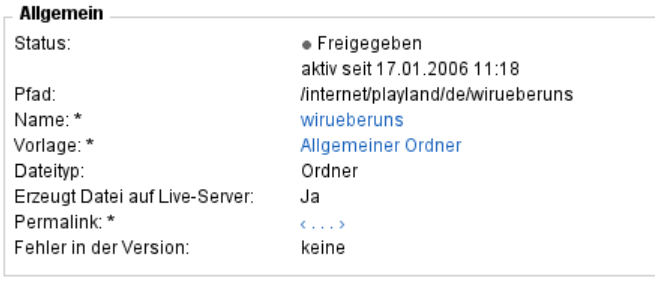
To enable the definition of permalinks in the editorial system of the CMS, please edit the `gui.xml` configuration file located in the `webapps/GUI/WEB-INF` directory of the CMS instance:

```
<bean id="inspectorRegistry" ...>
    ...
    <property name="enablePermalinks" value="true" />
</bean>
```

Afterwards, the GUI needs to be deployed:

```
./bin/rc.npsd deploy GUI
```

Permalinks now show up in the details view:



Permalinks need to be unique strings composed of characters permitted in URLs. Such a string becomes the path part of the URL of the CMS file concerned. The live server application has access to permalinks too, and is able to deliver the file using this path, in addition to its regular path.

2.18 Generating Forms for OMC Activities

If the Rails Connector has been connected to the Online Marketing Cockpit, i.e. if the OMC Connector has been installed and activated, you can have your Rails application generate forms based on OMC activities. When the website visitor submits such a form, the Rails Connector creates an activity from it and transmits this activity to the OMC. Thus, forms for customer requests, for event registration, and other tasks can be automatically created and are maintenance-free. The corresponding activities created in the OMC can be edited conveniently, or used for other purposes, e.g. for sending e-mails.

To create pages with such generated forms in the editorial system, you only require a file format named `OmcForm` and a CMS file based on this format. Since a view is provided for this format, your Rails application will then generate a form for the `contact form` activity type when the CMS file is delivered.

Technical Details and Refinement Options

The Rails application uses the supplied `OmcFormController` controller to deliver the CMS file based on the `OmcForm` format. In the view that belongs to this controller, the `OmcFormHelper` is used for generating the form fields.

The `OmcFormController` inherits from `DefaultOmcFormController`. The latter has methods that can be overridden to change the default settings of the form to be created – e.g. the activity kind to be used (`activity_kind`).

The activity kind – the default is `contact form` – determines the form fields the `OmcFormHelper` generates, as well as the type of activity it creates in the OMC when the form is submitted. The helper generates form fields for all custom fields of the activity kind and, optionally, a form field for the activity title. By means of a callback (`allow_custom_attribute?`), the creation of a form field for a custom field can be suppressed.

To create form pages with a different layout in the CMS, you can create and use individual formats as alternatives or complementary to `OmcForm`. If, for example, you would like to deliver an event registration form using you individual `OmcEventForm` format, you only need to define the `OmcEventFormController` class as a subclass of `RailsConnector::DefaultOmcFormController` in your Rails application. To change the behavior of your `OmcEventFormController`, override its methods. For details on creating and extending new controllers, please refer to sections Dedicated Controllers for Specific CMS File Formats and Customizing the Rails Connector.

You can now design the page by editing the view associated with the new controller subclass. In this view, the form fields can be generated using the `OmcFormHelper`.

For further information about form generation, please refer to the supplied RDoc documentation for the `DefaultOmcFormController`.

2.19 Activating Website Functions

2.19.1 Comments on Web Pages

The Rails Connector supports commenting on web pages. For this, it includes a database model as well as a controller and a partial.

To be able to use the commenting functionality, it needs to be activated first. For this, please edit the `config/initializers/rails_connector.rb` file and remove the hash character from the line `#:comments,`:

```
RailsConnector::Configuration.enable(
…
:comments,
…
)
```

Then, use the following command to have the required database migration steps generated:

```
rails generate rails_connector:comments
```

Run the migration script to create the database tables required for the comments:

```
rake db:migrate
```

To have the comments and the form for submitting a new comment displayed on a web page, the supplied `comments` partial is available. Use it as follows:

```
<%= render :partial => 'cms/comments' %>
```

If you wish to extend the way in which comments are processed, you can do so by subclassing the supplied `DefaultCommentsController`:

```
class CommentsController < RailsConnector::DefaultCommentsController
# Your code
end
```

2.19.2 Providing RSS Feeds

The Rails Connector includes a controller for delivering RSS feeds. Prior to using it, the RSS feature needs to be activated. For doing this, edit the `config/initializers/rails_connector.rb` file and remove the comment character from the line `# :rss,`:

```
RailsConnector::Configuration.enable(
…
:rss,
…
)
```

As a default, the RSS feed is generated from the CMS files located in a specific folder. This folder object must be loaded via the `config/initializers/rails_connector.rb` file, e.g. by means of a NamedLink:

```
# RSS-Feed:
#
# Specify which Object should be used for the RSS feed's parent folder
RailsConnector::Configuration::Rss.root = NamedLink.get_object('news')
```

The RSS feed is generated via the `DefaultRssController`, using the `app/views/rss/_item.rss.builder` template. For generating the feed differently, you can modify this template, or write your own controller that uses a different template:

```
class RssController < RailsConnector::DefaultRssController
  def podcast_feed
    @obj = NamedLink.get_object('podcast_de')
    headers['Content-Type'] = 'application/rss+xml'
    render :layout => false
  end
end
```

2.20 Integrating the Time Machine

Run the `rails_connector_addons` generator, supplied with the Infopark Rails Connector Addon Gem, in order to create all files necessary for using the "Time Machine":

```
$ script/generate infopark_rails_connector_addons
```

You should now have an `infopark_rails_connector_addons.rb` file in your application's `config/initializers` directory, in which the Time Machine is turned on by default. To override this, remove the `:time_machine` entry in the `RailsConnector::Configuration.enable()` statement.

Use the `TimeMachineHelper` to create a link that opens the Time Machine window:

```
<%= time_machine_link 'Time Machine' %>
```

In case the `MenuHelper` is being used on the same page, it is necessary to pass it the parameter `current_time` so that documents linked in the generated navigation receive the same time configuration. Consult the helper's documentation for further information on that. It can be found in the `doc` directory of the Infopark Rails Connector Addon gem.

2.21 Enabling Searches Using Infopark Search Server

The Rails Connector includes a default search page that can be accessed using the URL `/search` in your Rails application.

Infopark Search Server (SES) can be used to search for CMS content. After having installed the Rails Connector into your Rails application, you should have a `rails_connector.rb` file in your application's `config/initializers` directory in which the search feature is turned on by default. To switch the search off, remove the `:search` entry from the addons list in this file.

A SearchController, SearchHelper, and several views, including a search form and a results list, are provided by the addons gem and work out of the box.

Customization

You can choose to adjust the views according to your needs.

The Search Controller can be configured using the `rails_connector.rb` config file. You can adjust the host and the port of the Search Server as well as the collection to be searched:

```
RailsConnector::Configuration.search_options = {
  :host => 'custom_host',
  :port => 3011,
  :collection => 'cm-contents-en'
}
```

In order to further customize the search controller, you need to create your own search controller in your application:

```
class SearchController < RailsConnector::DefaultSearchController
  # Your custom code
end
```

The search controller uses instances of the `SearchRequest` class to form valid search queries from user input. Creating your own search controller lets you control all aspects of search processing, such as which documents are shown in search results, or how user input is sanitized:

```
class SearchRequest < RailsConnector::DefaultSearchRequest

  def self.sanitize(text)
    # Your implementation
  end

  def base_query
    # Define how `base_query_conditions`
    # are combined to form the VQL base query
  end

  def base_query_conditions
    super.merge(
      :custom_condition => 'VQL code' #, ...
    )
  end
end
```

Overwriting the search helpers works by a similar mechanism. To do so, create a file `app/helpers/search_helper.rb` in your application with the following content:

```
module SearchHelper
  include RailsConnector::DefaultSearchHelper

  # Overwrite or define helpers
end
```

2.22 Enabling the PDF Generator

Please proceed as follows to enable the PDF Generator component of Infopark Rails Connector.

2.22.1 Installing Apache FOP

The PDF generator makes use of Apache FOP (Formatting Objects Processor), to create PDF files.

1. Change to the directory where you wish to install Apache FOP, for example to `/opt/local`:

   ```
   $ cd /opt/local
   ```

2. Now, download Apache FOP, version 0.94, from the official website:

```
$ wget http://archive.apache.org/dist/xmlgraphics/fop/binaries/fop-0.94-bin-jdk1.4.zip
```

   The PDF generator is not compatible with newer versions of Apache FOP.

3. Unpack the downloaded archive:

```
$ unzip fop-0.94-bin-jdk1.4.zip
$ rm fop-0.94-bin-jdk1.4.zip
```

4. Change to the installation directory of the FOP and make the start script executable:

```
$ cd fop-0.94
$ chmod 755 fop
```

5. With large PDF files, Apache FOP requires more memory than it assigns to itself initially. To prevent a lack of memory, we recommend to increase the amount of RAM available to the FOP. To do this, open the file `fop-0.94/fop` and change the line

```
fop_exec_command="exec \"$JAVACMD\" $LOGCHOICE $LOGLEVEL -classpath (...)
```

   to the following:

```
fop_exec_command="exec \"$JAVACMD\" -Xmx500M $LOGCHOICE $LOGLEVEL -classpath (...)
```

6. Add the path of the FOP's installation directory to your search paths in the `PATH` environment variable.

2.22.2 Installing Tidy

The PDF Generator uses Tidy to clean up invalid XML. Therefore, please install Tidy first. The documentation for Tidy can be found on the official website, http://tidy.sourceforge.net.

The Tidy gem is also required. However, this gem has already been installed since the Rails Connector depends on it.

2.22.3 Activating the PDF Generator

To activate the PDF generator, open the file `RAILS_ROOT/config/initializers/rails_connector.rb` and remove the comment character from the line indicated below:

```
  [...]
  :time_machine,
# :pdf_generator
)
```

2.23 Maintenance Tasks

2.23.1 Removing Sessions from the Database

If you configured Rails to store browser sessions in the database, it is recommended to remove them regularly in order to prevent the table concerned from becoming too large and thus impairing overall performance.

For this, you can use the following Ruby code as a rake task in your project. This task can and should be run regularly as a cron job.

```
namespace :db do
  namespace :sessions do
    desc "Prune database-stored sessions older than one week"
    task :prune => :environment do
      CGI::Session::ActiveRecordStore::Session.delete_all ["updated_at < ?", 1.week.ago ]
    end

    desc "Count database sessions"
    task :count => :environment do
      puts "Currently storing #{CGI::Session::ActiveRecordStore::Session.count} sessions"
    end
  end
end
```

2.24 Customizing the Rails Connector

All components of Infopark Rails Connector are provided ready-to-run. In addition, various interfaces are provided to enable you to adjust the software according to your requirements.

There are interfaces to extend and replace parts of the following components:

- Controllers (e.g. `CmsController`, `SearchController`)
- Helpers (e.g. `CmsHelper`, `SearchHelper`)
- Views (e.g. for search results)
- The `Obj` model

2.24.1 Controllers

The controller classes of Infopark Rails Connector are extendable by means of inheritance. The `CmsController` class, for example, is a sub-class of `RailsConnector::DefaultCmsController` and thus inherits all of its behavior. The `CmsController` class implements no additional functionality but merely acts as a placeholder for your custom code.

In order to adjust the behavior of the CMS controller, create the file `app/controllers/cms_controller.rb` in your application. Then copy the following code and paste it into the file:

```
class CmsController < RailsConnector::DefaultCmsController
  # Your custom code
end
```

2.24.2 Helpers

The hierarchy of the Rails Connector's helper modules is similar to that of the controller classes. There is, for example, a `CmsHelper` module that uses the functionality of the `DefaultCmsHelper` module but does not add any functionality to it. In order to add or replace CMS helper methods, create the file `app/helpers/cms_helper.rb` in your application with the following content:

```
module CmsHelper
  include RailsConnector::DefaultCmsHelper

  # Your custom code
end
```

2.24.3 Views

You can replace the Rails Connector's default views. If you wish to do so, please refer to the corresponding section, Customize Rails Connector Views.

2.24.4 The `Obj` Model

On initialization, the Rails Connector looks for a module named `ObjExtensions` in your application code. If it exists, its method `enable` is called. You can use this mechanism to add custom features to the `Obj` model. To do so, create a file named `obj_extensions.rb` in your application and start off with the following code:

```
module ObjExtensions
  def self.enable
    Obj.class_eval do
      # Your own method definitions
    end
  end
end
```

2.25 Best Programming Practices

2.26 Making Use of Link Management in Projects Based on Rails Connector

Infopark CMS Fiona features internal link management that ensures, for example, that CMS file references remain intact when files are renamed or moved. The Rails Connector, however, is not supplied with information about changed file paths. For this reason, you should avoid using calls such as `Obj.find(1234)` or `Obj.find_by_path("/company/news/")` in your Rails application.

To be, nevertheless, able to access specific CMS files directly, the Rails Connector offers so-called named links. In the CMS, create a file named `NamedLink`, based on the `NamedLink` file format, and give it a `related_links` link list field. You can now maintain your file references using this link list (the links contained in link lists are covered by link management). In the Rails application, you can then access the linked files by means of the titles that have been given to the links (e.g.

`NamedLink.get_object("news")`). Furthermore, the named link are held in the cache, which saves resources when they are accessed.

Typical use cases for named links are, for example:

- Display all the CMS files contained in the news folder on your home page:
  `NamedLink.get_object("news").each do ...`
- Perform specific actions for a specific page:
  `if @obj == NamedLink.get_object("homepage") ...`
- Fetch a field value from a specific file:
  `NamedLink.get_object("important_notes").body`

Further information about this can be found in the API documentation on the `NamedLink` class.

Please take account of the following when using named links:

Only one file with the `NamedLink` format may exist in the CMS. Otherwise, the Rails application will produce errors. To prevent editors from inadvertently creating further files based on the `NamedLink` format, the format should be deactivated after the file has been created.

Furthermore, the CMS files to which the named links point must be neither deactivated nor unreleased. If the Rails application tries to access a deactivated or unreleased CMS file via a named link, an exception is thrown. If this exception is not explicitly caught it causes all the pages containing this named link to be broken. Depending on where named links pointing to nonexistent targets are used, the complete website might fail.

2.27 Adapting a Rails Project to the Local Development Environment

A Rails project can be configured by means of the files `config/environment.rb`, `config/environments/*.rb`, and `config/initializers/rails_connector.rb`.

From version 6.7.3 these configuration files are overridden by Ruby files located in the `config/local` directory. This makes it possible to adapt the Rails Connector to the local development environment without having to change the project configuration.

By means of these files the Rails configuration as well as the Rails Connector configuration can be adapted locally. An example:

```
config.action_mailer.delivery_method = :my_local_delivery_method

RailsConnector::Configuration.instance_name = "my_local_instance_name"
```

Immediately after the Rails Connector has been installed, `config/local/configuration.rb` is empty, meaning that no changes to the project configuration are made.

If the development of your Rails application is supported by a version control system, it should be configured so as to ignore the files located in the `config/local` directory.

2.28 Tips and Tricks

2.29 Date, Time, and Time Zones

Infopark Rails Connector automatically converts date values originating from the CMS to the time zone currently active in the Rails application. To do this, it uses the `Time` Ruby class. The timezone active in the Rails application can be determined as follows:

```
$ rails console
Loading development environment (Rails 3.0.5)
>> Time.now.zone
=> "CET"
```

See also the Rails documentation on time zones at http://apidock.com/rails/ActiveSupport/TimeWithZone.

2.30 Accessing the API Documentation

The API documentation of the Rails Connector is generated by means of RDoc during the installation of the Rails Connector. To access the documentation the RDoc server needs to be started first:

```
gem server
```

Afterwards, you can read the documentation for all the installed gems by pointing a web browser to `http://0.0.0.0:8808/`.