Grails Jasper Plugin - Reference Documentation

Introduction

JasperReports is an open source Java reporting tool for producing documents that can be viewed, printed or exported in different formats such as PDF, HTML, Microsoft Excel, RTF, OpenOffice, XML, comma-separated values (CSV) or Word. It can be used in Java-enabled applications, including Java EE or web applications, to generate dynamic content.

This plugin is a port of the original JasperReports plugin later updated for **Grails 3** to the newest JasperReports API and based of **Grails 5**, that adds easy support for launching jasper reports from GSP pages.

The plugin package names have been kept the same and the methods and taglib are compatible with previous versions.

In addition to this document, you may want to read official Jasper Reports Library Documentation here.

Contributors are welcome.

Features

- The jars for executing `.jasper` reports (already compiled) and/or compiling them from `.jrxml` files, on the fly, first.
- A GSP taglib for launching reports.
- Corresponding controller and service logic (that can be invoked directly)
- `35x47` icons (`src/main/public/images/icons/*.png`) for every supported file format .

Issues

All issues must be reported to GitHub Issues.

Release Notes

- Updated to JasperReports 6.20.0.
- Upgraded to Grails 5.1.9.
- Updated file type icons.
- Removed demo pages from release plugin JAR.

- Update to jasperreports 5.5.0
- Cleanup (Thanks to Burt Beckwith)
- various bugfixes

- Update to jasperreports 4.5.0 and POI 3.7
- Support for Images in HTML Reports (thanks to Rafael Gutierrez for the patch)
- various bugfixes

- Update to jasperreports 4.0.0 (sounds like a new major release of the lib, but it isn't)

New features:

- Update to jasperreports 3.7.4
- generate reports with service methods (see the included demo.gsp for documentation)

New features:

- Support for OpenDocument and OOXML export (ODS, ODT, DOCX, XLSX, PPTX)
- Option to disable default parameters
- Parameter pass-through
- Japser libraries updated to the newest version (3.7.3)
- Refactored service with the ability to create a single document form more than one jrxml/jasper file.
- finally uses packages

Bug fixes: Changelog

New features:

- New tags which allow for greater control of JasperForms layout and button placement, and unobtrusive JS and CSS. see demo.gsp for documentation
- Icons rendered via CSS and new small PDF icon included
- Can now use latest iReport to edit reports thanks to updated Jasper libraries

New features:

- Added many new attributes to `<g:jasperReport>` that control the rendering (see demo.gsp)
- Now, if the g:jasperReport tag does not have a body, then the rendered HTML will be just a series of one or more `<A>` tags (no form and no javascript for submitting said form).
- Combined the redundant admin.gsp and howto.gsp into a single demo.gsp.
- Added lots more documentation and examples to demo.gsp.

Bug fixes:

- The Format attribute is now tolerant of lower case format values (pdf vs. PDF) and spaces between values.
- The tag validation code now checks for the two required attributes: jasper and format
- Rendering now uses ` ` and `<strong>` (rather than nothing and `<b>`) in places
- No longer renders the word "null" when the name attribute is left blank

Infrastructure changes:

- Numerous. (See subversion log.)

New features:

- Attribute `from` was removed.
- Attributes `controller` and 'action' was included.
- The developer have to use the follow architecture if the developer don’t want to use SQL into reports:

New features:

- Attribute 'inline=<boolean>' was included in jasperReport tag. Now, pdf-files can be shown inline in the web-browser.

Bug fixed:

- Solved problem that the 'format'-attribute did not accept spaces.
- Solved problem that sometimes the from-parameter was not found.
- The default locale of the report is the request locale.
- The default subreport folder is the same of the report.

New features:

- The user can to retrieve data report from domain classes:
- Attribute 'from' was included in jasperReport tag.
- XLS parameters were improved: (Thanks to Sebastian Esch)

- One page per sheet;
- Auto detect cell type;
- Remove empty space between rows;
- White page background is disabled.

Bug fixed:

- Plugin didn't work on Linux (File separator was wrong). (Thanks to Sebastian Esch)

Bug fixed:

- Bug in JasperService.groovy that can cause connection leaks, connection is never closed. (Thanks to Pass F. B. Travis)

Acknowledgments

Many thanks to all the users who reported issues and sent pull requests.

Authors and Contributors

- Craig Andrews
- Burt Beckwith
- Puneet Behl
- Mansi Arora
- Manvendra Singh
- https://github.com/daraii

License

This plugin is released under the Apache License, Version 2.0

Usage

Installation

For Grails 5.x

Add the following dependency under `build.gradle`:

```
implementation "io.jellycat.plugins:jasper:2.2.0"
```

For Grails 3.x

Add the following dependency under `build.gradle`:

```
compile "org.grails.plugins:jasper:2.0.0.RC1"
```

For Grails 2.x

Add the following plugin under plugins in `BuildConfig.groovy`:

```
compile "org.grails.plugins:jasper:2.2.0"
```

What is installed?

- The jars for executing `.jasper` reports (already compiled) and/or compiling them from `.jrxml` files, on the fly, first.
- A GSP taglib for launching reports.
- Corresponding controller and service logic (that can be invoked directly)
- 35x47 icons (`src/main/public/images/icons/*.png`) for every supported file format.

Configuration

The default location for your report templates is `classpath:/public/reports` in your project directory. Here you can place your `*.jasper` or `*.jrxml` (`jrxml` files will be compiled automatically by the plugin).

> Work with `jrxml` files if you can! They can be compiled by the plugin if a newer JasperReports version is available. This way you don't need to manually recompile all your reports if you want to use the new version with braking changes.

You can set another report folder location with the jasper.dir.reports property in your `application.yml`.

```
jasper:
    dir:
        reports: 'classpath:/public/reports'
```

NOTE	Notice the directory locations and other paths are represented using the Spring Framework's Resource notation. Please check the documentation for how to represent your report location.

It’s possible to use different locations for different environments.

```
environments:
    development:
        jasper:
            dir:
                reports: 'classpath:/public/reports'
    production:
        jasper:
            dir:
                reports: 'classpath:/public/prod/reports'
```

Basic usage

Tags

The plugin provides a number of tags to help with the integration in your pages.

jasperReport

The `jasperReport` tag generates a button for every file specified file format. With a click on one of these icons you generate the report which is returned as the response.

NOTE	The `jasperReport` tag should not be nested with a form element, as it uses a form element in its implementation, and nesting of forms is not allowed.

```
<g:jasperReport
          jasper="sample-jasper-plugin"
          format="PDF,HTML,XML,CSV,XLS,RTF,TEXT,ODT,ODS,DOCX,XLSX,PPTX"
          name="Parameter Example">

    Your name: <input type="text" name="name"/>

</g:jasperReport>
```

The input of the text field will be passed to the report as a parameter.

- jasper - filepath, relative to your configured report folder, of the report, no file extension needed (Required)
- format - supply the file formats you want to use in a simple list (Required)
- name - name of the report
- delimiter - delimiter between the icons representing the file formats.

- delimiterBefore - delimiter in front of the icons
- delimiterAfter - delimiter at the end of the icons
- description - description of the report
- buttonPosition - position of the icons (top or below)

jasperForm

The `jasperForm` works the same way as the `jasperReport` tag, but gives you more control over how the form is rendered in HTML.

NOTE	The jasperForm tag should not be nested with a form element, as it uses a form element in its implementation, and nesting of forms is not allowed.

```
<g:jasperForm controller="jasper" action="exampleWithData" id="1498"
jasper="w_iReport" >

    ..form contents..

    <g:jasperButton format="pdf" jasper="jasper-test" text="PDF" />

    .. other html..

</g:jasperForm>
```

- jasper - filepath, relative to your configured report folder, of the report, no file extension needed. (Required)
- controller - The controller the form will submit to. (Required)
- action - The action the form will submit to. (Required)
- id - The id attribute for the form.
- class - Style class to use for the form. The default is "jasperReport".

jasperButton

Use the `jasperButton` inside a `jasperForm` to submit and generate the report.

```
 <g:jasperButton format="pdf" jasper="jasper-test" text="PDF" />
```

- format - The name of the supported output format. ex. 'pdf' or 'PDF'. (Required)
- class - Class of the element in addition to default.
- text - Text to be included next to button ex. 'print'.

Services

From version 1.1 upwards it's possible to generate your report, without the controller action from above, with simple service methods (so that you can generate your reports with a cron job in combination with the Quartz plugin).

The central element for this feature is a new wrapper class `JasperReportDef`. Instead of putting everything in the parameter map you create a simple object containing the relevant data.

```
def reportDef = JasperReportDef(name:'your_report.jasper',
fileFormat:JasperExportFormat.PDF_FORMAT)
```

As you can see there are only two required attributes. Of course, you need provide the name of your report and the target file format. All available file formats can be found in the `JasperExportFormat` enum. You just have to choose one.

- **name** - Name of the Report. (Required)
- **fileFormat** - File format of the Report. Please use the JasperExportFormat Enum. (Required)
- **folder** - The folder where you placed your reports. Defaults to `classpath:/public/reports` if unset and no global setting (jasper.report.dir in application.yml, application.groovy or application.properties) exists.
- **reportData** - Collection containing the data of your report (leave empty if you want to use a SQL query inside your report)
- **locale** - Locale to use in the report generation.
- **parameters** - All additional report and exporter parameters as a Map.

All you need to do now is to call one of the methods provided in `JasperService`:

- `generateReport(JasperReportDef reportDef)` - Generate a "normal" report.
- `generateReport(List<JasperReportDef> reports)` - Generate a single response containing multiple reports.

Both return a ByteArrayOutputStream with which you can do everything you want.

```
class YourClass {

    def jasperService

    public void yourMethod() {
        def reportDef = new JasperReportDef(name:'your_report.jasper',
fileFormat:JasperExportFormat.PDF_FORMAT)
        FileUtils.writeByteArrayToFile(new File("/your/target/path/test.pdf"),
jasperService.generateReport(reportDef).toByteArray())
    }
}
```

NOTE The example above uses the apache common-io `FileUtils` to store the response on the disc.