camptocamp

INNOVATIVE SOLUTIONS
BY OPEN SOURCE EXPERTS

MAPFISH PRINT 3

COURSE GUIDE

Workshop

TABLE OF CONTENTS

About Camptocamp

Company specialized in Open Source, pioneering in software development:

- Geographical Information System (**GIS**)
- Business Management (**ERP**)
- Server Infrastructure (**IT Automation and Orchestration**)

Presence in three countries :

- Switzerland (Camptocamp SA)
- France (Camptocamp France SAS)
- Germany (Linuxland GmbH)

Geospatial department:

- Contributor to famous projects as OpenLayers, GeoNetwork, MapServer, QGIS, etc.
- Editor of two main products geOrchestra and GeoMapfish
- Development of tailor-made solutions based on QGIS, GeoMapfish, geOrchestra, etc.

INTRODUCTION

Agenda of this chapter

Goal: Get an idea on what MapFish Print is and how a report is generated.

- What is MapFish Print?
- Architecture
- Print process
- Basic configuration
- Resources

What is MapFish Print?

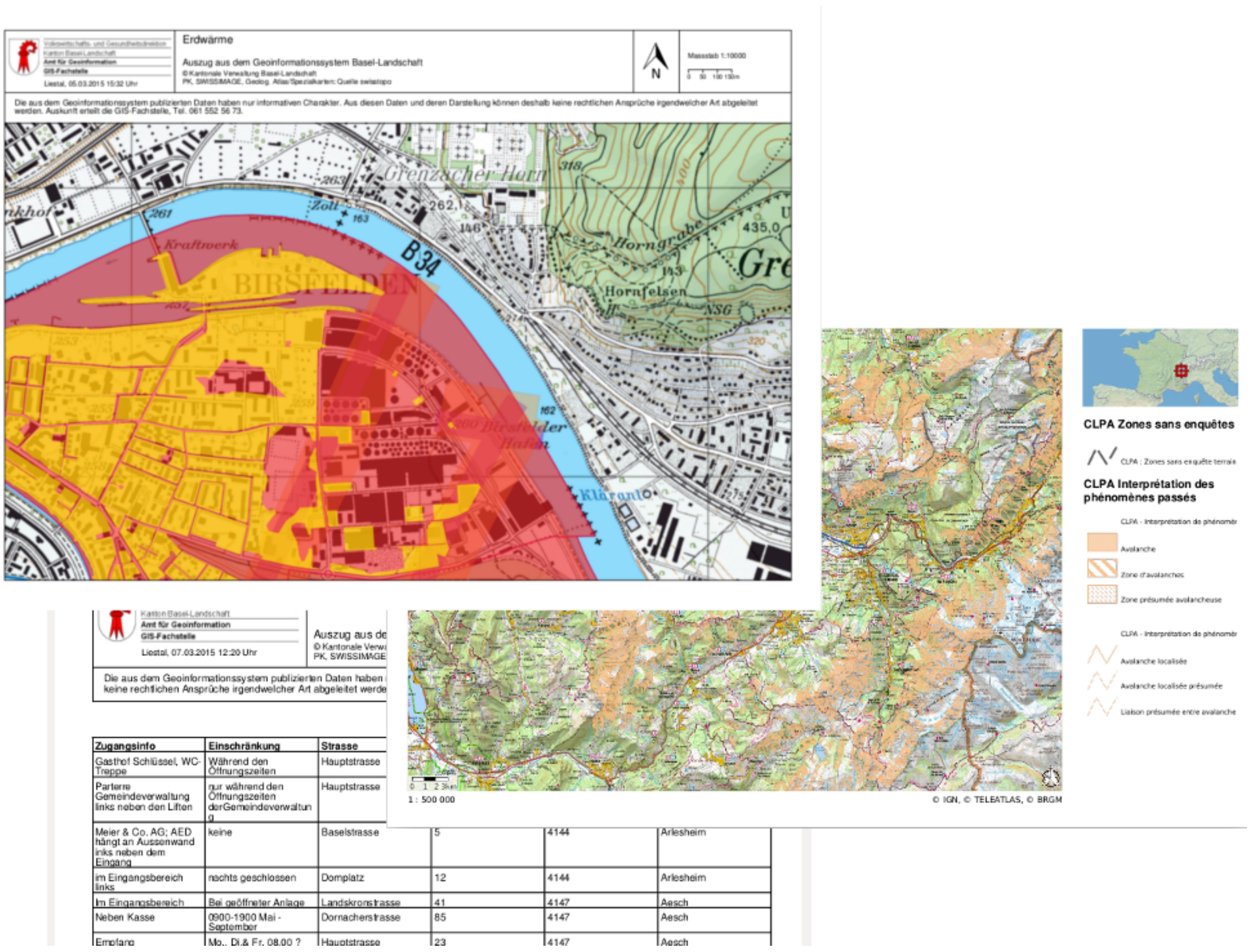
- Goal of the project: Create reports that contain maps and map related components
- A Java library and web-application

Notes:

The goal of MapFish Print is to create reports with maps and other map related components, like scalebars, north-arrows or legend.

The project consists of a Java library that can be used in own Java programs, and a web-application that is to be deployed to a Java application server.

Examples



Volkswirtschafts- und Gesundheitsdirektion
Kanton Basel-Landschaft
Amt für Geoinformation
GIS-Fachstelle
Liestal, 05.03.2015 15:32 Uhr

Erdwärme

Auszug aus dem Geoinformationssystem Basel-Landschaft
© Kantonale Verwaltung Basel-Landschaft
PK, SWISSIMAGE, Geolog. Atlas/Spezialkarten: Quelle swisstopo

N

Massstab 1:10000
0 50 100 150m

Die aus dem Geoinformationssystem publizierten Daten haben nur informativen Charakter. Aus diesen Daten und deren Darstellung können deshalb keine rechtlichen Ansprüche irgendwelcher Art abgeleitet werden. Auskunft erteilt die GIS-Fachstelle, Tel. 061 552 56 73.

Kanton Basel-Landschaft
Amt für Geoinformation
GIS-Fachstelle
Liestal, 07.03.2015 12:20 Uhr

Auszug aus de
© Kantonale Verwa
PK, SWISSIMAGE

Die aus dem Geoinformationssystem publizierten Daten haben
keine rechtlichen Ansprüche irgendwelcher Art abgeleitet werde

Zugangsinfo	Einschränkung	Strasse			
Gasthof Schlüssel, WC-Treppe	Während den Öffnungszeiten	Hauptstrasse			
Parterre Gemeindeverwaltung links neben den Liften	nur während den Öffnungszeiten derGemeindeverwaltung	Hauptstrasse			
Meier & Co. AG; AED hängt an Aussenwand inks neben dem Eingang	keine	Baselstrasse	5	4144	Arlesheim
im Eingangsbereich links	nachts geschlossen	Domplatz	12	4144	Arlesheim
Im Eingangsbereich	Bei geöffneter Anlage	Landskronstrasse	41	4147	Aesch
Neben Kasse	0900-1900 Mai - September	Dornacherstrasse	85	4147	Aesch
Empfang	Mo., Di.& Fr. 08.00 ?	Hauptstrasse	23	4147	Aesch

0 1 2 3km
1 : 500 000
© IGN, © TELEATLAS, © BRGM

CLPA Zones sans enquêtes

CLPA : Zones sans enquête terrain

CLPA Interprétation des phénomènes passés

CLPA - Interprétation de phénomèn

🟧	Avalanche
▨	Zone d'avalanches
▩	Zone présumée avalancheuse

CLPA - Interprétation de phénomèn

/\/	Avalanche localisée
/\/	Avalanche localisée présumée
/\/	Liaison présumée entre avalanche

Features

- Components: map, overview-map, scalebar, north-arrow, legend
- Other elements: tables, diagrams, graphics
- Supported geo-data: GeoJSON, GML, GeoTIFF, WMS, WMTS, XYZ/OSM
- Vector styling: SLD, JSON style
- Layouting via JasperReports
- Multi-page support (e.g. with multiple maps)
- Highly customizable
- Integrates with external datasources (e.g. databases)

Architecture

GeoTools

Mapping

JasperReports

Layout

spring

Plugin Framework

mapfish

MapFish Print

Web API / Security / Widgets

Notes:

MapFish Print 3 is built on mature open-source libraries namely GeoTools, JasperReports and the Spring framework.

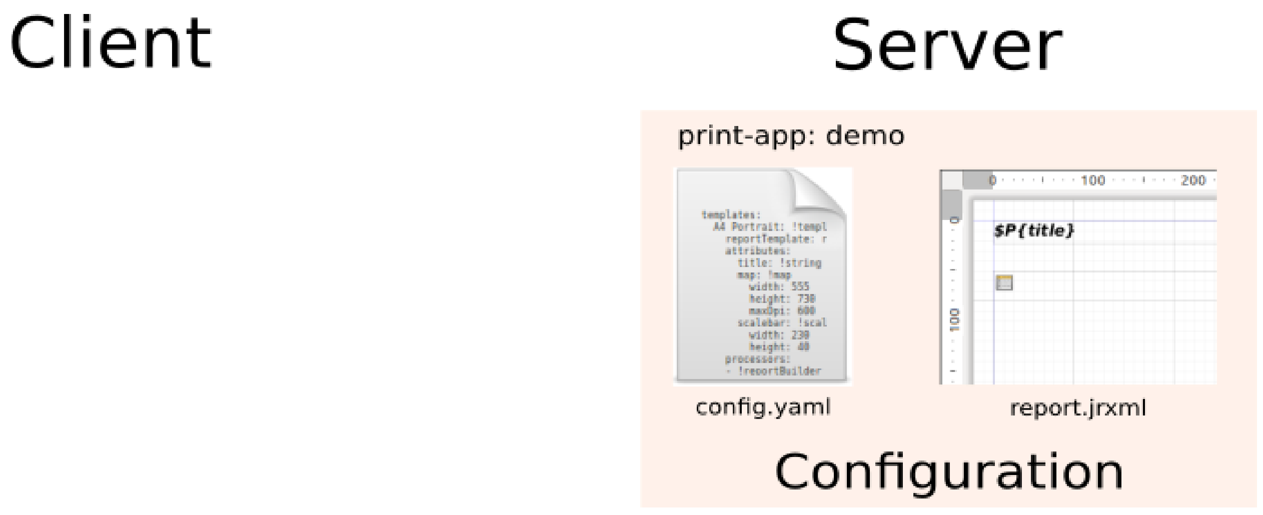
GeoTools, which for example is also powering GeoServer, is used for everything related to mapping. It is used to generate the map layer graphics, that are embedded into the report.

The JasperReports library is an open-source reporting engine that is able to read from many data sources and to generate a variety of output formats (e.g. HTML, PDF, LibreOffice formats, ...). MapFish Print is using the JasperReports library to layout the report. Templates for JasperReports can be edited with a visual editor.

The architecture of MapFish Print is a plugin architecture building on the Spring framework. This allows to extend the functionality with custom plugins. For example when a geo-data format is missing, a custom plugin can be written for that format and be registered with MapFish Print.

While GeoTools and JasperReports are doing the heavy lifting, MapFish Print itself provides an API via its web-application and offers reports widgets like scalebars, overview-maps or legends.

Print process: Configuration



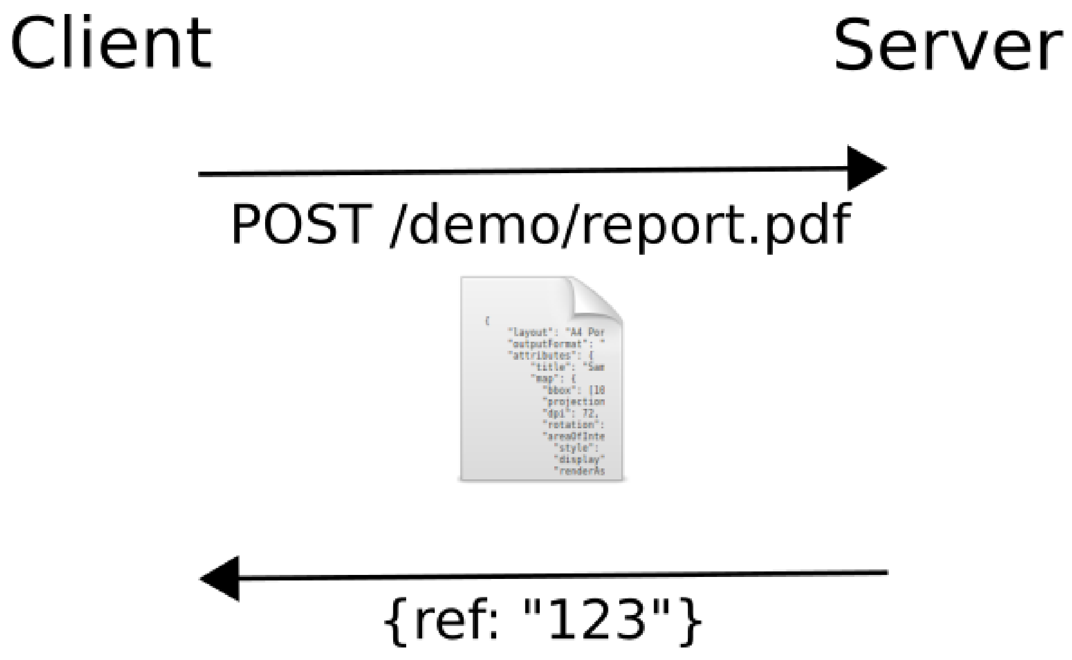
```
templates:
  A4 Portrait: !templ
    reportTemplate: r
    attributes:
      title: !string
      map: !map
        width: 555
        height: 730
        maxDpi: 600
      scalebar: !scal
        width: 230
        height: 40
    processors:
    - !reportBuilder
```

Notes:

The following slides will walk through the process of generating a report when using the MapFish Print web-application.

Each report must have a print configuration which is stored on the server. Each configuration (or 'print-app') must a least have a YAML configuration file (`config.yaml`) and a JasperReports template file (`report.jrxml`).

Print process: Creating a print job



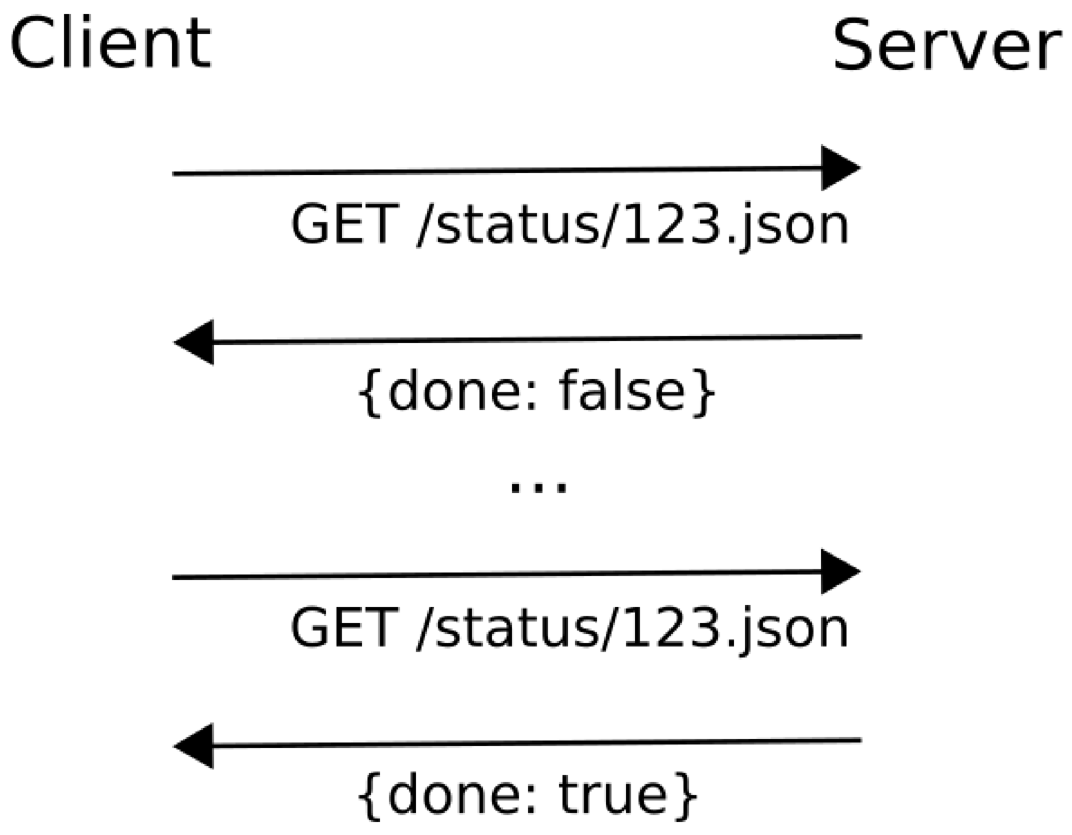
Notes:

To generate a report, a client sends a print request to the server. This request uses the JSON format and species for example the report title, the map extent that we want to print and the geo-data that should be shown.

The server registers the print job in a job queue and returns a reference id to the client.

The server processes the job queue and handles each print. It parses the given request, downloads the required data, creates the map image and generates the report file.

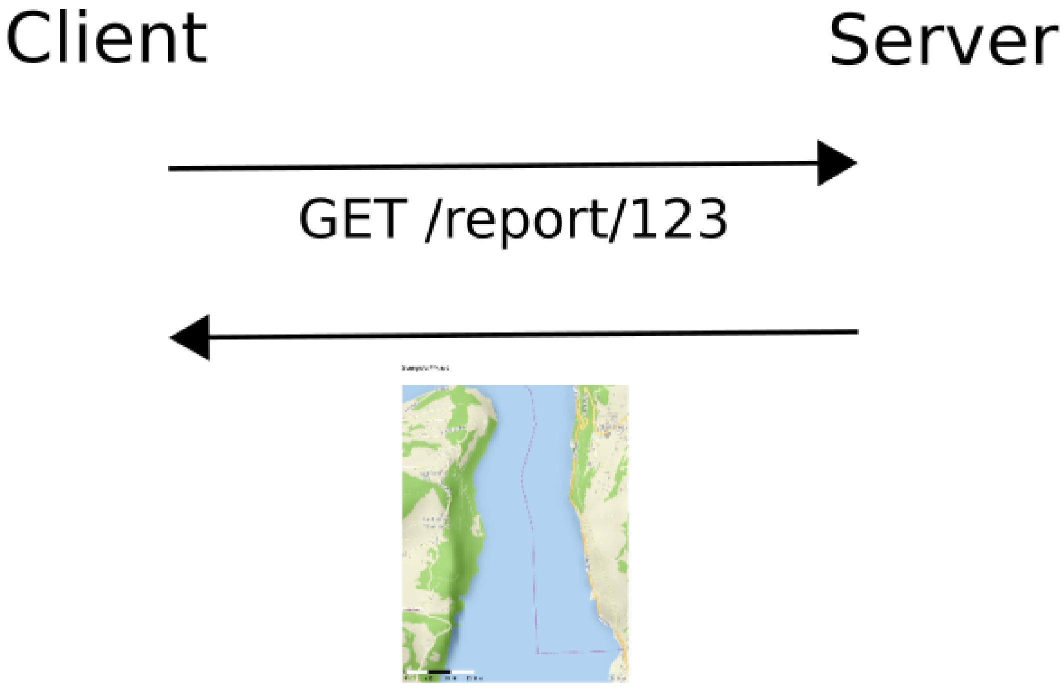
Print process: Polling the status of a print job



Notes:

The client can use the received job id to request the status of the print. As long as the print is still running, the server will return `done: false`. Once the report has been generated, the response will be `done: true`. Now, the report can be downloaded (see next slide).

Print process: Getting the created report



Basic configuration (config.yaml)

```yaml
templates:
  A4 Portrait: !template
    reportTemplate: report.jrxml

    attributes:
      title: !string {}
      map: !map
        width: 555
        height: 730
        maxDpi: 300
      scalebar: !scalebar
        width: 230
        height: 40

    processors:
    - !reportBuilder
      directory: '.'
    - !createMap
      inputMapper: {map: map}
      outputMapper: {mapSubReport: mapSubReport}
    - !createScalebar {}
```

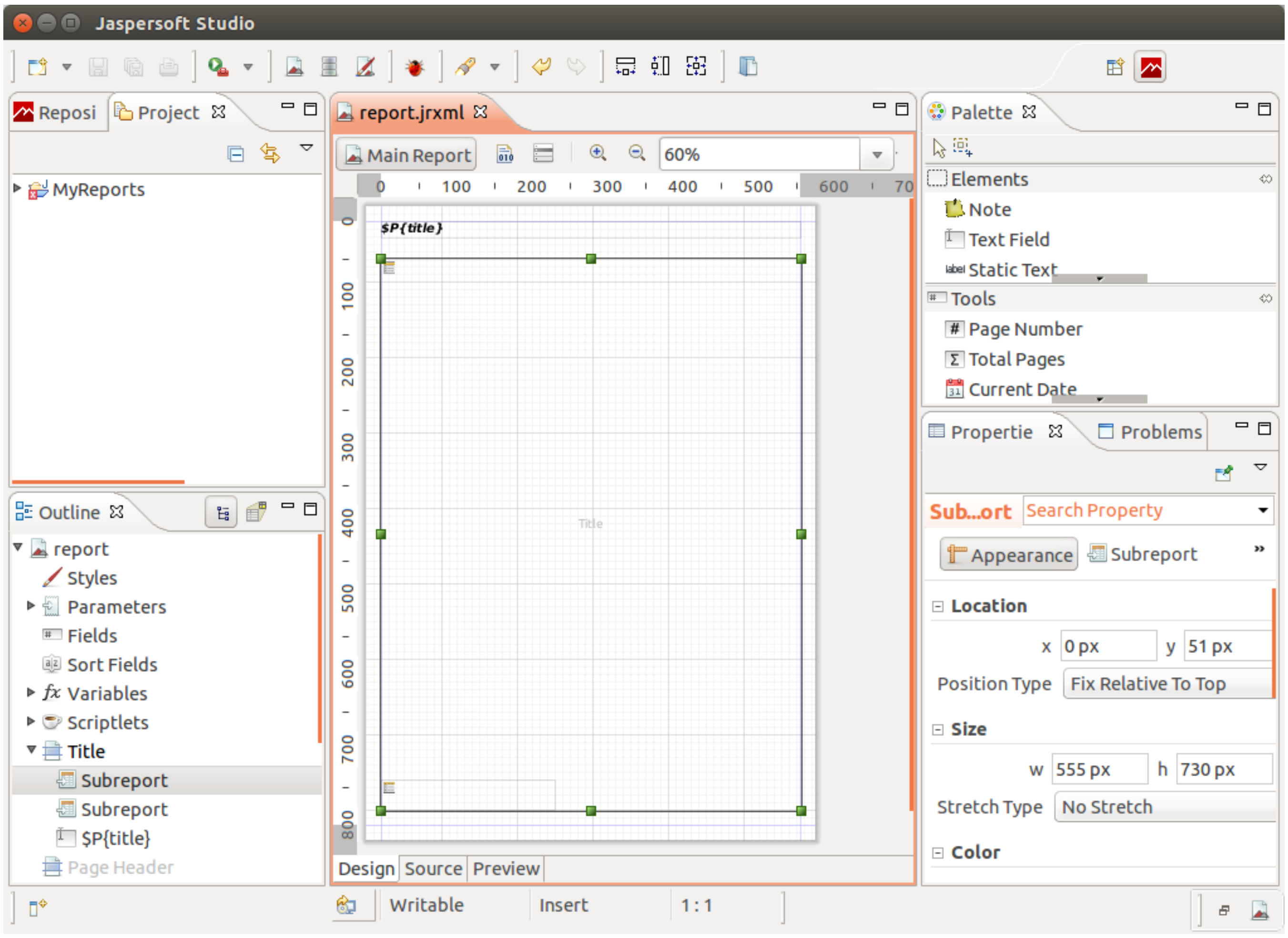
Notes:

We will take a quick look at the print configuration and the request to generate a report. For now, the purpose is not to understand every detail but only to get an idea about the parts that are involved.

We will start with the `config.yaml` file which uses the YAML format. This file configures the available templates for a print configuration. Each template must have a reference to a JasperReports template file (in this case `report.jrml`).

A template has an `attributes` and a `processors` section. The attributes define the input for processors, which do the actual work (e.g. compile report templates or creating map graphics).

Basic configuration (template file)



Location
x 0 px y 51 px
Position Type: Fix Relative To Top
Size
w 555 px h 730 px
Stretch Type: No Stretch
Color

Notes:

The screenshot above shows a template in JasperSoft Studio. The template file defines the layout. E.g. it positions the title above the map area. Output by the processors or attributes (like the title) can be embedded into the template.

Basic configuration (JSON request)

```json
{
  "layout": "A4 Portrait",
  "outputFormat": "pdf",
  "attributes": {
    "title": "Sample Print",
    "map": {
      "projection": "EPSG:900913",
      "dpi": 72,
      "rotation": 0,
      "center": [957352, 5936844],
      "scale": 25000,
      "layers": [
        {
          "type": "osm",
          "baseURL": "http://tile.osm.ch/osm-swiss-style",
          "imageExtension": "png"
        }
      ]
    }
  }
}
```

Notes:

To generate a report, a print request using the JSON format has to be provided. The print request must specify the layout/template that should be used, the output format and further attributes (like the title and extent for the map).

Ressources

- Documentation
- Code: GitHub repository
- Example configurations
- Mailing list
- Template creation: Getting Started with Jaspersoft Studio

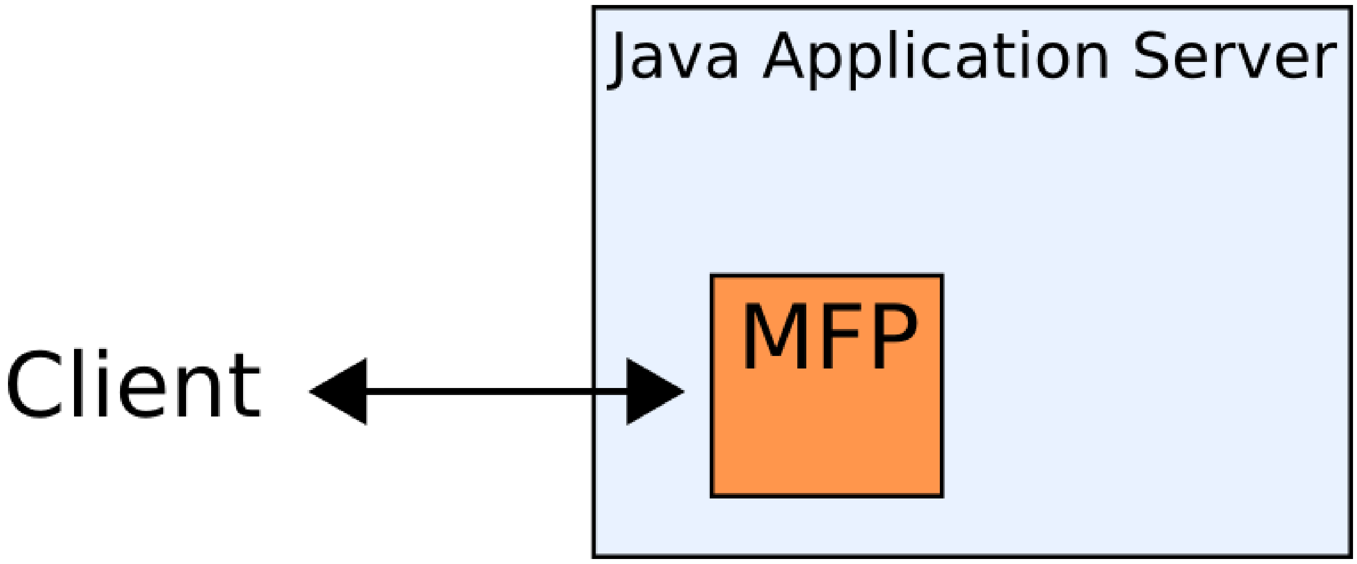
INSTALLATION

Agenda of this chapter

Goal: Setting up MapFish Print and generating a first report.

- MFP as web-application
- MFP as command-line application

Mapfish Print as web-application



Notes:

MapFish Print is provided as Java WAR file which is to be deployed on a Java application server like Tomcat or Jetty.

For the following, we will use a fresh instance of Tomcat. Download Tomcat from the Tomcat website (Binary distributions - Core). Extract the downloaded archive and make the start and stop scripts executable.

```
$ cd apache-tomcat-8.0.36/bin
# chmod +x *.sh
```

We will assume that the Tomcat instance is located at `/home/user/apache-tomcat-8.0.36/`. For convenience we store the path to Tomcat in a variable:

```
$ TOMCAT=/home/user/apache-tomcat-8.0.36/
```

Now try to start Tomcat with:

```
$ $TOMCAT/bin/startup.sh
```

You should be able to open http://localhost:8080 in your browser.

To stop Tomcat again, use:

```
$ $TOMCAT/bin/shutdown.sh
```

The log files of Tomcat are located in `apache-tomcat-8.0.36/logs` (depending on your installation). For example to get the logs for MapFish Print run:

```
$ tail -f $TOMCAT/logs/catalina.out
```

Now, let's deploy an instance of MapFish Print. Download a recent version of the MFP WAR from the MapFish Print website. Rename the downloaded file (e.g. `print-servlet-3.6-SNAPSHOT.war`) to `print.war`.

Then deploy the WAR file by copying it into the `webapps` folder of Tomcat and restart Tomcat:

```
$ mv print-servlet-3.5.0.war print.war
$ cp print.war /home/user/apache-tomcat-8.0.36/webapps/
$ $TOMCAT/bin/startup.sh
```

Open http://localhost:8080/print/ in your browser. You should see a basic interface to test print configurations.

When redeploying MapFish Print it is recommended to restart Tomcat to properly reload all classes.

Exercise 2.1: Installation

- **Setup Tomcat and deploy MapFish Print**
- **Open http://localhost:8080/print/ in a browser and print a report using the example configuration `simple`.**

Customizing the WAR

- Print configurations: `/print-apps`
- Log configuration: `/WEB-INF/classes/logback.xml`
- Configuration parameters: `/WEB-INF/classes/mapfish-spring.properties`

Notes:

To customize the deployed instance of MapFish Print, the WAR file can be modified. A WAR is simply a ZIP archive which can be opened with any archive software. To customize the WAR, unzip the WAR file, do your modifications and then create a new ZIP archive. To ease this task a bash script `update-war.sh` is provided for this workshop. Otherwise similar steps could be integrated in the build system of a project.

The print configurations are located inside a folder `print-apps` of the WAR. To add a custom print configuration, run the following command:

```
$ ./update-war.sh print.war configurations/00-workshop/ \
    print-new.war
```

This adds the folder `configurations/00-workshop/` to the `prints-apps` of the WAR `print.war` and creates a new WAR `print-new.war`

Then deploy the customized WAR:

```
$ $TOMCAT/bin/shutdown.sh
$ cp print-new.war $TOMCAT/webapps/print.war
$ $TOMCAT/bin/startup.sh
```

If you open http://localhost:8080/print/ again, you should see a new print configuration `00-workshop`.

Exercise 2.2: Customizing the WAR

- **Create a custom WAR using the print configuration `00-workshop` with the help of the batch script `update-war.sh`.**
- **Create a report for the print configuration `00-workshop`.**
- **To change the report title, add an attribute `"title": "World Map"` in the request.**

Using the MFP command-line-interface

For testing

```
$ print-cli/bin/print -config config.yaml -spec requestData.json -ou
```

Notes:

Updating a WAR file and redeploying takes some time. During development it is often convenient to use the MapFish CLI (command-line-interface) application to test reports.

Download a recent version of the MFP CLI application from the MapFish Print website. Unzip the downloaded file (e.g. `print-cli-3.5.0-zip.zip`) and rename the folder to `print-cli`.

Then generate a report with:

```
$ print-cli/bin/print \
  -config configurations/00-workshop/config.yaml \
  -spec configurations/00-workshop/requestData.json \
  -output test.pdf
```

Exercise 2.3: Using the CLI application

- **Create a report for the print configuration `00-workshop` with the CLI application.**
- **Change the extent of the printed map by setting a different center and scale.**

YAML CONFIGURATION

Agenda of this chapter

Goal: Understanding the basic structure and elements of the configuration.

- Structure
- Attributes
- Processors
- Overview of available attributes/processors
- Other configuration elements

Basic structure of a configuration file

```
pdfConfig: !pdfConfig
  author: "..."
  subject: "..."

templates:
  A4 Portrait: !template
    reportTemplate: report.jrxml

    attributes:
      title: !string {}
      map: !map
        width: 555
        height: 730
        maxDpi: 300
      scalebar: !scalebar
        width: 230
        height: 40

    processors:
    - !reportBuilder
      directory: '.'
    - !createMap
      inputMapper: {map: map}
      outputMapper: {mapSubReport: mapSubReport}
    - !createScalebar {}
```

Notes:

Each print configuration/app must have a single `config.yaml` file which contains the configuration. This file uses the format YAML. Besides, basic data-structures like strings or numbers, YAML supports lists, mappings (dictionaries or lists of key-value-pairs) and custom types.

For example, the `processors` property contains a list:

```
processors:
- ...
- ...
- ...
```

The `attributes` property is a mapping which contains arbitrary key-value-pairs:

```
attributes:
  title: ...
  map: ...
  scalebar: ...
```

`!pdfConfig`, `!template` or `!map` are custom types that accept only specific properties:

```
map: !map
  width: 555
```

```
      height: 730
      maxDpi: 300
```

Configuration files for Mapfish Print must contain a `templates` property at the root level. Besides, other properties like `pdfConfig` are possible at the root level.

The `templates` property expects a mapping of `!template` instances. The properties that can be set on instances of `!template` are listed in the documentation. At least the properties `reportTemplate`, `attributes` and `processors` should be set. `reportTemplate` contains the filename of the main JasperReports template.

Template attributes

```
attributes:
  title: !string
    default: "Countries"
  description: !string {}
  showHeader: !boolean {}
  map: !map
    width: 555
    height: 730
    maxDpi: 300
  scalebar: !scalebar
    width: 230
    height: 40
```

Notes:

There are two sorts of attributes. Attributes that serve as input for processors and attributes that are directly passed as parameter to the JasperReport template.

For example the attributes `title`, `description` and `showHeader` are not used by a processor, but can be used in the JasperReports template. These attributes can have a default value (like for `title`) or must be provided in the print request.

Most processors have a corresponding attribute type, e.g. the `!createMap` processor has a `!map` attribute type which contains all of the information required to render a map.

These attribute types for processors contain properties that must be given in the configuration file (*configuration* properties) and also properties that can be given in the print request (*input* properties). For example the map size (width/height) must be given in the configuration file, because the map size is fixed. Other properties like the layers that should be shown can be given in the print request.

The properties of an attribute that must be given in the configuration file are listed under the `Configuration` section of an attribute in the documentation. The properties that can be given in the print request are listed under the `Inputs` section. See for example the documentation for the !map attribute.

For the *input* properties, a default value can be defined in the configuration file. For example the following configuration sets a default projection for all maps. If the projection is not given in a print request, the default projection will be used.

```
attributes:
  map: !map
    width: 555
    height: 730
    maxDpi: 300
    default:
      projection: "epsg:3857"
```

Template processors

```
processors:
- !reportBuilder
  directory: '.'
- !createMap
  inputMapper: {map: map}
  outputMapper: {mapSubReport: map}
- !createScalebar {}
```

Notes:

When a print job is started, attributes are passed to the processors. The processors work with the attributes in order to generate the maps, tables or legends that are required to generate the report.

Some processors expect *configuration* properties that are directly defined on the processor. For example the `reportBuilder` processor, which compiles JasperReports templates, has a `directory` property.

Otherwise, processors get their input from the defined attributes and generate new attributes that are available in the JasperReports template. For example the `!createMap` processor expects a `!map` attribute and generates a `mapSubReport` attribute that can be referenced in the JasperReports template.

The processors expect that attributes are available under a certain name. For example the `!createMap` processor expects an attribute named `map`. To avoid name conflicts (for example when using more than one map in a report), input and output mappers can be defined. In the example below an attribute `map1` is mapped to the `map` attribute that the `!createMap` processor expects, and the output `mapSubReport` is mapped to an attribute `mapSubReport1`.

```
attributes:
  map1: !map
    width: 555
    height: 730
    maxDpi: 300
processors:
- !createMap
  inputMapper: {map1: map}
  outputMapper: {mapSubReport: mapSubReport1}
```

The `inputMapper` and `outputMapper`configuration can be left out, if the attributes match. For example:

```
processors:
- !createScalebar {}
```

Available processors

- createMap
- createScalebar
- createNorthArrow
- createOverviewMap
- prepareLegend
- prepareTable
- createDataSource
- ...

Documentation: Processors

Other configuration elements

- pdfConfig: PDF Metadata

Documentation: pdfConfig

```
pdfConfig: !pdfConfig
  author: "..."
  subject: "..."
```

- proxy: Proxy for requests

Documentation: proxy

```
proxies:
  - !proxy
    scheme: http
    host: proxy.host.com
    port: 8888
    username: username
    password: xyzpassword
    matchers:
      - !localMatch
        reject: true
      - !dnsMatch
        host: www.camptocamp.com
        reject: true
      - !acceptAll {}
```

Proxy all requests except localhost and www.camptocamp.com

Configure HTTP requests

- Restrict access only to certain addresses
- Add or forward headers (e.g. authentication headers)
- Convert URLs (e.g. from `http://domain.com/tiles` to `http://localhost:1234`)

Example

```
- !configureHttpRequests
  httpProcessors:
    # change myhost.com urls to localhost
    - !mapUri
      mapping:
        (http)://myhost.com(.*) : "$1://localhost$2"
    # forward headers
    - !forwardHeaders
      headers: [Cookie, Referrer]
    # only allow localhost requests, any other requests
    # will be rejected
    - !restrictUris
      matchers: [!localMatch {}]
```

Documentation: configureHttpRequests

Notes:

The `configureHttpRequests` processor allows to control and configure the HTTP requests that are made by Mapfish Print. For example authentication headers can be forwarded, URLs can be rewritten from addresses with a domain name to IP addresses (to avoid a DNS resolution step) and access to certain addresses can be restricted.

It is important to configure the restrictions correctly to avoid that internal services are exposed.

Exercise 3.1: YAML Configuration

- **Make a copy of the print configuration `00-workshop` and for example name the folder `01-exercise-config`.**
- **Add a new attribute `description` (string) in `config.yaml` without a default value. Also set the attribute in the print request (`requestData.json`).**
- **Configure a scalebar processor, including a scalebar attribute.**
- **For now, do not show the description attribute and the scalebar in the JasperReports template. But test if the configuration is correct by generating a report (for example with the CLI application).**

REPORT TEMPLATES

Agenda of this chapter

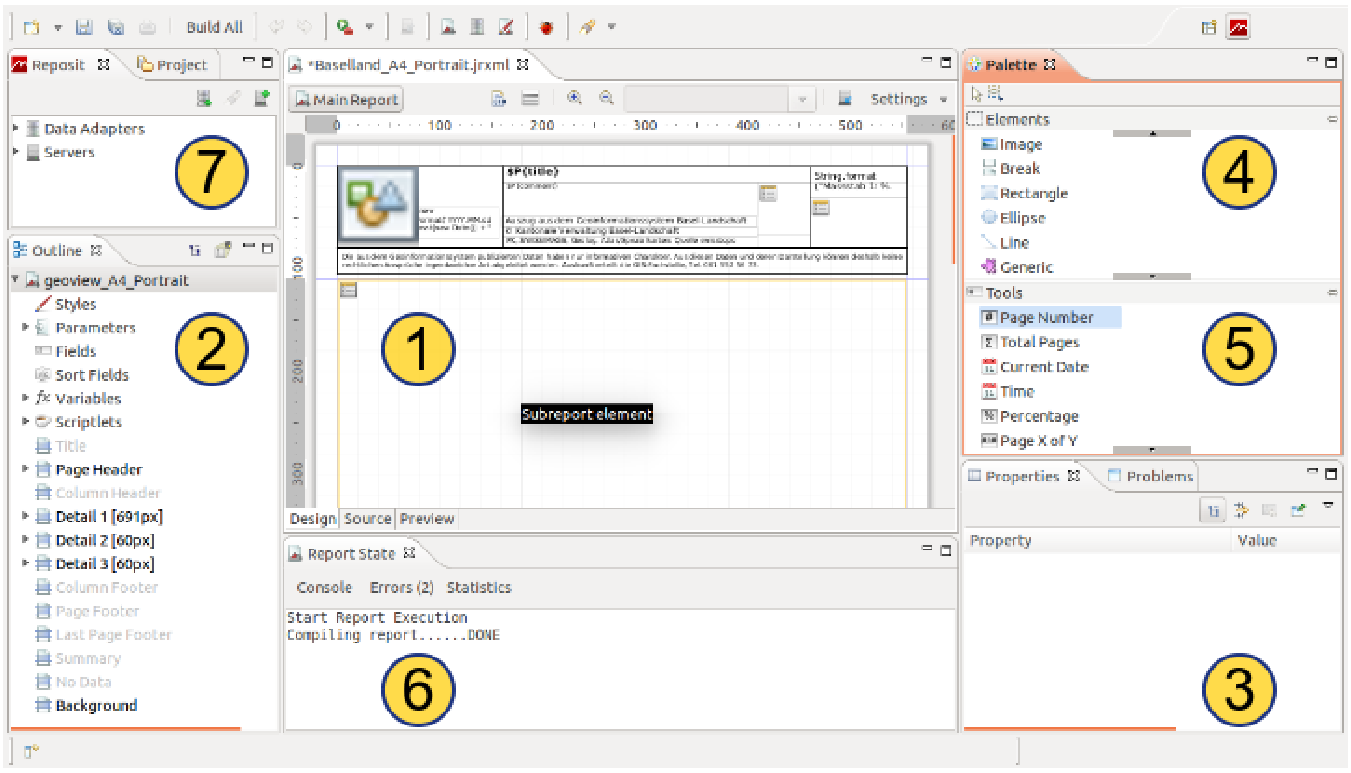
Goal: Learn how the YAML configuration and the templates are connected and how to use JasperSoft Studio to edit the templates.

- Structure, concepts
- Static text
- Images
- Text fields with expressions (date, page)
- Text fields with attribute values
- How is the map included?

Report Structure: JRXML files

- Basically an XML
- Called "template" of the print service
- Skeleton which will be filled by data from a request
- Can be edited in a simple text editor or easier with JasperSoft Studio (WYSIWYG editor)

JasperSoft Studio



- Editor / Source / Preview (1)
- Template outline (2)
- Properties of selected items (3)
- Elements available for insertion in the report (4)
- Shortcuts to pre-configured fields (5)
- Status (6)
- Project view (7)

Static Text Elements

Used to show static text defined at design time.

```
<staticText>
  <reportElement x="12" y="443" width="68" height="20" uuid="..."/>
  <textElement>
    <font fontName="Arial" size="12"/>
  </textElement>
  <text><![CDATA[Some static text]]></text>
</staticText>
```

Can be used for:

- labels
- descriptions
- copyright information

Text Fields with Expressions

Text fields leave you with more options than the static text element especially in terms of:

- size of the field
- what is contained in the field

These elements can be used for:

- dates
- page numbers

```
"Lausanne, " + new
SimpleDateFormat("dd.MM.YYYY
HH:mm").format(new Date()) + " Uhr"
```

$\longrightarrow$

```
Lausanne, 25.6.2016 12:43 Uhr
```

Text Fields with Expressions

```
<textField>
  <reportElement x="592" y="440" width="208" height="30" uuid="...",
  <textElement>
    <font fontName="Arial" size="12"/>
  </textElement>
  <textFieldExpression>
    <![CDATA[
      "Created: " +
      new SimpleDateFormat("dd.MM.YYYY HH:mm").format(
        new Date())
    ]]>
  </textFieldExpression>
</textField>
```

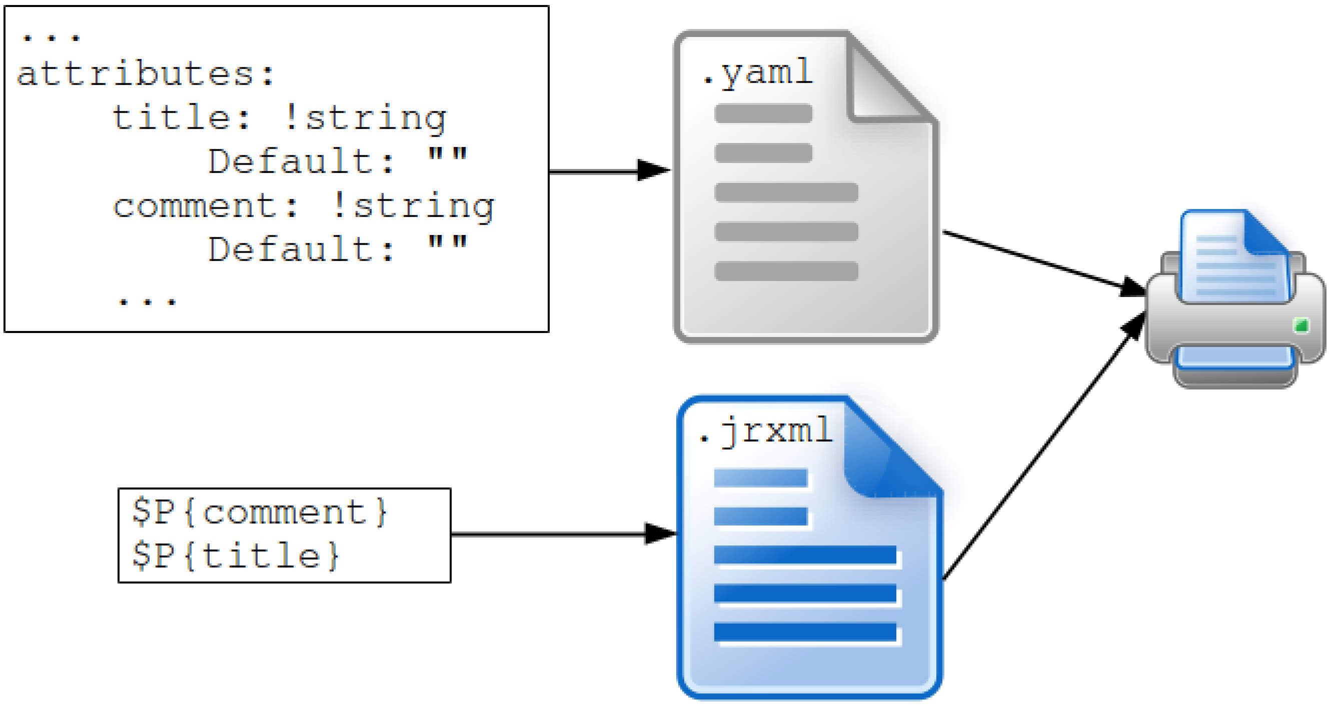
Text Fields with Attribute Values

Attribute values from the `config.yaml` can be added as attributes in the report.

This can be used to:

- Add comments
- Add custom titles
- Show/hide elements (e.g. `showFooter`).

The attribute used in the text field needs to be defined in the `config.yaml` and included as a parameter in the `.jrxml` file.



```
...
attributes:
    title: !string
        Default: ""
    comment: !string
        Default: ""
    ...
```

```
$P{comment}
$P{title}
```

Text Fields with Attribute Values

```
<parameter name="title" class="java.lang.String"/>
...
<textField>
  <reportElement x="0" y="1" width="800" height="50" uuid="..."/>
  <textElement textAlignment="Center">
    <font size="36"/>
  </textElement>
  <textFieldExpression><![CDATA[$P{title}]]></textFieldExpression>
</textField>
```

Images

Most commonly images are used to insert raster images (such as GIF, PNG and JPEG).

This can be used to add:

- logos
- all kind of graphics

The images must be inside the folder of the print configuration.

```
<image hAlign="Left" vAlign="Middle">
  <reportElement x="90" y="424" width="183" height="50" uuid="..."/
  <imageExpression><![CDATA["logo.png"]]></imageExpression>
</image>
```

Including a Map

- A map is included as a sub-report element (report in a report).
- The sub-report is created by the `!createMap` processor.
- The map properties are defined in the `config.yaml`.
- The layers shown on the map are defined in the print request.

Including a Map

The parameter `map` contains the path to the sub-report.

```
<parameter name="map" class="java.lang.String"/>
...
<subreport>
  <reportElement x="0" y="94" width="800" height="330" uuid="...">
    <property name="local_mesure_unitwidth" value="pixel"/>
    <property name="com.jaspersoft.studio.unit.width" value="px"/>
    <property name="local_mesure_unitheight" value="pixel"/>
    <property name="com.jaspersoft.studio.unit.height" value="px"/>
  </reportElement>
  <subreportExpression><![CDATA[$P{map}]]></subreportExpression>
</subreport>
```

The size of the sub-report should match the size of the map configured in `config.yaml`.

Resources

Documentation: Getting Started with Jaspersoft Studio

Jaspersoft Community

Exercise 4.1: Installing JasperSoft Studio

- **Download the latest version of JasperSoft Studio from here: http://community.jaspersoft.com/project/jaspersoft-studio/releases**
- **Extract the file.**
- **To start JasperSoft Studio open a shell and type**

```
cd folder/containing/TIBCOJaspersoftStudio-X.X.X.final
./runubuntu.sh
```

Notes:

Note: On Ubuntu 16.04 when using the default desktop manager, there might be problems with the interface of JasperSoft Studio (no outline, preview is not updated, ...). In this case add the line `export SWT_GTK3=0` at the top of `runubuntu.sh` so that the file looks like this:

```
#!/bin/bash
export SWT_GTK3=0
DIR=$(dirname "$0")
export UBUNTU_MENUPROXY=0;
"$DIR"/Jaspersoft\ Studio $*
```

Exercise 4.2: Templates

This exercise will be based on the exercise from chapter 3 (configuration). Either continue with your own configuration or start with the solution of that exercise located at `configurations/01-exercise-config-solution`. Copy the configuration to a new folder `02-exercise-template`.

- **Add a static text field below the map with the text "Copyright: ...".**
- **Add a text field showing the date and time.**
- **Add a logo to the report (copy the file `02-exercise-template-solution/logo.png` into your configuration folder).**
- **Show the value of the attribute `description` (which was defined in the previous exercise in the `config.yaml`).**
- **Show the sub-report of the scalebar that was configured in the previous exercise. Use `scalebarSubReport` as parameter name and the size 230x40px.**

Countries

0
500
1500km

camptocamp

Copyright ... 28.06.2016 17:16

REFERENCING GEO-DATA IN A PRINT REQUEST

Agenda of this chapter

Goal: Learn how geo-data can be referenced in a request.

- Supported geo-data
- Styling vector data

Geo-data in a print request

```json
{
  "layout": "A4 Portrait",
  "outputFormat": "pdf",
  "attributes": {
    "title": "Sample Print",
    "map": {
      "projection": "EPSG:3857",
      "dpi": 72,
      "rotation": 0,
      "center": [957352, 5936844],
      "scale": 25000,
      "layers": [
        {
          "type": "osm",
          "baseURL": "http://tile.osm.ch/osm-swiss-style",
          "imageExtension": "png"
        }
      ]
    }
  }
}
```

Notes:

The actual geo-data that should be shown in a map is usually referenced in the map attribute of the print request.

In the above example a tile layer with the standard Mercator tiling scheme (layer type: `osm`) is used.

Supported layer types

- GeoJSON
- GML/WFS
- GeoTIFF
- WMS and tiled WMS
- WMTS
- XYZ/OSM

Documentation: Layers

Example WMS

```
"map": {
  "projection": "EPSG:21781",
  "dpi": 180,
  "rotation": 0,
  "center": [615928, 174957],
  "scale": 1000000,
  "layers": [
    {
      "type": "WMS",
      "layers": ["ch.swisstopo.pixelkarte-farbe-pk1000.noscale"],
      "baseURL": "http://wms.geo.admin.ch/",
      "imageFormat": "image/jpeg",
      "version": "1.1.1",
      "customParams": {
      }
    }
  ]
}
```

Notes:

When using layer type `WMS` a single request is made to fetch the layer image for the map. When using higher DPI values, the size of the request image might exceed the maximum size of the WMS server. In that case, layer type tilewms can be used instead, which makes multiple requests to the WMS server.

Example GeoJSON

```
"map": {
  "longitudeFirst": true,
  "center": [5, 45],
  "scale": 100000000,
  "projection": "EPSG:4326",
  "dpi": 72,
  "rotation": 0,
  "layers": [
    {
      "type": "geojson",
      "geoJson": "file://countries.geojson",
      "style": {
        "version": "2",
        "*": {
          "symbolizers": [
            {
              "type" : "polygon",
              "fillColor": "#5E7F99",
              "fillOpacity": 1,
              "strokeColor": "#CC1D18",
              "strokeOpacity": 1,
              "strokeWidth": 1
            }
          ]
        }
      }
    }
  ]
}
```

Notes:

The GeoJSON layer allows to show vector data in the map. The property `geoJson` can either contain an URL to a GeoJSON file/service, a file name to a GeoJSON file inside the configuration folder or directly GeoJSON data.

```
{
  "type": "geojson",
  "geoJson": {
    "type": "FeatureCollection",
    "features": [{
        "type": "Feature",
        "properties": {
          "name": "Boat"
        },
        "geometry": {
            "type": "Point",
            "coordinates": [957352, 5936844]
        }
    }]
  }
}
```

Vector styling

- With SLD styles
- Mapfish JSON Style Version 1 (similar to OpenLayers 2 styles)
- Mapfish JSON Style Version 2 (similar to SLD)

Documentation: Styles

SLD styles

`style.sld`

```
<?xml version="1.0" encoding="UTF-8"?>
<StyledLayerDescriptor ...>
  <NamedLayer>
    <Name>countries_style</Name>
    <UserStyle>
      <FeatureTypeStyle>
        <Rule>
          <PolygonSymbolizer>
            <Stroke>
              <CssParameter name="stroke">#CC1D18</CssParameter>
              <CssParameter name="stroke-width">1</CssParameter>
            </Stroke>
            <Fill>
              <CssParameter name="fill">#5E7F99</CssParameter>
            </Fill>
          </PolygonSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

In request:

```
  "style": "file://style.sld"
```

Documentation: SLD Reference

SLD Cookbook

Version 2 JSON Styles

```
"style": {
  "version": "2",
  "*": {
    "symbolizers": [
      {
        "type" : "polygon",
        "fillColor": "#5E7F99",
        "fillOpacity": 1,
        "strokeColor": "#CC1D18",
        "strokeOpacity": 1,
        "strokeWidth": 1
      },
      {
        "type": "text",
        "label": "[name]"
      }
    ]
  }
}
```

Notes:

Version 2 JSON styles have a similar structure as the SLD format. The style consists of multiple styling rules with a filter condition. In the above example, the filter condition is * which matches all features. Each styling rule can have multiple symbolizers. In this case a polygon symbolizer is used. There are point, line, polygon and text symbolizers.

In a filter condition, properties of features can be referenced. For example `[in('FRA')]` selects all features with the feature identifier `FRA`. Or `[population > 100000]` selects all feature where the value for `population` is greater than 100000.

Exercice 5.1: Geo-data

- Using one of the previous configurations, print a map in the projection 'epsg:3857' at a location and scale of your choice using an `osm` layer with the URL `http://tile.thunderforest.com/cycle/`. Note that the center coordinate must be given in the projection of the map (you can use http://epsg.io/3857 to get a coordinate in the right projection).

- Add a red point as `geojson` layer.

Static overlay and background layers

Static layers defined in `config.yaml`

```
attributes:
  map: !map
    ...
  overlayLayers: !staticLayers
    default:
      layers:
        - type: "grid"
          numberOfLines: [10, 10]
          labelColor: rgba(0,0,0,0)
          haloColor: rgba(0,0,0,0)
processors:
- !addOverlayLayers
  inputMapper:
      overlayLayers: staticLayers
      map: map
- !createMap {}
```

Example

Notes:

If all reports generated with a template should contain the same background or overlay layers, these static layers can be defined in the configuration file.

Questions & Next Steps

Still Learning:

- Tables
- Legends
- Data-sources (e.g. arbitrary number of maps in a report)
- Integrate with external data-sources (e.g. databases)
- Integration in UI
- ...

PROPOSED SOLUTIONS

Solution 2.1: Installation

Follow the instructions given in the slide notes.

Solution 2.2: Customizing the WAR

Follow the instructions given in the slide notes.

Solution 2.3: Using the CLI application

Follow the instructions given in the slide notes.

Solution 3.1: YAML Configuration

The solution is given in `configurations/01-exercise-config-solution`.

A test report can be generated with:

```
$ print-cli/bin/print \
  -config configurations/01-exercise-config-solution/config.yaml \
  -spec configurations/01-exercise-config-solution/requestData.json \
  -output test.pdf
```

Solution 4.1: Installing JasperSoft Studio

No solution for this exercise.

Solution 4.2: Templates

The solution is given in `configurations/02-exercise-template-solution`.

A test report can be generated with:

```
$ print-cli/bin/print \
  -config configurations/02-exercise-template-solution/config.yaml \
  -spec configurations/02-exercise-template-solution/requestData.json \
  -output test.pdf
```

Solution 5.1: Geo-data

A possible solution is in file: `configurations/03-geodata/requestData-exercise-solution.json`.

WWW.CAMPTOCAMP.COM

camp

Open Source specialist, Camptocamp consists of three divisions: GEOSPATIAL SOLUTIONS, BUSINESS SOLUTIONS and INFRASTRUCTURE SOLUTIONS. Our professional, innovative and responsive services help you implement your most ambitious projects.

SWITZERLAND	Quartier de l'Innovation EPFL / PSE-A / CH-1015 Lausanne / Tel +41 21 619 10 10
FRANCE	Savoie Technolac / BP 352 / F- 73372 Le Bourget-du-Lac / Tel +33 4 58 48 20 00
Germany	Linuxland GmbH / Thomas-Dehler-Str. 9 / 81737 München / +49 89 99 34 14 40

Follow us