camptocamp

INNOVATIVE SOLUTIONS
BY OPEN SOURCE EXPERTS

MAPFISH PRINT 3

STUDENT GUIDE

Workshop

TABLE OF CONTENTS

Introduction
Installation
YAML Configuration
Report Templates
Referencing geo-data in a print request
More processors
Integration in an UI
Datasource processor
External datasources
Supplemental exerices

À propos de Camptocamp

Société spécialiste en Open Source, innovante dans le développement logiciel :

- Systèmes d'Information Geographique (**SIG**)
- Business Management (**ERP**)
- Gestion de serveur (**IT Automation and Orchestration**)

Présent dans trois pays :

- Suisse (Camptocamp SA)
- France (Camptocamp France SAS)
- Allemagne (Linuxland GmbH)

Département geospatial :

- Contributeurs/commiter sur les projets OpenLayers, GeoNetwork, MapServer, QGIS, ...
- Éditeur des produits geOrchestra et GeoMapfish
- Développement à la carte de produits métiers basé sur QGIs, GeoMapfish, geOrchestra

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?

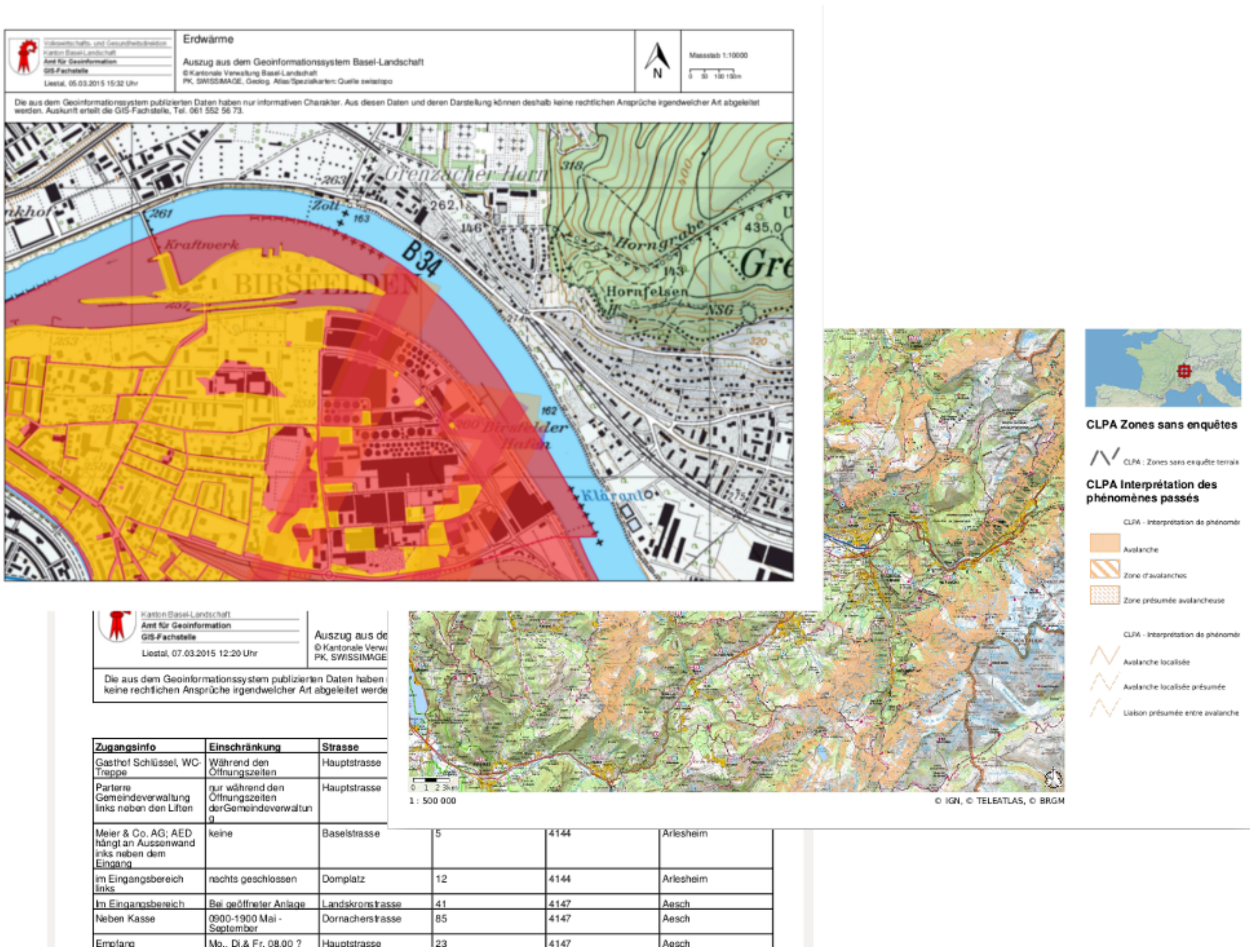
- Goal of the project: Create reports that contain maps and map related components
- A Java library and web-application

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

CLPA Zones sans enquêtes

/\/	CLPA : Zones sans enquête terrain

CLPA Interprétation des phénomènes passés

	CLPA - Interprétation de phénomèr
🟧	Avalanche
▨	Zone d'avalanches
▒	Zone présumée avalancheuse
	CLPA - Interprétation de phénomèr
/\/	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.

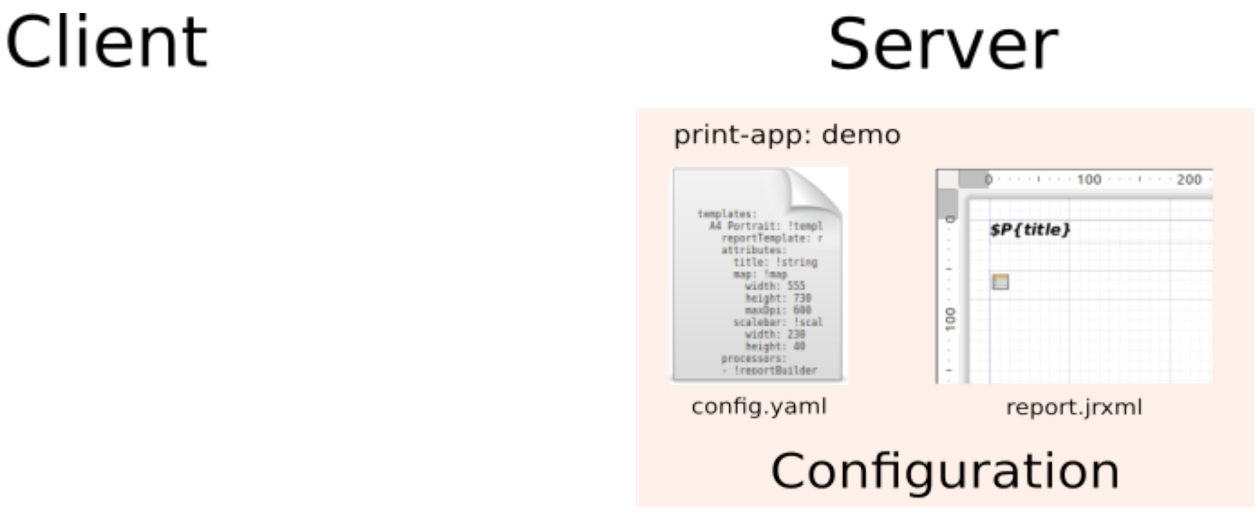
GeoTools, which for example is also powering GeoServer, is used for everything related to mapping. It is used to generate the map layer graphics, that are embedded into the report.

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 ar
API via its web-application and offers reports widgets like scalebars, overview-maps or
legends.

Print process: Configuration



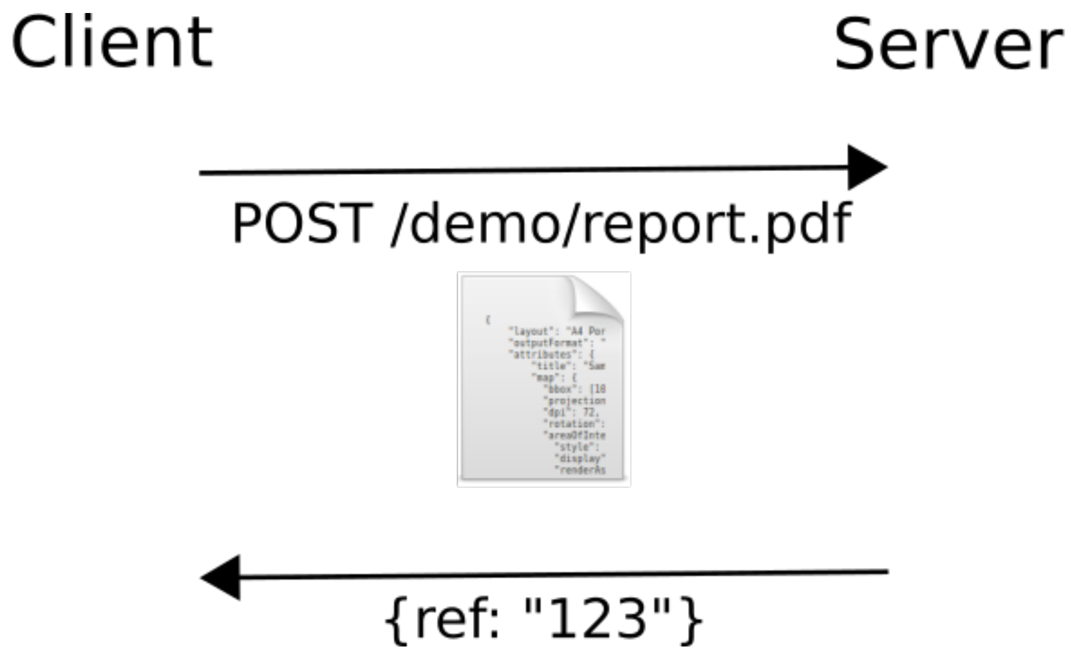
```
templates:
  A4 Portrait: !templ
    reportTemplate: r
    attributes:
      title: !string
      map: !map
        width: 555
        height: 730
        maxDpi: 600
      scalebar: !scal
        width: 230
        height: 40
    processors:
    - !reportBuilder
```

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



```
{
    "layout": "A4 Por
    "outputFormat": "
    "attributes": {
        "title": "Sam
        "map": {
          "bbox": [10
          "projection
          "dpi": 72,
          "rotation":
          "areaOfInte
            "style":
            "display"
            "renderAs
```

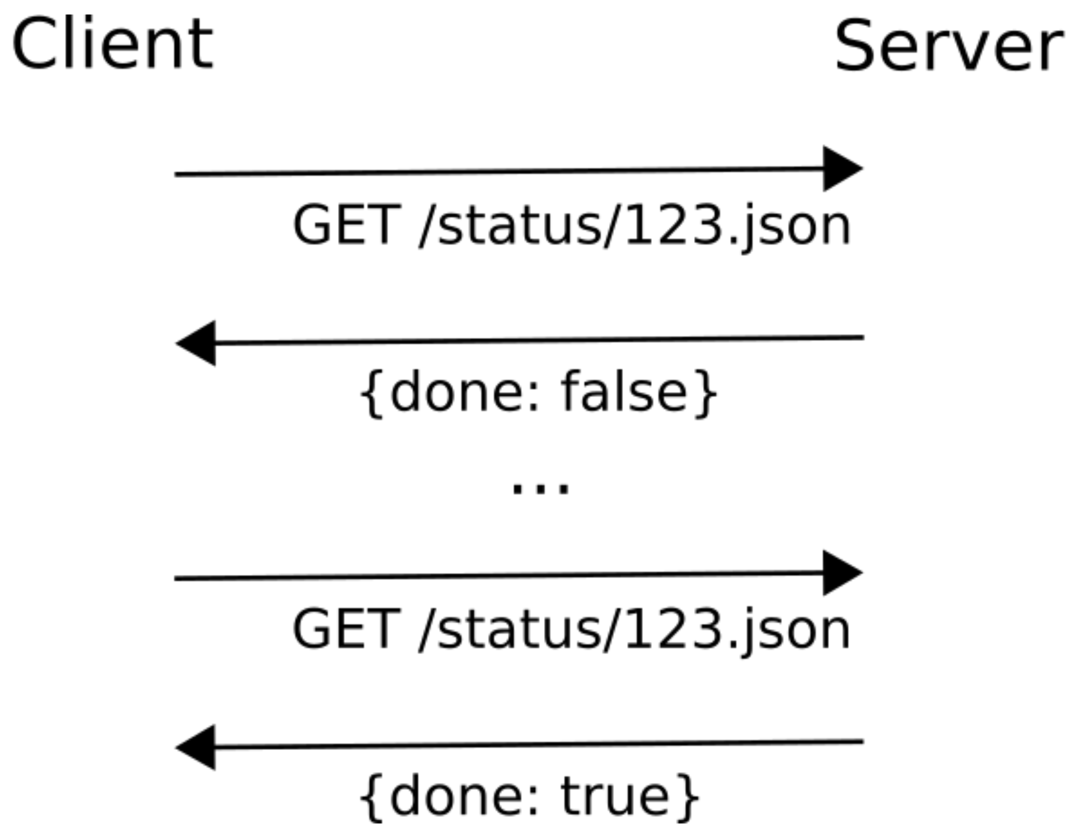
Notes:

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 th 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.

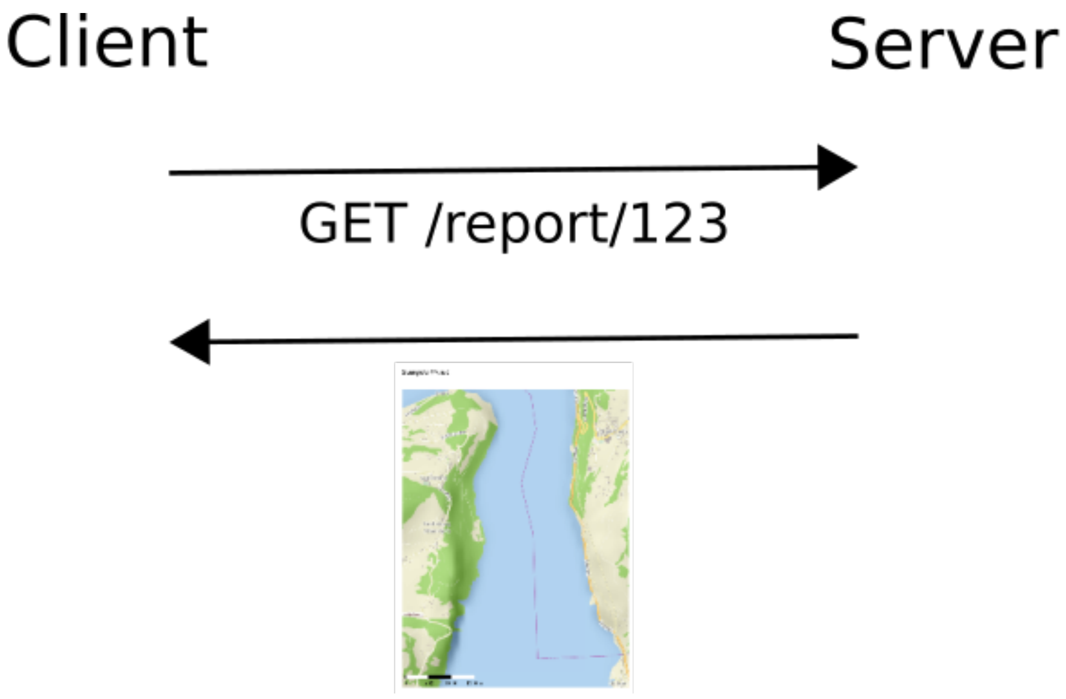
Print process: Polling the status of a print job



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)

```
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 {}
```

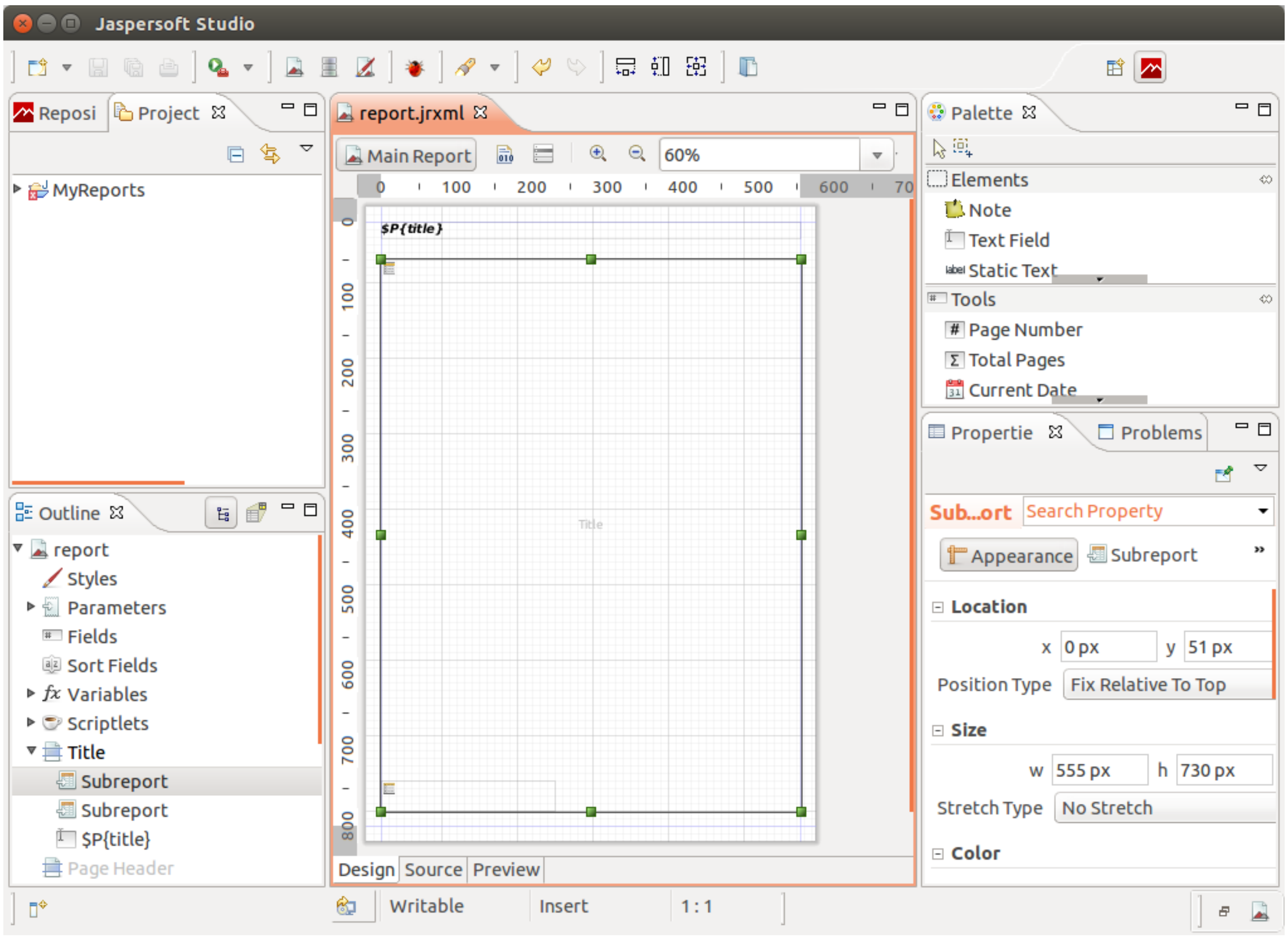
Notes:

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)

```
{
  "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

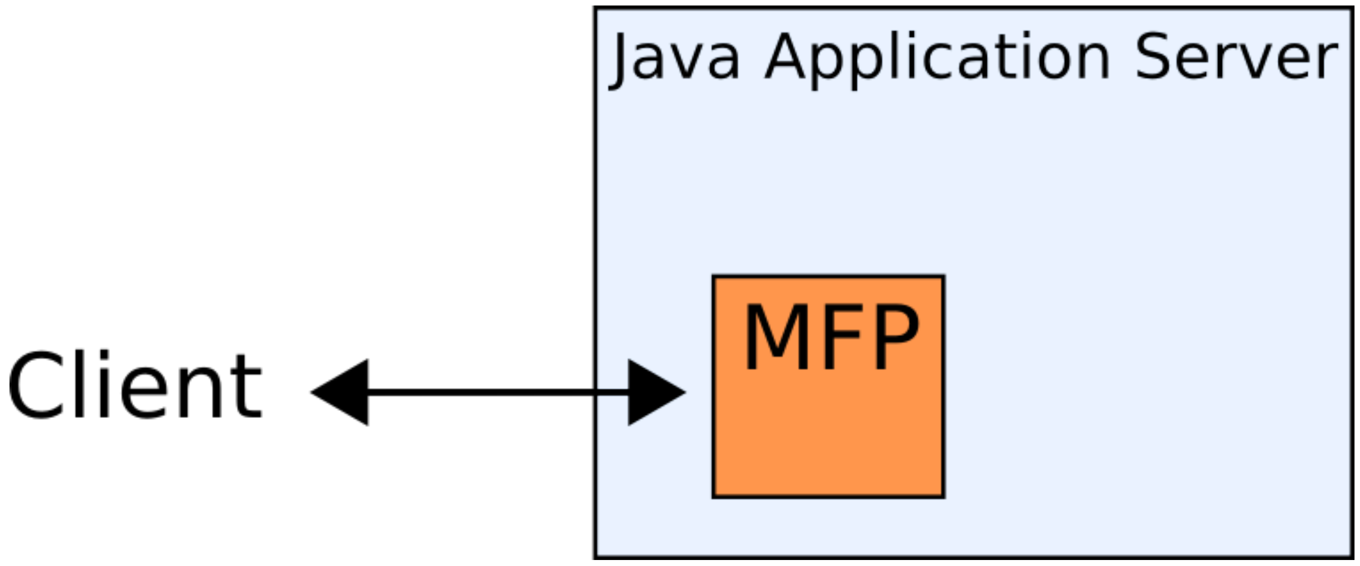
INSTALLATION

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 serve
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 anc
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`.**

Solution 2.2: Installation

Follow the instructions given in the slide notes.

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 tas a bash script `update-war.sh` is provided for this workshop. Otherwise similar steps could b integrated in the build system of a project.

The print configurations are located inside a folder `print-apps` of the WAR. To add a custor 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 2.3: 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.**

Solution 2.4: Customizing the WAR

Follow the instructions given in the slide notes.

Using the MFP command-line-interface

For testing

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

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 2.5: 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.**

Solution 2.6: Using the CLI application

Follow the instructions given in the slide notes.

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
```

Documentation: Attributes

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 defau 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 giver 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 wit 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 attribute 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 i 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 exampl when using more than one map in a report), input and output mappers can be defined. In th 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 `mapSubReport`

```
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.

REPORT TEMPLATES

Agenda of this chapter

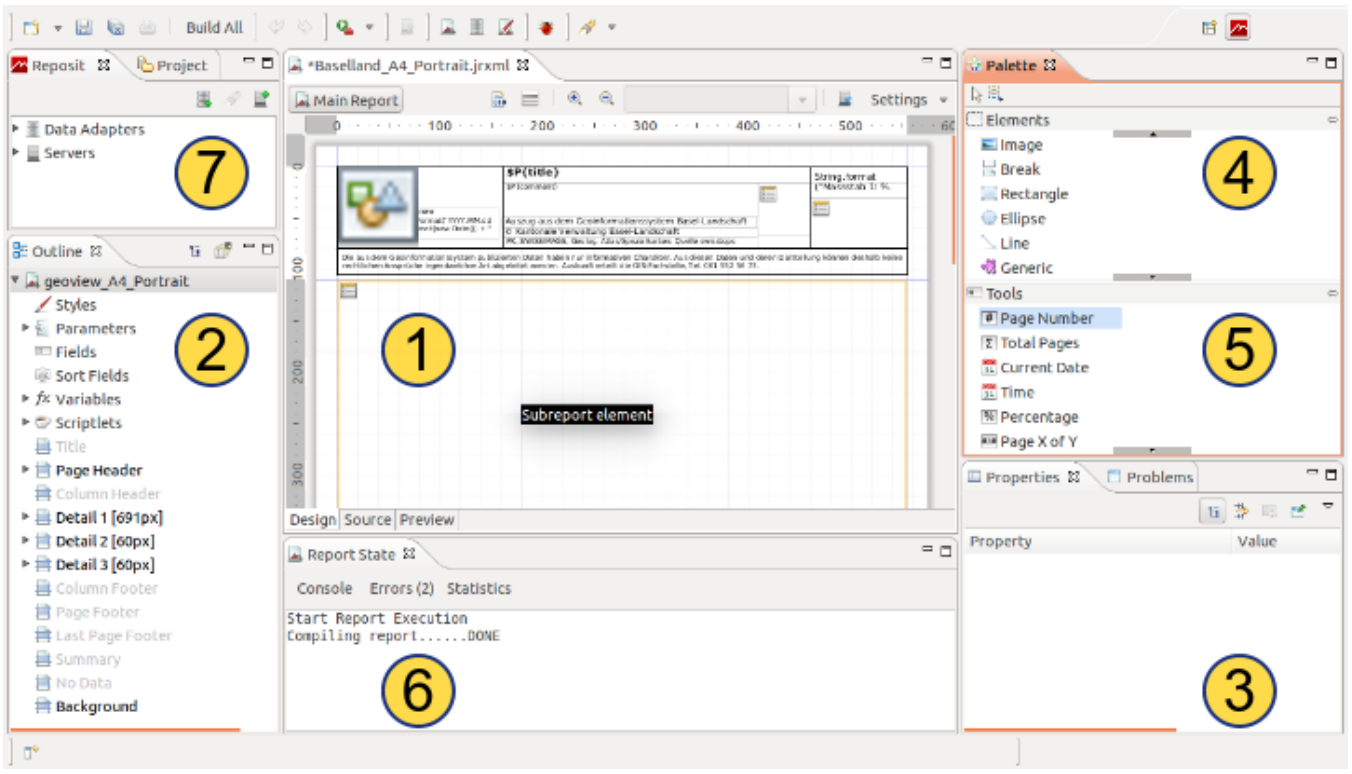
Goal: Learn how the YAML configuration and the templates are connected and how to use JasperSoft Studio to edit the templates.

- 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



```
Start Report Execution
Compiling report......DONE
```

- 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

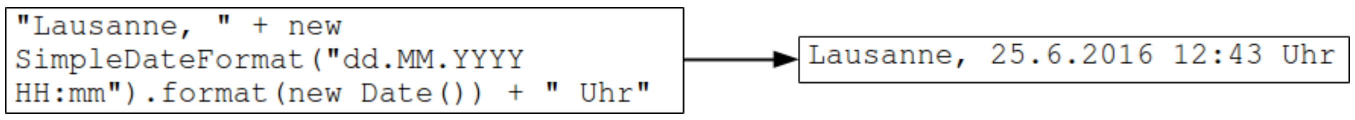
Text Fields with Expressions

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"
```

```
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>
```

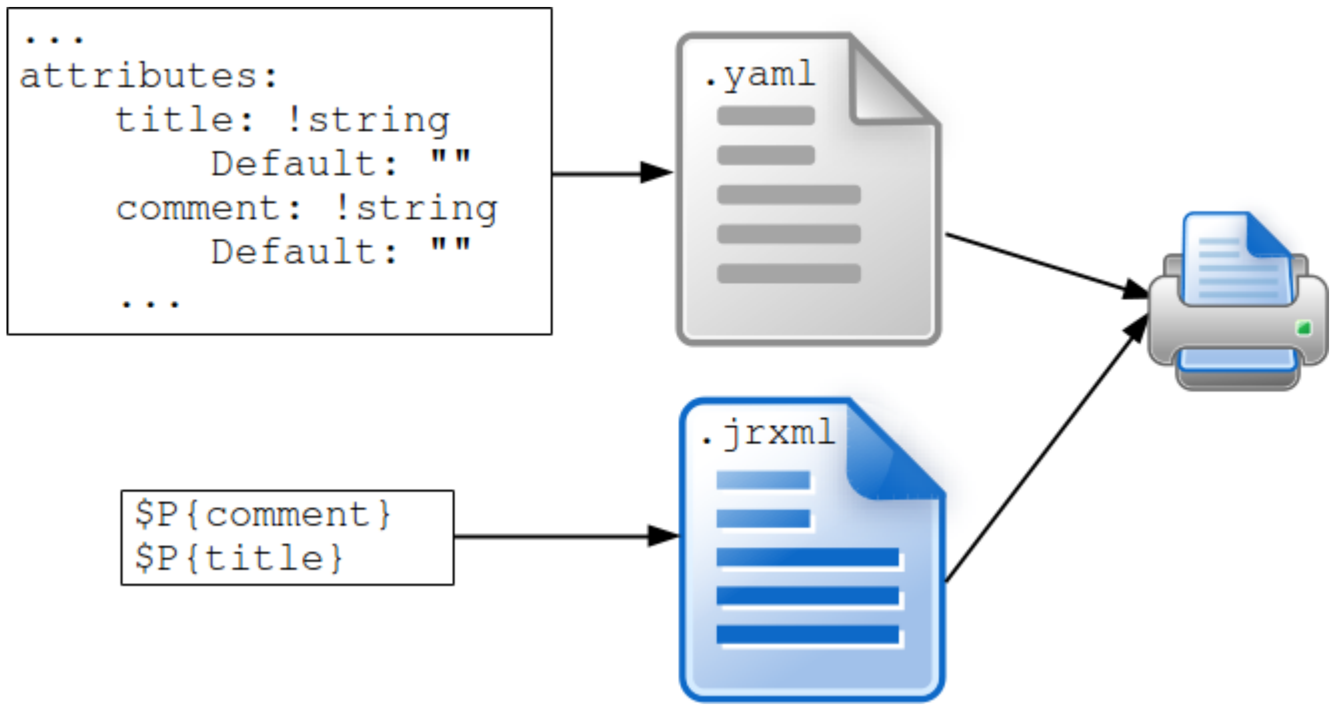
Text Fields with Attribute Values

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 $*
```

Solution 4.2: Installing JasperSoft Studio

No solution for this exercise.

Exercise 4.3: Empty template

Download a basic example:

- **Download the basic example: https://geomapfish-demo.camptocamp.net/bl_empty.zip**
- **Look for the existing static texts, text fields, images, general structure, etc. in the .jrxml file.**

Solution 4.4: Empty template

No solution for this exercise.

Exercise 4.5: First print

Do your first print:

- **Download the print cli application (snapshot of mapfish-print): https://geomapfish-demo.camptocamp.net/core-3.9-SNAPSHOT.zip**
- **Move into the folder the print you report with (you may adapt the path):**

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

Solution 4.6: First print

You can see the result in the expected output of the downloaded basic_example.zip

Exercise 4.7: Add a map

Add a map:

- **Download the request data JSON with map data: https://geomapfish-demo.camptocamp.net/requestData-map.json**
- **Add a "map" attributes and a "createMap" processor to the config.yaml file**
- **Add a "mapSubReport" into the jrxml file with jasper report studio.**
- **Print your report with the map.**

Solution 4.8: Add a map

Solution: https://geomapfish-demo.camptocamp.net/bl_map.zip

Diff: https://github.com/sbrunner/mapfish-print/commit/348d4da658d6c3984b7ee2366850f2669b02f43b

Exercise 4.9: Add data

Add data:

- **Add the gemeinde, parzelle, pointX, pointY attributes in the json file.**
- **Configure theses attributes into the config.yaml file.**
- **Use them into jasper report studio.**
- **Print your report with these attributes.**

Solution 4.10: Add data

Solution: https://geomapfish-demo.camptocamp.net/bl_attributes.zip

Diff: https://github.com/sbrunner/mapfish-print/commit/b6f6a3d3198dc14ef80ca96267f1c9a1d3fad190

REFERENCING GEO-DATA IN A PRINT

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

```
{
  "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 attribut 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 siz 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.

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.

MORE PROCESSORS

Agenda of this chapter

Goal: Get to know more processors.

- North-arrow processor
- Overview map processor
- Legend processor
- Table processor

North-arrow processor

- Allows to add a north-arrow to a report.
- Rotates with the map (points to the north of the map).
- Image files such as SVG, png, jpg,... can be used for the base image of the north-arrow.

Documentation: createNorthArrow

North-arrow processor

Example configuration:

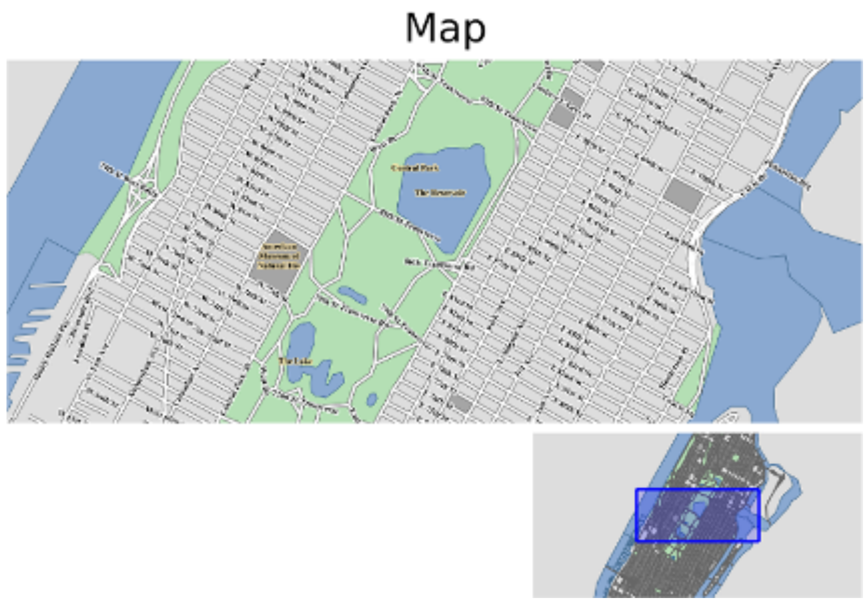
```
attributes:
  ...
  northArrow: !northArrow
    size: 50
    default:
      graphic: "north.svg"

processors:
  ...
- !createNorthArrow {}
```

To place the north-arrow in the template, add a parameter and use a sub-report element:

```
<parameter name="northArrowSubReport" class="java.lang.String"/>
  ...
<subreport>
  <reportElement x="727" y="1" width="50" height="50" uuid="...">
    <property name="com.jaspersoft.studio.unit.width" value="pixel"/>
    <property name="com.jaspersoft.studio.unit.height" value="pixel"/>
  </reportElement>
  <subreportExpression>
    <![CDATA[$P{northArrowSubReport}]]>
  </subreportExpression>
</subreport>
```

Overview map processor



- Adds an overview map for a map.
- The overview map does not need to have the same rotation as the main map.
- It can use the same layers as the main map or custom layers.
- Internally the same processor as to create a map is used (`CreateMapProcessor`).

Documentation: createOverviewMap

Overview map processor

Example configuration:

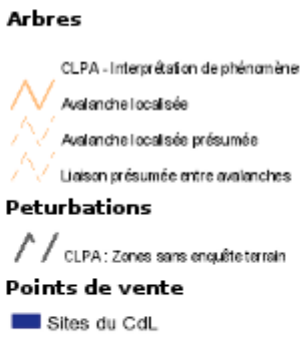
```
attributes:
  ...
  overviewMapDef: !overviewMap
    width: 300
    height: 200
    maxDpi: 400

processors:
  ...
  - !createOverviewMap
    inputMapper: {
      mapDef: map,
      overviewMapDef: overviewMap
    }
    outputMapper: {
      overviewMapSubReport: overviewMapOut
    }
```

Notes:

The `createOverviewMap` processor requires a map and an overview-map attribute. Like the `createMap` processor it creates a sub-report, that can be embedded as such in the main template.

Legend processor


Arbres
CLPA - Interprétation de phénomène
Avalanche localisée
Avalanche localisée présumée
Liaison présumée entre avalanches
Peturbations
CLPA : Zones sans enquête terrain
Points de vente
Sites du CdL

- Adds a legend to a report.
- The data for the legend needs to be specified in the request data.
- Needs a jrxml template.

Documentation: prepareLegend

Legend processor

Example configuration:

```
attributes:
  ...
  legend: !legend {}

processors:
  ...
  # create a datasource for the report with the 'legend' attribute
  - !prepareLegend
    maxWidth: 148
    dpi: 100
    # template that needs to be available
    template: legend.jrxml
```

Notes:

The `prepareLegend` processor downloads the legend graphics that are given in the request data and creates a datasource that can be used to fill a sub-report.

In the main template two parameters for the datasource and for the path to the compiled template have to be defined:

```
<parameter name="legend" class="net.sf.jasperreports.engine.data.JRTa
<parameter name="legendSubReport" class="java.lang.String"/>
```

Then a sub-report element for the legend can be specified:

```
<subreport>
  <reportElement x="0" y="390" width="148" height="42" uuid="..."/>
  <dataSourceExpression><![CDATA[$P{legend}]]></dataSourceExpression>
  <subreportExpression><![CDATA[$P{legendSubReport}]]></subreportExpr
</subreport>
```

The sub-report template `legend.jrxml` should have fields for the name and icon of a legend entry.

```
<jasperReport ...>
  <field name="name" class="java.lang.String"/>
  <field name="icon" class="java.awt.Image"/>
  <field name="level" class="java.lang.Integer"/>
  <detail>
    <band height="15" splitType="Prevent">
      <printWhenExpression><![CDATA[!$F{name}.equals("")]]>
```

```
        </printWhenExpression>
        <textField isStretchWithOverflow="true">
          <reportElement x="0" y="0" width="185" height="13"
            uuid="..."/>
          <textFieldExpression><![CDATA[$F{name}]]>
          </textFieldExpression>
        </textField>
      </band>
      <band height="14" splitType="Prevent">
        <printWhenExpression><![CDATA[$F{icon} != null]]>
        </printWhenExpression>
        <image scaleImage="RealHeight">
          <reportElement stretchType="RelativeToTallestObject"
            x="0" y="0" width="185" height="13" uuid="..."/>
          <imageExpression><![CDATA[$F{icon}]]>
          </imageExpression>
        </image>
      </band>
    </detail>
</jasperReport>
```

Full examples are available here:

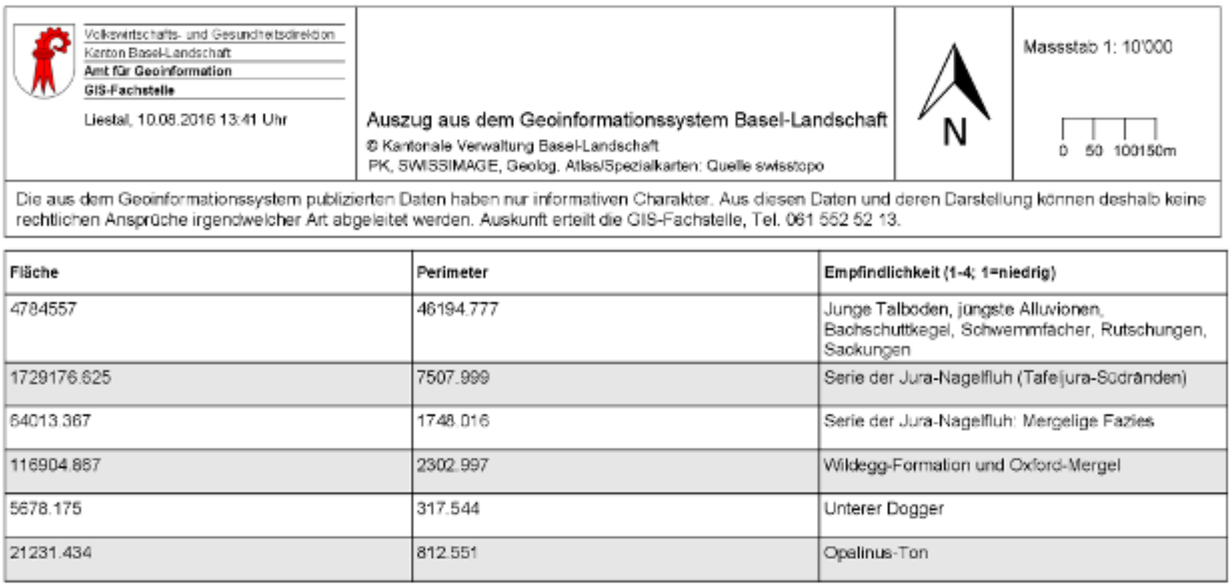
examples/verboseExample

examples/legend_cropped

Legend example request

```
"legend": {
  "name": "",
  "classes": [{
    "name": "Trees",
    "icons": ["http://.../trees.png"]
  }, {
    "name": "Parking",
    "icons": ["http://.../parking.png"]
  }]
},
```

Table processor



Volkswirtschafts- und Gesundheitsdirektion
Kanton Basel-Landschaft
Amt für Geoinformation
GIS-Fachstelle

Liestal, 10.08.2016 13:41 Uhr

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

N
Massstab 1: 10'000
0 50 100150m

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 52 13.

Fläche	**Perimeter**	**Empfindlichkeit (1-4; 1=niedrig)**
4784557	46194.777	Junge Talboden, jüngste Alluvionen, Bachschuttkegel, Schwemmfacher, Rutschungen, Sackungen
1729176.625	7507.999	Serie der Jura-Nagelfluh (Tafeljura-Südränden)
64013.367	1748.016	Serie der Jura-Nagelfluh: Mergelige Fazies
116904.867	2302.997	Wildegg-Formation und Oxford-Mergel
5678.175	317.544	Unterer Dogger
21231.434	812.551	Opalinus-Ton

- Adds a table to the report
- Can be used to show information for features on the map.
- The data shown in the table is defined in the request data.

Table processor

Example configuration:

```
attributes:
  ...
  table: !table {}

processors:
    ...
  - !prepareTable
      dynamic: true
      outputMapper: {
          table: tableDataSource
      }
```

Notes:

The table processor as configured above generates a sub-report and a datasource for the table. The table in this sub-report uses a default design. To adapt the design, a custom template can be provided (attribute `jasperTemplate`, see also in the documentation).

Table processor: Template

```
<parameter name="tableSubReport" class="java.lang.String"/>
<parameter name="tableDataSource"
    class="net.sf.jasperreports.engine.JRDataSource"/>
...
<subreport>
  <reportElement stretchType="RelativeToTallestObject"
      x="0" y="177" width="555" height="42" uuid="...">
    <property name="local_mesure_unitwidth" value="pixel"/>
    <property name="com.jaspersoft.studio.unit.width" value="px"/>
  </reportElement>
  <dataSourceExpression>
    <![CDATA[$P{tableDataSource}]]>
  </dataSourceExpression>
  <subreportExpression>
    <![CDATA[$P{tableSubReport}]]>
  </subreportExpression>
</subreport>
```

Table request example

The data contained in the table is defined in the request:

```
"attributes": {
  ...
  "table": {
    "columns": ["col1", "col2", ...],
    "data": [
      ["data_col1", "data_col2", ...],
      [ ...],
      ...
    ]
 },
  ...
}
```

INTEGRATION IN AN UI

Agenda of this chapter

Goal: Learn how MFP can be integrated in an UI.

- Web API
- JavaScript libraries

Web API

- Getting information about available print configurations
- Creating a print job
- Getting the status of a print job
- Canceling a print job
- Downloading a generated report

Documentation: Web API

Notes:

The web API of the MapFish Print web-application allows to integrate MFP in other applicatio
(client- or server-side).

List available print configurations

`GET /print/apps.json`

```
[
  "simple",
  "default",
  ...
]
```

Example request: http://localhost:8080/print-servlet-3.6/print/apps.json

Capabilities of a print configuration

`GET /print/:appId/capabilities.json`

```
{
  "app": "simple",
  "layouts": [{
    "name": "A4 portrait",
    "attributes": [{
      "name": "title",
      "type": "String",
      "default": "Countries"
    }, {
      "name": "map",
      "type": "MapAttributeValues",
      "clientParams": {
        "center": {
          "type": "double",
          "isArray": true
        },
        ...
      },
      "clientInfo": {
        "height": 330,
        "width": 780,
        "dpiSuggestions": [72, 120, 200, 254, 300],
        "maxDPI": 400
      }
    }]
  }],
  "formats": ["bmp", "gif", "pdf", "png", "tif", "tiff"]
}
```

Example request: http://localhost:8080/print-servlet-3.6/print/simple/capabilities.json

Creating a print job

`POST /print/:appId/report.:format`

Example request body:

```
{
  "layout": "A4 portrait",
  "outputFormat": "pdf",
  "attributes": {
    "map": {
...
```

Example request: http://localhost:8080/print-servlet-3.6/print/simple/report.pdf

Example response:

```
{
  "ref": "4d5e2be9",
  "statusURL": "/print-servlet-3.6/print/status/4d5e2be9.json",
  "downloadURL": "/print-servlet-3.6/print/report/4d5e2be9"
}
```

Getting the status of a print job

`GET /print/status/:ref.json`

Example request: http://localhost:8080/print-servlet-3.6/print/status/4d5e2be9.json

Example response:

```
{
  "done": false,
  "status": "running",
  "elapsedTime": 513,
  "waitingTime": 0,
  "downloadURL": "/print-servlet-3.6/print/report/4d5e2be9"
}
```

Documentation: Status

Canceling a print job

`DELETE /print/cancel/:ref`

Example request: http://localhost:8080/print-servlet-3.6/print/cancel/4d5e2be9

Getting a generated report

`GET /print/report/:ref`

Example request: http://localhost:8080/print-servlet-3.6/print/report/4d5e2be9

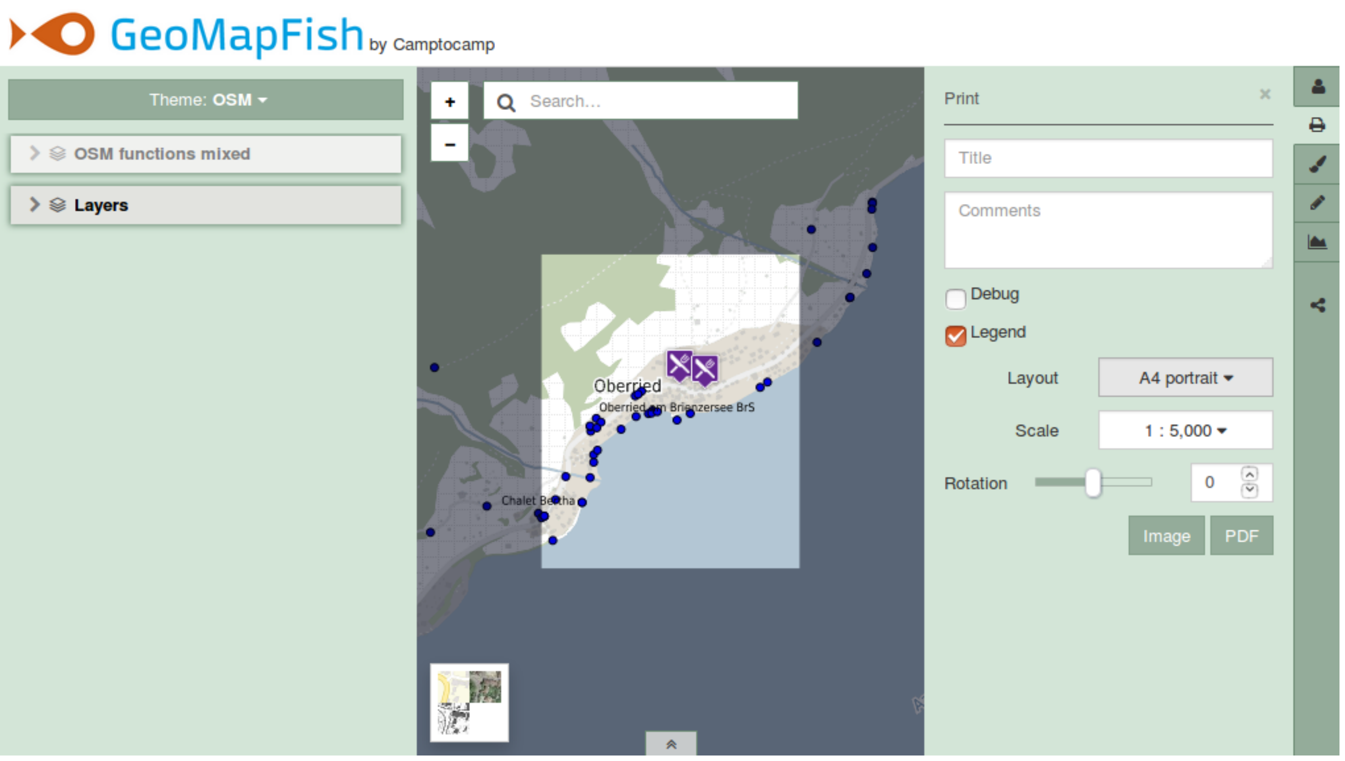
Create and get report (in one request)

```
POST /print/:appId/buildreport.:format
```

Example request: http://localhost:8080/print-servlet-3.6/print/simple/buildreport.pdf

ngeo

- JavaScript library for developing OpenLayers 3 applications with Angular
- Support for MapFish Print 3
  - Generates a print request from an OL 3 map (converts layers and styles)
  - Generates UI from print configuration capabilities



Title
Comments
Debug
✓ Legend
Layout A4 portrait
Scale 1 : 5,000
Rotation 0

Notes:

ngeo is a JavaScript library that aims to ease the development of applications with OpenLayers 3 and Angular. ngeo also provides an Angular service for MapFish Print 3, that generates a print request from a given OpenLayers 3 map. The service converts the layers and styles defined in the OpenLayers map to the format that is understood by MapFish Print 3.

Links:

ngeo

Example application with ngeo

Example with MFP service

Support in GeoExt 3 (ExtJS 6)

GeoExt 3 example

GeoExt 3

DATASOURCE PROCESSOR

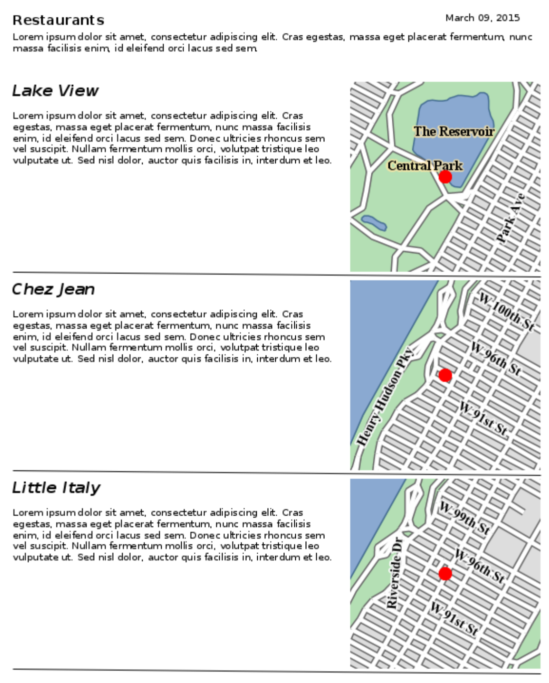
Agenda of this chapter

Goal: Get to know the datasource processor.

- Concept of datasources with JasperReports
- The datasource processor
- Templates for the datasource processor

Idea

An arbitrary number of maps, tables, ... in a report.



Restaurants

March 09, 2015

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras egestas, massa eget placerat fermentum, nunc massa facilisis enim, id eleifend orci lacus sed sem.

Lake View

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras egestas, massa eget placerat fermentum, nunc massa facilisis enim, id eleifend orci lacus sed sem. Donec ultricies rhoncus sem vel suscipit. Nullam fermentum mollis orci, volutpat tristique leo vulputate ut. Sed nisl dolor, auctor quis facilisis in, interdum et leo.

Chez Jean

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras egestas, massa eget placerat fermentum, nunc massa facilisis enim, id eleifend orci lacus sed sem. Donec ultricies rhoncus sem vel suscipit. Nullam fermentum mollis orci, volutpat tristique leo vulputate ut. Sed nisl dolor, auctor quis facilisis in, interdum et leo.

Little Italy

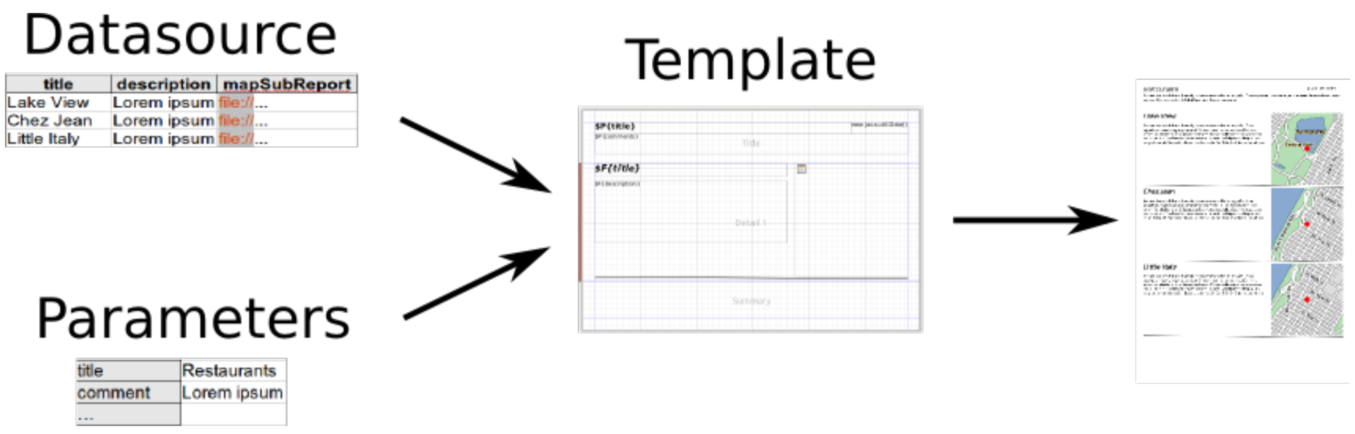
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras egestas, massa eget placerat fermentum, nunc massa facilisis enim, id eleifend orci lacus sed sem. Donec ultricies rhoncus sem vel suscipit. Nullam fermentum mollis orci, volutpat tristique leo vulputate ut. Sed nisl dolor, auctor quis facilisis in, interdum et leo.

Notes:

So far the reports in this workshop were relatively static: If one map was defined in the print configuration, one map was included in the report. If two maps were needed, two maps had to be defined in the configuration. Sometimes it is required that reports are more flexible. Fo example if the number of maps or tables that should be shown in a report is not known in advance.

In this chapter we will explore how the `createDataSource` processor allows to do that.

Concept of datasources



Notes:

Datasources are data containers that contain tabular data. A datasource can for example be a database or a file. The JasperReports template engine uses the data from a datasource to fill a report. Each row in the datasource repeats the `detail` band in a report template.

Besides databases and files, a datasource can also be created by MapFish Print with data provided in the print request. A datasource can not only contain simple values like numbers or strings, a map can also be rendered for each row in the datasource. This is done with the `createDataSource` processor.

The datasource processor

```
templates:
    A4 portrait: !template
        reportTemplate: A4_Portrait.jrxml
        tableData: jrDataSource
        attributes:
            title: !string {}
            comments: !string {}
            datasource: !datasource
                attributes:
                    title: !string {}
                    description: !string {}
                    map: !map
                        maxDpi: 254
                        width: 200
                        height: 200
        processors:
        - !reportBuilder
            directory: '.'
        - !createDataSource
            processors:
            - !createMap {}
```

Notes:

The createDataSource processor receives data from the print request, (optionally) runs processors on each row of the data, and then creates a datasource which is used by JasperReports to fill the report template.

To use the `createDataSource` processor an attribute of type `datasource` has to be definec The datasource attribute describes the fields of the datasource. The property `attributes` is similar to the `attributes` property of the template. Attributes with simple types like numbers or strings, but also complex types like maps can be listed here.

In the processor section, the `createDataSource` processor has to be defined. By setting the `processors` property, processors that should be run on each row of the datasource can be specified. In the above example a map is created for each entry.

To make the datasource available to the template, the property `tableData` has to be set. Th attribute `jrDataSource` is created by the `createDataSource` processor and is assigned to this property:

```
tableData: jrDataSource
```

Alternatively, the `createDataSource` can also use the datasource with a provided sub-report. This might be useful if a second datasource is used for the main template.

Datasource: Example request

```
{
  "layout": "A4 portrait",
  "outputFormat": "pdf",
  "attributes": {
    "title": "Restaurants",
    "comments": "...",
    "datasource": [{
      "title": "Lake View",
      "description": "...",
      "map": {
        "center": [-8233518, 4980320],
        "scale": 25000,
        ...
      }
    }, {
      "title": "Chez Jean",
      "description": "...",
      "map": {
        "center": [-8234918, 4981920],
        "scale": 25000,
        ...
      }
    }, ...]
  }
}
```

Notes:

In the print request, the `datasource` attribute is a list of entries. Each entry contains the attributes that were defined in the configuration.

Template: Fields

```
<field name="title" class="java.lang.String"/>
<field name="description" class="java.lang.String"/>
<field name="mapSubReport" class="java.lang.String"/>
```

Notes:

In the JasperReports template, the attributes defined on the `datasource` attribute have to b listed as report fields (not as parameters). The field `mapSubReport` is for the output generated by the map processor.

Template: Detail band

```
<detail>
  <band height="209" splitType="Stretch">
    <textField>
      ...
      <textFieldExpression><![CDATA[$F{title}]]></textFieldExpression>
    </textField>
    <subreport>
      <reportElement ...>
        ...
      </reportElement>
      <subreportExpression><![CDATA[$F{mapSubReport}]]></subreportExpress
    </subreport>
    <line>
      <reportElement x="0" y="200" width="555" height="5" uuid="..."/>
    </line>
    <textField>
      ...
      <textFieldExpression><![CDATA[$F{description}]]></textFieldExpressi
    </textField>
  </band>
</detail>
```

Notes:

The defined fields can be used in the `detail` band of a template. The `detail` band is repeated for every entry of the datasource.

Resources

Documentation: createDataSource processor

Documentation: datasource attribute

Examples: multiple maps

Exercise 8.1: Datasources

Continue the previous exercise. Create a datasource with a title, some info, and the map.

/!\ The first band should be transformed as a title band, the map should be in the details band.

Solution 8.2: Datasources

Solution: https://geomapfish-demo.camptocamp.net/bl_datasource.zip

Diff: https://github.com/sbrunner/mapfish-print/commit/1da4c0f218960b23c7b81064b083ddfc8f966a8d

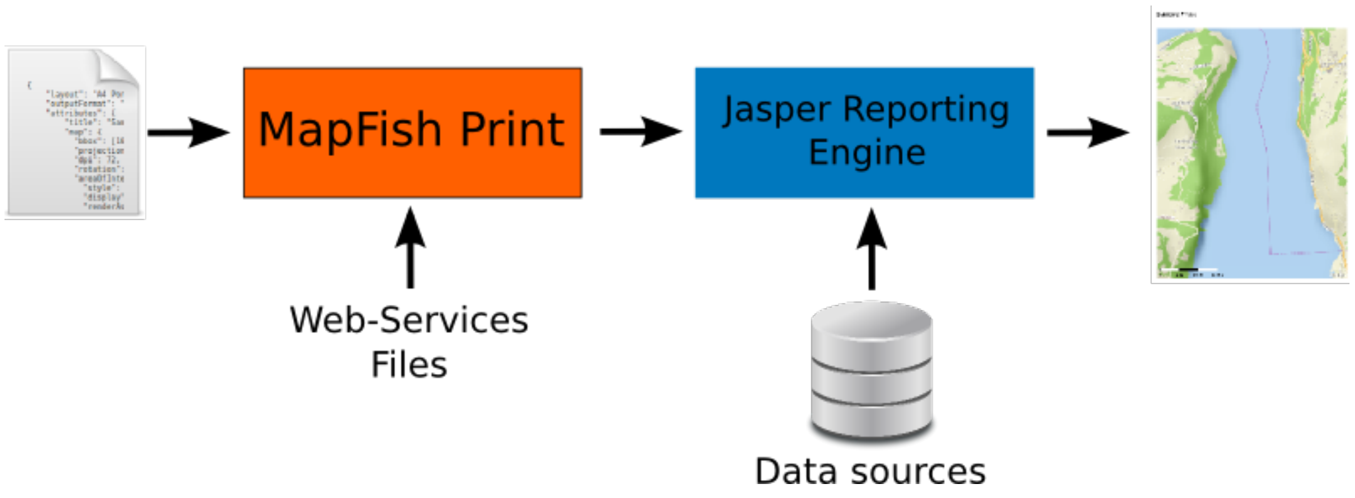
EXTERNAL DATASOURCES

Agenda of this chapter

Goal: Learn how to integrate external datasources in a report.

- Idea
- Connecting to a database from a report

Idea



Notes:

As we have seen in the previous chapter, MapFish Print can create a datasource, that the report is built with. But it is also possible to directly configure a datasource within the report template.

In that case, Mapfish Print for example renders a map. And additional data is loaded from a database. This has the advantage that the data has not to be sent in the print request.

Complex reports are possible with this technique. Single or multiple entries can be loaded from the database. Additional data can be requested in separate queries and be shown in sub-reports. Charts can be created with this data, ...

The following slides will walk through the basic configuration that is required to set up external datasources.

Configuring the database connection

```
templates:
    A4 portrait: !template
        reportTemplate: A4_Portrait.jrxml
        jdbcUser: "www-data"
        jdbcPassword: "www-data"
        jdbcUrl: "jdbc:postgresql://localhost:5432/mydb"
        attributes:
            id: !integer {}
```

Notes:

The database connection parameters are configured in the YAML configuration. With this configuration, MapFish Print will generate a datasource, and start the report generation with the datasource.

When generating the report, a JDBC driver for the database that is used must be available on the Java class-path. When using the MFP CLI, copy the driver JAR file in `print-cli/lib`. If you are using the MFP web-application, you can either copy the file in the Tomcat `lib` folder (e.g. `apache-tomcat-8.0.36/lib`) or package the JAR together with the MFP WAR (`WEB-INF/lib`).

Datasource query

```
<parameter name="id" class="java.lang.Integer"/>
<parameter name="mapSubReport" class="java.lang.String"/>

<queryString>
    <![CDATA[
        select id, name, address, phone, hours
        from restaurants
        where id = $P{id}
    ]]>
</queryString>

<field name="id" class="java.lang.Integer"/>
<field name="name" class="java.lang.String"/>
<field name="address" class="java.lang.String"/>
<field name="phone" class="java.lang.String"/>
<field name="hours" class="java.lang.String"/>
```

Notes:

Inside the template file, a query that loads the data for a restaurant is defined. The parameter `$P{id}` is received from the print request, and is used in the where clause of the query.

This query selects a single row, but a query could also return multiple results.

The full example is available in `configurations/07-external-datasources`.

Resources

TIBCO Jaspersoft Studio User Guide: Data Sources

SUPPLEMENTAL EXERICES

Exercise 10.1: Conditional

Add a conditional text element with different text and color.

Tips: Looks for the "Print When Expression" field in an element of Jasper Report (in "Appearance).

Solution 10.2: Datasources

Solution: https://geomapfish-demo.camptocamp.net/bl_conditional.zip

Diff: https://github.com/sbrunner/mapfish-print/commit/b2289f3223aa70e86b50310f9d8d207f555b8f53

Exercise 10.3: More on layers

Specify the layer in the YAML file.

- **Move the layer definition to the config file.**

Solution 10.4: More on layers

Solution: https://geomapfish-demo.camptocamp.net/bl_layers.zip

Diff: https://github.com/sbrunner/mapfish-print/commit/3c61c605bc8e4139630c6f05ea195d32bc815a1e

Exercise 10.5: Dynamic table

Add a dynamic table in the report.

Solution 10.6: Dynamic table

Solution: https://geomapfish-demo.camptocamp.net/bl_table.zip

Diff: https://github.com/sbrunner/mapfish-print/commit/8bd02dd33eef13ea0f1e3a652a6f1372ac2c45f0

Exercise 10.7: Vector layer

Use vector layer in the report configuration.

Solution 10.8: Vector layer

Solution: https://geomapfish-demo.camptocamp.net/bl_vector.zip

Diff: https://github.com/sbrunner/mapfish-print/commit/0022889d14e852344e82c5553c2b0a5297e46b2e

Workshop Summary

During this workshop, we:

- Learned the structure of the configuration file.
- Designed templates with JasperSoft Studio.
- Used components like maps or north-arrows.
- Rendered different geo-data.
- Saw how to build more complex reports with datasources.

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.