Reporting with JasperReports

David THIBAU – 2021

david.thibau@gmail.com

Agenda

- Introduction
  - Business Intelligence
  - OpenSource model of Jaspersoft Suite
- *JasperReport* library
  - Generation model
  - The JRXML file
  - Distribution and Examples
- *JasperStudio :* Getting started
  - Introduction / Installation
  - Datasources
  - Dimension and structure of a report
  - Basic elements
  - Parameters, expression, variables
- Advanced elements
  - Groups
  - Subreports
  - Datasets, List and tables
  - Charts
  - Crosstab
- Java and JasperReport
  - JasperReport's API
  - Scriptlets
  - Build Tools
- Annex
  - Data Adapters
  - Template and wizards
  - Report Books
  - Spring Integration
  - Design API

Introduction

Business Intelligence
Opensource model, TIBCO JasperSoft

Business Intelligence

- **Business Intelligence (BI)**: Technologies and applications that allow companies to better understand their activity
  => provide historical, current or forecast views of the activity of the company
- The common functions of the BI are:
  - Data mining
  - Reporting
  - OLAP (Online Analytic Processing)
  - Data analysis (Aggregation)
  - Forecast analysis
  - BigData, Machine Learning
- The objective is the decision support.

Data warehouses

- BI applications handle large volumes of data : **data warehouses**

- They can be built with **ETL** (*Extract Transform Load* ou *datapumping*) tools which extract and transform data from multiple heterogeneous sources
  - In lot of real case, reporting activities use simply **relational production databases**.

OpenSource model of JasperSoft suite

TIBCO

TIBCO bought the JasperReport project in order to offer a complete BI Commercial suite: The ***JasperSoft suite***

2 distributions are available :

- **Community edition** : Completely free, it takes advantage of the community to contribute and test new features
- **Enterprise edition** : Offers more tested versions, provides access to TIBCO support and legal commitment (certification), provides additional features

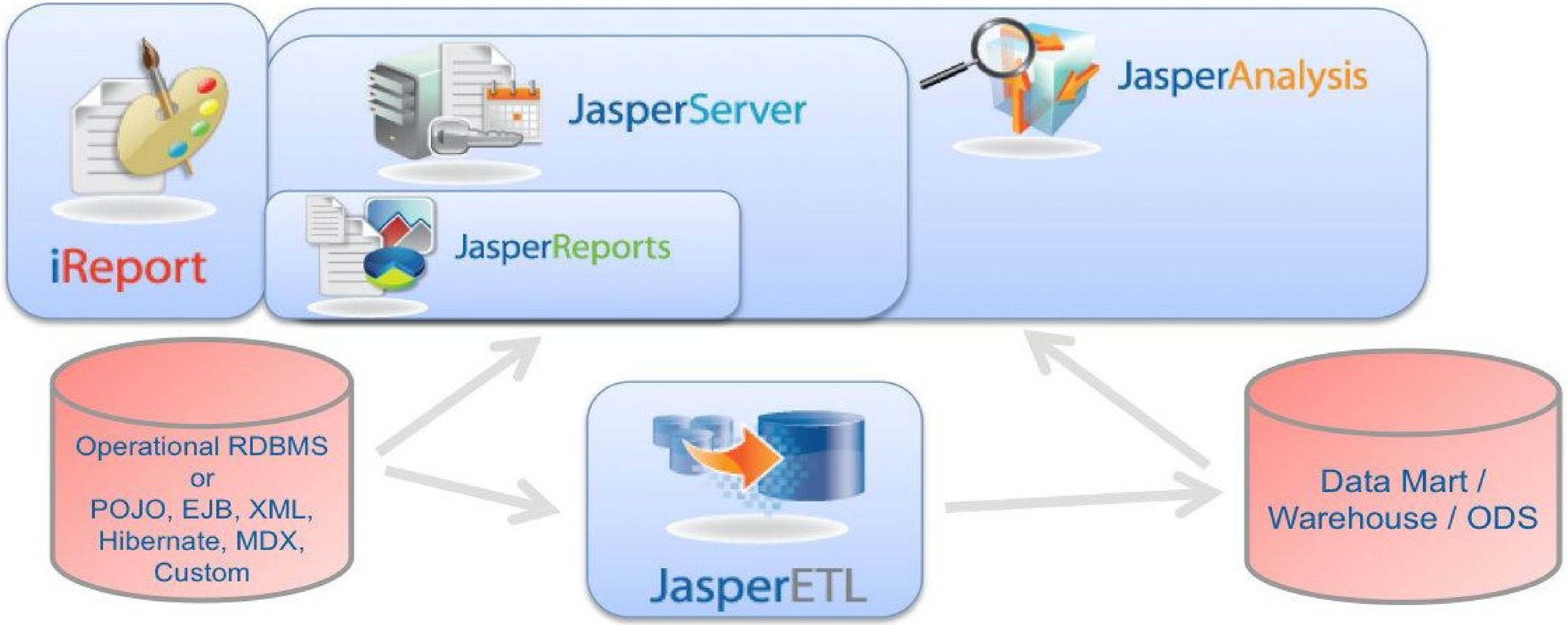
Objectives of *Jaspersoft* suite

- TIBCO's objectives
  - A complete BI suite ready to use
  - Easily integrated into business applications
  - Functional but affordable in terms of complexity
  - Competitive in terms of deployment cost
  - Interoperable with existing systems
  - Viralization via an OpenSource mode
  - Support for mobile devices, JavaScript technologies

Components of the JasperSoft suite

- **JasperReport** :
  Java library at the core of the suite, generates reports in different formats from an XML description.
- **IReport / JasperStudio** :
  « WYSIWYG » graphical tool to design reports, use by Java developers AND business experts
- **JasperServer** :
  Publish reports to end users.
  Administrative tasks : Report deployment, Access rights, Scheduling of generation tasks, Delivering reports by email, ... REST interface
- **JasperAnalysis** :
  Interactive OLAP data analysis application for business professionals.
- **JasperETL** :
  Talend-based data integration tool

Jaspersoft suite


iReport
JasperServer
JasperReports
JasperAnalysis
Operational RDBMS or POJO, EJB, XML, Hibernate, MDX, Custom
JasperETL
Data Mart / Warehouse / ODS

Licences

- 3 kinds of licences :
  - **JasperSoft OEM** intended for resellers or integrators of JasperSoft products
  - **Commerciale** : Intended for end users of JasperSoft products, in particular *JasperStudio*, *JasperServer* and *JasperAnalysis*
  - **OpenSource Licence**
    - **GPL** : any derivative work adopt the same licence and must be free
    - **LGPL** : application containing the JasperSoft product can commercialized

Comparative

	Community	Entreprise / Cloud
JasperReport	LGPL	Identical
JasperStudio	GPL	More graphical components
JasperServer	GPL	Several data sources, Cache, Cluster, BigData,
JasperAnalysis	GPL	Some Add-ons

JasperReport

Generation model
The JRXML file
Distribution and examples

Introduction

- Full Java, available as a jar, it can be integrated in any Java applications :
  - Application desktop
  - Application web
  - Batch
- The library generates reports from a design in XML format and from a data source (relational database or other)
- The report may be directed to :
  - Screen
  - Printer
  - Files (PDF, HTML, XLS, RTF, ODT, CSV, TXT et XML)

Generation model

The report is generated in 4 steps:

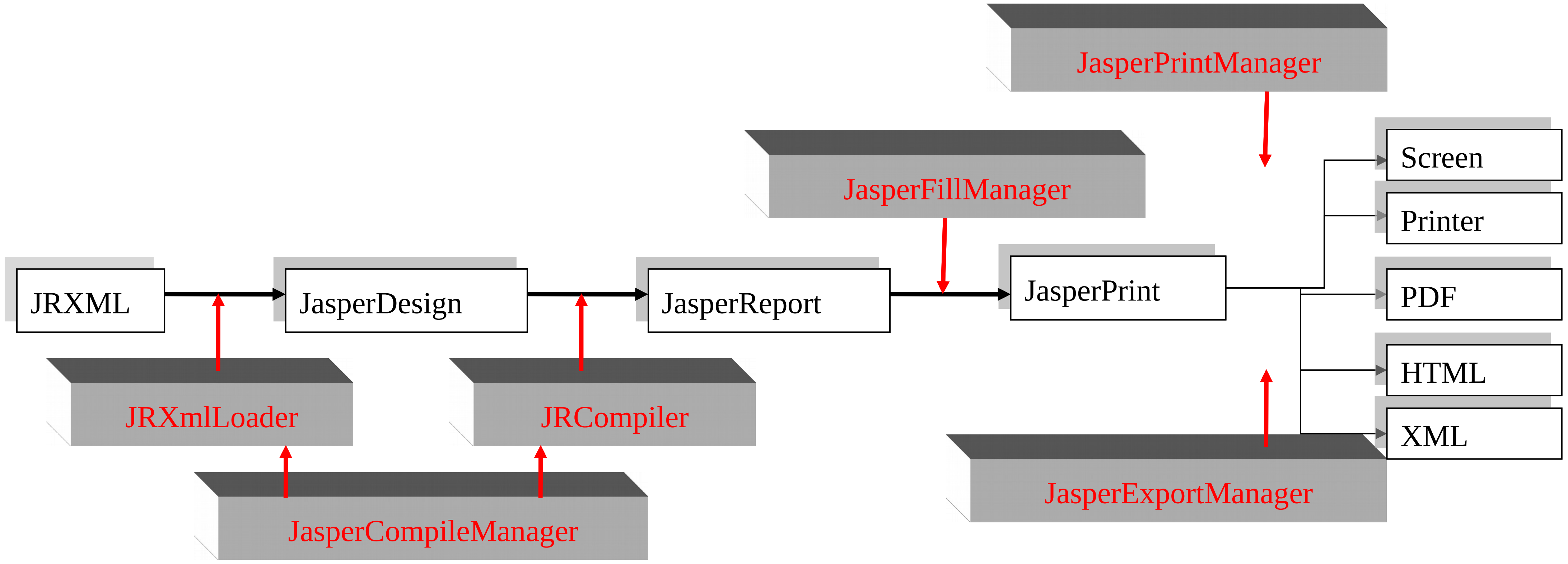
1. Design the **JRXML** file which defines :
   - The report's structure
   - Its layout and visual rendering
   - Dynamic data from a data source (SQL Query, method invocation, ...).

2. **Compile** the report, generate a binary file *.jasper*

3. **Fill data :** Produce the report merged with dynamic data. The result is a binary file in a proprietary format of JasperReport *.jrprint*

4. **Export** the print in another format (PDF, HTML, ...)

Report's Lifecycle



Usage

1) Developers design a report in their environment using *JasperStudio*

2) The compiled report is deployed in *JasperServer* or in a business application integrating the *JasperReport* library

3) Reports are generated dynamically by end users or batch processes

JasperReport

Generation model

Distribution and examples

The JRXML file

Installation

- Prerequisites :
  - Java
  - *ant/ivy* allowing to run the examples and generate the documentation (use of *hsqldb*)
- Download archive (*http://sourceforge.net/projects/jasperreports/*)
- Use binary or build from the sources via *ant* scripts

Distribution

- The archive contains sources and build files (*Ant/Ivy* et *Maven*). The build generates documentation, execute test (examples) and build the binary
  - ***src*** : Sources
  - ***build*** : Result of build
  - ***demo*** : Examples
  - ***doc*** : Documentation (configuration properties and xml schema)
  - ***lib*** : Third-party librairies
  - ***dist*** : Produced jars

JasperReport’s configuration

The generation engine reads its configuration from ***jasperreport.properties***

- Many options are available
- Reference documentation is generated in *dist/docs/config-reference.html*

JasperReport

Generation model
Distribution and examples
The JRXML file

Basics of XML

- Well-formed :
  - Only one root element : (*<jasperReport>)*
  - No overlapped tags
  - Attributes with quotes
- Valid against a schema which describes the correct sequences of elements:
  - Version 3.6 : *http://jasperreports.sourceforge.net/xsd/jasperreport.xsd*
- Documentation : *dist/docs/schema.reference.html*

File’s structure

Elements inside the root element are :

- **Declaratives** elements (Style, Variables, Parameters, ...)
- **Structural** elements which defines horizontal bands with a specified height
- Inside the bands, **display elemnts** from the available palette

Declarative elements

- ***property**** : N arbitrary properties global to the report
- ***import**** : N import of java classes/packages
- ***template**** : N references to a report template
- ***reportFont**** : N police definition used in the report
- ***style**** : N styles definitions
- ***subDataset**** : N sub-datasets (Alternatives to the main dataset)
- ***scriptlet**** : N scriptlets (Hooks Java)
- ***parameter**** : N parameters (data from outside)
- ***queryString?*** : Only one main query (SQL or other)
- ***field**** : N fields definition
- ***sortField**** : N sort fields (Java sort)
- ***variable**** : N variables definitions (computation)
- ***filterExpression?*** : 1 Java expression to filter data
- ***group**** : N groups (used to group data and calculate agregation)

Structural elements

- ***background?*** : 1 background printed on each page
- ***title?*** : 1 title, printed once at the beginning of the report
- ***pageHeader?*** : 1 page header printed on each page
- ***columnHeader?*** : 1 column header printed on each column (for multi-columns report)
- ***detail?*** : 1 detail. Printed for each row of the main dataset
- ***columnFooter?*** : 1 column footer (for multi-columns report)
- ***pageFooter?*** : 1 page footer printed on each page
- ***lastPageFooter?*** : The footer of the last page
- ***summary?*** : 1 summary printed once at the end of the report
- ***noData?*** : Alternative text which may be printed when there are no data

Example

```
<?xml version="1.0" encoding="UTF-8"?>
    <jasperReport
     xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports
     http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
     name="report2" pageWidth="612" pageHeight="792" leftMargin="20" rightMargin="20" topMargin="20" bottomMargin="20">
            <background>  <band splitType="Stretch"/>  </background>
            <title>
                   <band height="79" splitType="Stretch">
                            <staticText> <reportElement x="236" y="29" width="100" height="20"/>
                                    <text><![CDATA[Mon titre centré]]></text>
                            </staticText>
                    </band>
            </title>
            <pageHeader> <band height="35" splitType="Stretch"/> </pageHeader>
            <detail> <band height="125" splitType="Stretch"/> </detail>
            <pageFooter>
                    <band height="54" splitType="Stretch">
                       <textField pattern="EEEEE dd MMMMM yyyy"> <reportElement x="472" y="34" width="100" height="20"/>
                          <textFieldExpression class="java.util.Date">
                             <![CDATA[new  java.util.Date()]]></textFieldExpression>
                          </textField>
                          <image> <reportElement x="0" y="46" width="78" height="8"/>
                             <imageExpression class="java.lang.String">
                              <![CDATA["/home/Jasper/images/logo-plb-footer.gif"]]></imageExpression>
                            </image>
                    </band>
            </pageFooter>
    </jasperReport>
```

Jasper Studio Getting started

Presentation and installation
Data adapters
Fields
Report wizard
Dimensions and band
Simple elements
Parameters, expressions, variables

Introduction

- *jasperStudio* is a **WYSWIG** design tool that offers a complete environment for creating, testing, previewing or exporting *JasperReport* reports (Based on Eclipse)
- Supported Data sources: databases, OLAP connections, XML or CSV files, JavaBeans collections and even specific sources
- Supported query languages : SQL, HQL, xPath, EJBQL and MDX and even allows to plug in other languages if necessary (PL / SQL).
- For developers, jasperStudio can generate the *.jasper* files which can then be deployed in java applications
- *jasperStudio* can also be used to deploy reports directly into ***JasperServer***.

Main features

- Full compatibility with JasperReports XML tags… if the correct versions are used
- « *WYSIWYG* » editor
- Syntax coloring for expressions (Java, Javascript, Groovy)
- Support for internationalization
- Report structure browser: Outline
- Built-in compilation, fill data, export and preview tools
- Wizards to create reports, subreports, crosstabs, charts
- Support for report templates, Style Libraries.
- Library of standard objects (for example the number of pages)
- Drag-and-drop functionality, unlimited Undo / redo
- Explorer of a JasperServer repository.
- Support of all databases accessible by JDBC, extensible to other types of data sources
- SQL, MDX, XPath query debugging assistant

Available documentation

- Website
  https://community.jaspersoft.com/documentation?version=49176
- Reference guide :
  « *JasperStudio User Guide* »

Installation

- Prerequisites :
  - Java 1.8+
  - 256MB RAM + 50MB disk

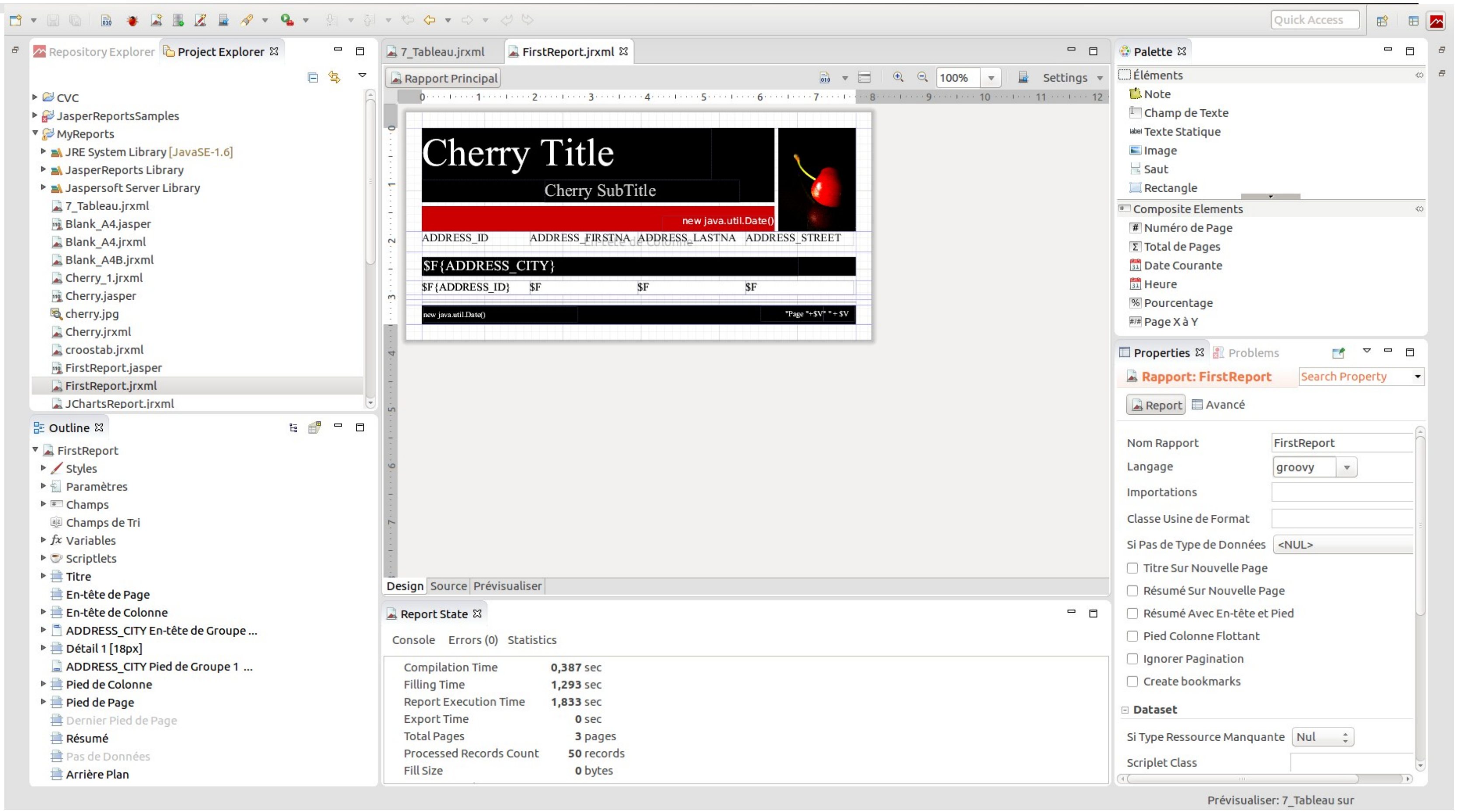
- Binary Distributions :
  - *TIBCOJaspersoftStudio-x.x.x.tgz* : Archive to unzip
  - *TIBCOJaspersoftStudio-x.x.x-windows-installer.exe* : Installer for Windows.

- Also available as an Eclipse plugin

Installation

- From the binary version, just unzip it, then run the executable: Jasper Studio
  - With some Ubuntu releases use the *./runubuntu.sh* script (Ubuntu)
- The Windows installer, adds an entry in "My programs" + a shortcut on the desktop

Presentation *JasperStudio*



```
Compilation Time          0,387 sec
Filling Time              1,293 sec
Report Execution Time     1,833 sec
Export Time                   0 sec
Total Pages                   3 pages
Processed Records Count      50 records
Fill Size                     0 bytes
```

Nom Rapport	FirstReport
Langage	groovy
Importations	
Classe Usine de Format	
Si Pas de Type de Données	<NUL>
☐ Titre Sur Nouvelle Page	
☐ Résumé Sur Nouvelle Page	
☐ Résumé Avec En-tête et Pied	
☐ Pied Colonne Flottant	
☐ Ignorer Pagination	
☐ Create bookmarks	
Dataset	
Si Type Ressource Manquante	Nul
Scriplet Class	

Multi-windowing

- Workbench is composed of several windows:
  - **Central design space** for editing reports, viewing XML and previewing
  - **Outline**: Hierarchical view of the report, allowing you to select its elements
  - **Properties view**: Allows you to edit the properties of the selected element
  - **Palette**: Place items on the report
  - **Formatting tools**: Align elements
  - **Problems**: Warnings, XML parsing problem, positioning problem
  - **Report state**: Result of internal programs (Compilation, data fusion, ..)
  - **JasperServer Repository Explorer**
- General configuration of the IDE : *Windows→ Preferences→ JasperSoft Studio*

Test configuration

- Test the configuration by creating an empty report :
  1. *File → New → Jasper Studio → JasperReports Project*
  2. *File → New → JasperReport*
  3. *Choose the template « blank »*
  4. *Choose a filename.*
- If everything is normal, jasperStudio generates a *.jasper* file and displays a preview of a blank page

Jasper Studio
Getting started

Presentation and installation
Data adapters
Fields
Report wizard
Dimensions and band
Simple elements
Parameters, expressions, variables

Introduction

- JasperReport supports different types of data sources or ***Data Adapter***
  (via the Java interface JRDataSource)

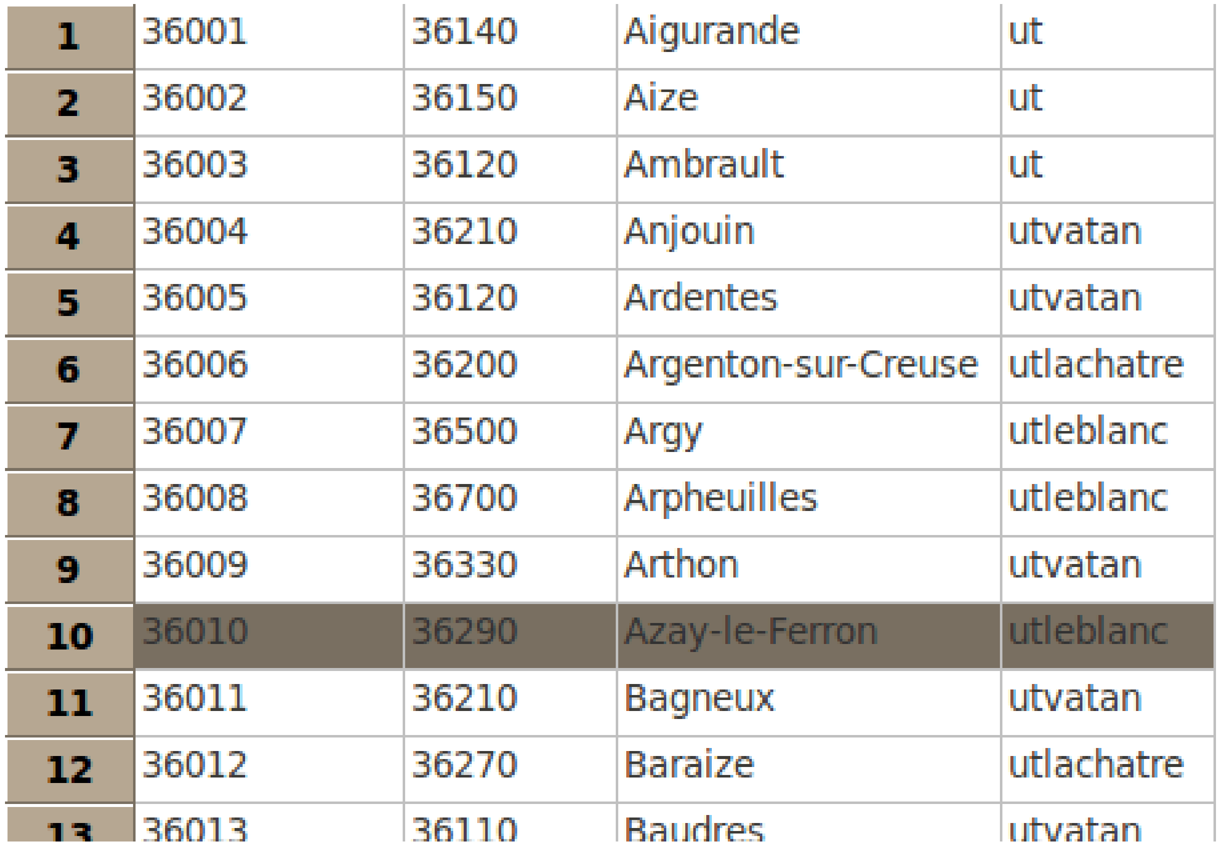
- The most used implementation is a relational database via JDBC
  (implementation JRResultSetDataSource which wraps a java.sql.ResultSet)

JRDataSource iteration

- A ***JRDataSource*** is simply an object able to iterate over a set of records: the rows or records.
- It is a consumable object: During generation, the lines are read one by one and it is not possible to go back

JRDataSource illustration

During generation, there is a single current record

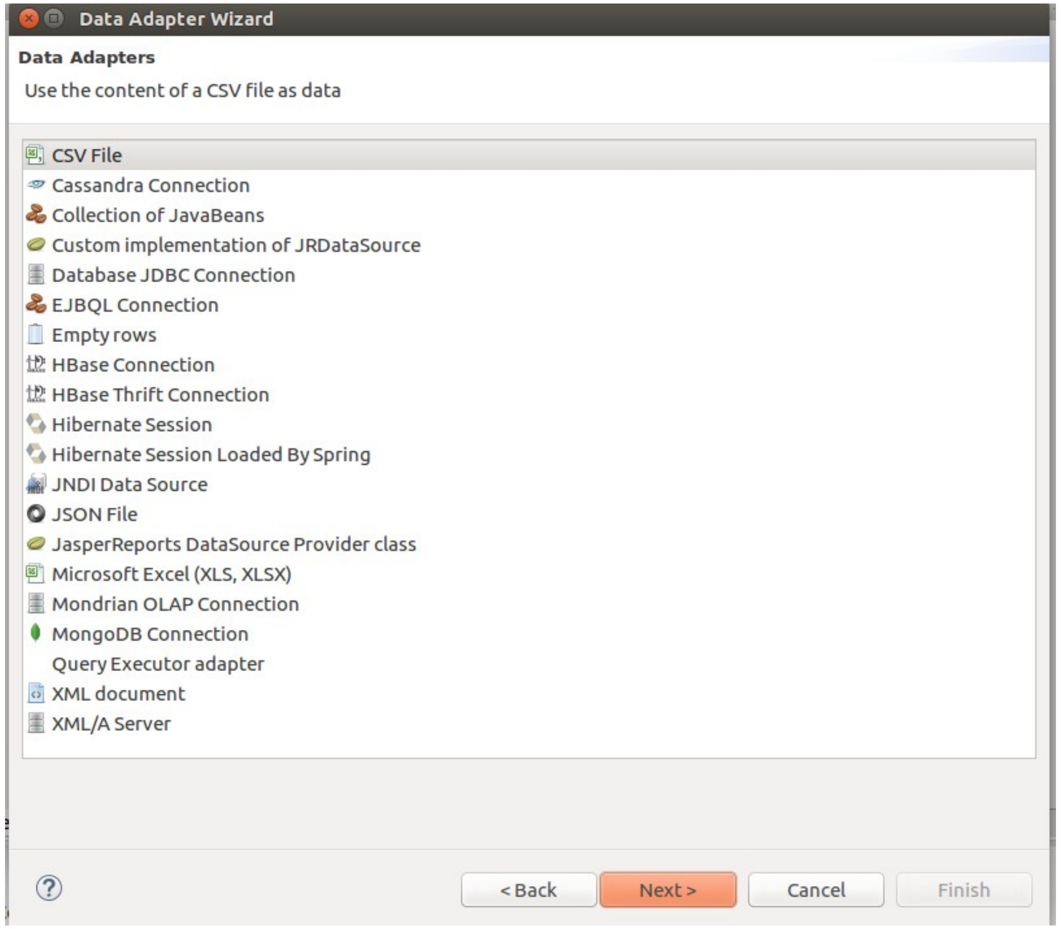


1	36001	36140	Aigurande	ut
2	36002	36150	Aize	ut
3	36003	36120	Ambrault	ut
4	36004	36210	Anjouin	utvatan
5	36005	36120	Ardentes	utvatan
6	36006	36200	Argenton-sur-Creuse	utlachatre
7	36007	36500	Argy	utleblanc
8	36008	36700	Arpheuilles	utleblanc
9	36009	36330	Arthon	utvatan
10	36010	36290	Azay-le-Ferron	utleblanc
11	36011	36210	Bagneux	utvatan
12	36012	36270	Baraize	utlachatre
13	36013	36110	Baudres	utvatan

Supported Data Adapters

- **JDBC connection** / **JNDI Data Source**: Connection to a relational database in Java
- **Java Code :** Collection of JavaBeans, Hibernate, EJB, Custom implementation of JRDataSource
- **Simple files :** XML, CSV, JSON, Excel
- **NOSQL** : Cassandra, MongoDB
- **Connexion OLAP Mondrian ou XMLA** : Connection used for data analysis
- **Empty data source** : The report has no dynamic data

File → New Data Adapter


Data Adapter Wizard

Data Adapters

Use the content of a CSV file as data

- CSV File
- Cassandra Connection
- Collection of JavaBeans
- Custom implementation of JRDataSource
- Database JDBC Connection
- EJBQL Connection
- Empty rows
- HBase Connection
- HBase Thrift Connection
- Hibernate Session
- Hibernate Session Loaded By Spring
- JNDI Data Source
- JSON File
- JasperReports DataSource Provider class
- Microsoft Excel (XLS, XLSX)
- Mondrian OLAP Connection
- MongoDB Connection
- Query Executor adapter
- XML document
- XML/A Server

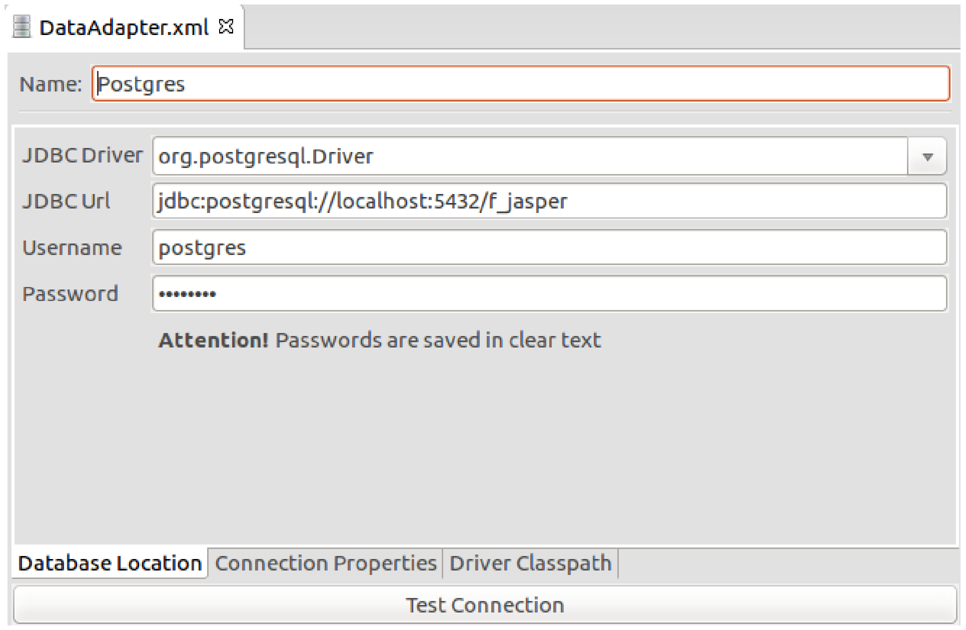
Reminders on JDBC

- A JDBC connection allows a Java program to connect to a relational database (all databases are supported)
- It is necessary to use the JDBC drivers (Java library) provided by the base editor.
- Configuring a JDBC connection with :
  - The main **driver** class
  - The **URL** for accessing the database (syntax dependent on the editor)
  - A valid database **account** (login / password)
- *jasperStudio* provides the JDBC drivers for HyperSonic. The others must be installed

Driver Installation

The JDBC driver library must be found by jasperStudio.

When creating a DataAdapter,the .jar containing the Driver is specified



Name: Postgres

JDBC Driver	org.postgresql.Driver
JDBC Url	jdbc:postgresql://localhost:5432/f_jasper
Username	postgres
Password	••••••••

Attention! Passwords are saved in clear text

Dissociation report and data source

- There is no strict association between a report and a data source.
- During generation, the data source is provided as a parameter
- It is thus possible to reuse the same design on different sources.
  - Example: Development database and production database

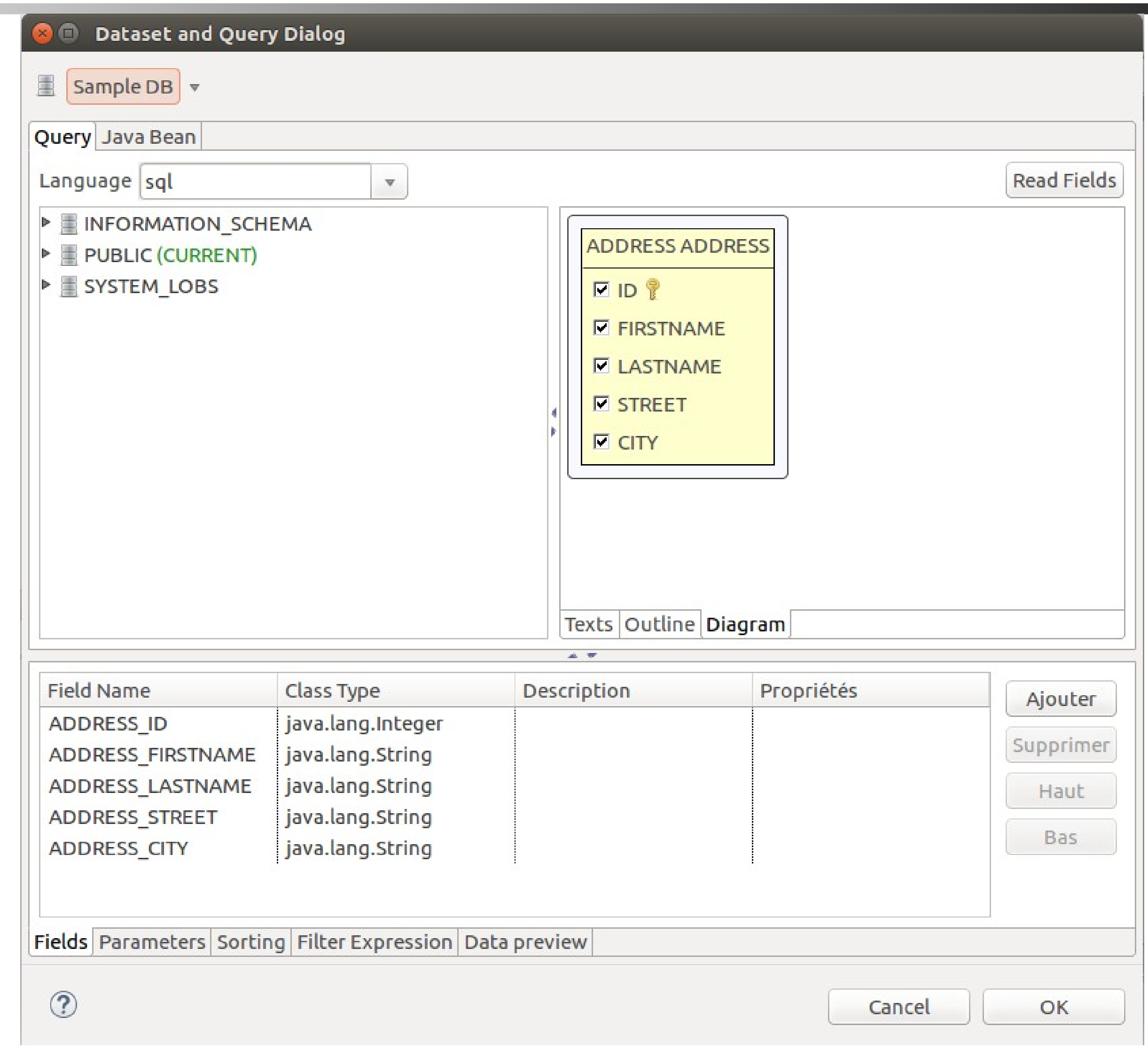
Common mistakes

- 3 types of errors can occur when configuring a JDBC connection:
  - **ClassNotFoundError**: The driver is not in the classpath
  - **URL is not correct**. Use the iReport wizard to define the URL or refer to the publisher's site
  - **Connection parameters are not correct.** Check the name of the database, the user and the password.

SQL query defined in the report

- *jasperStudio* provides several tools for working with JDBC and SQL
  - Testing the JDBC connection
  - Automatic discovery of possible fields with their type
  - Ability to automatically save fields
  - A wizard for debugging the queries (Query Designer)
  - A preview of the data returned by the query

DataSet Dialog



Language: sql

ADDRESS ADDRESS
- ☑ ID
- ☑ FIRSTNAME
- ☑ LASTNAME
- ☑ STREET
- ☑ CITY

Field Name	Class Type	Description	Propriétés
ADDRESS_ID	java.lang.Integer		
ADDRESS_FIRSTNAME	java.lang.String		
ADDRESS_LASTNAME	java.lang.String		
ADDRESS_STREET	java.lang.String		
ADDRESS_CITY	java.lang.String		

Sort and filter

- Records retrieved from a data source can be sorted and filtered programmatically by *JasperReport*
  - The filter expression must return a boolean
  - Sorting is based on one or more fields, with their sorting direction
- When using relational databases, it is generally more efficient to use appropriate SQL *where* or *order* clauses.

Jasper Studio
Getting started

Presentation and installation
Data adapters
Fields
Report wizard
Dimensions and band
Simple elements
Parameters, expressions, variables

Fields

- Fields are the only way to display dynamic data from data sources
- Fields are declared via the ***<field>*** with
  - A **name** attribute: Name of the column retrieved by the query. It must be unique in the report.
  - The **class** attribute: Class of the object that can be used in the report.
- <*field*> elements are placed under the <*jasperReport*> element

Type mapping Java/SQL

SQL Type	Java class	SQL Type	Java class
CHAR	String	REAL	Float
VARCHAR	String	FLOAT	Double
LONGVARCHAR	String	DOUBLE	Double
NUMERIC	java.math.BigDecimal	BINARY	byte[]
DECIMAL	java.math.BigDecimal	VARBINARY	byte[]
BIT	Boolean	LONGVARBINARY	byte[]
TINYINT	Integer	DATE	java.sql.Date

Example

```
<jasperReport>
    <field name="id" class="java.lang.Integer"/>
    <field name="name" class="java.lang.String"/>
    <field name="city" class="java.lang.String">
        <fieldDescription>
            Ville de naissance
        </fieldDescription>
     </field>
....
</japserReport>
```

Field rules

- Each field must then correspond to a column of the ResultSet :
  same name and a compatible type

=> If the name of a field does not correspond to a column of the *ResultSet*, an exception will be thrown at run time

=> Columns found in the *ResultSet* but not having any associated fields will not be accessible in the report.

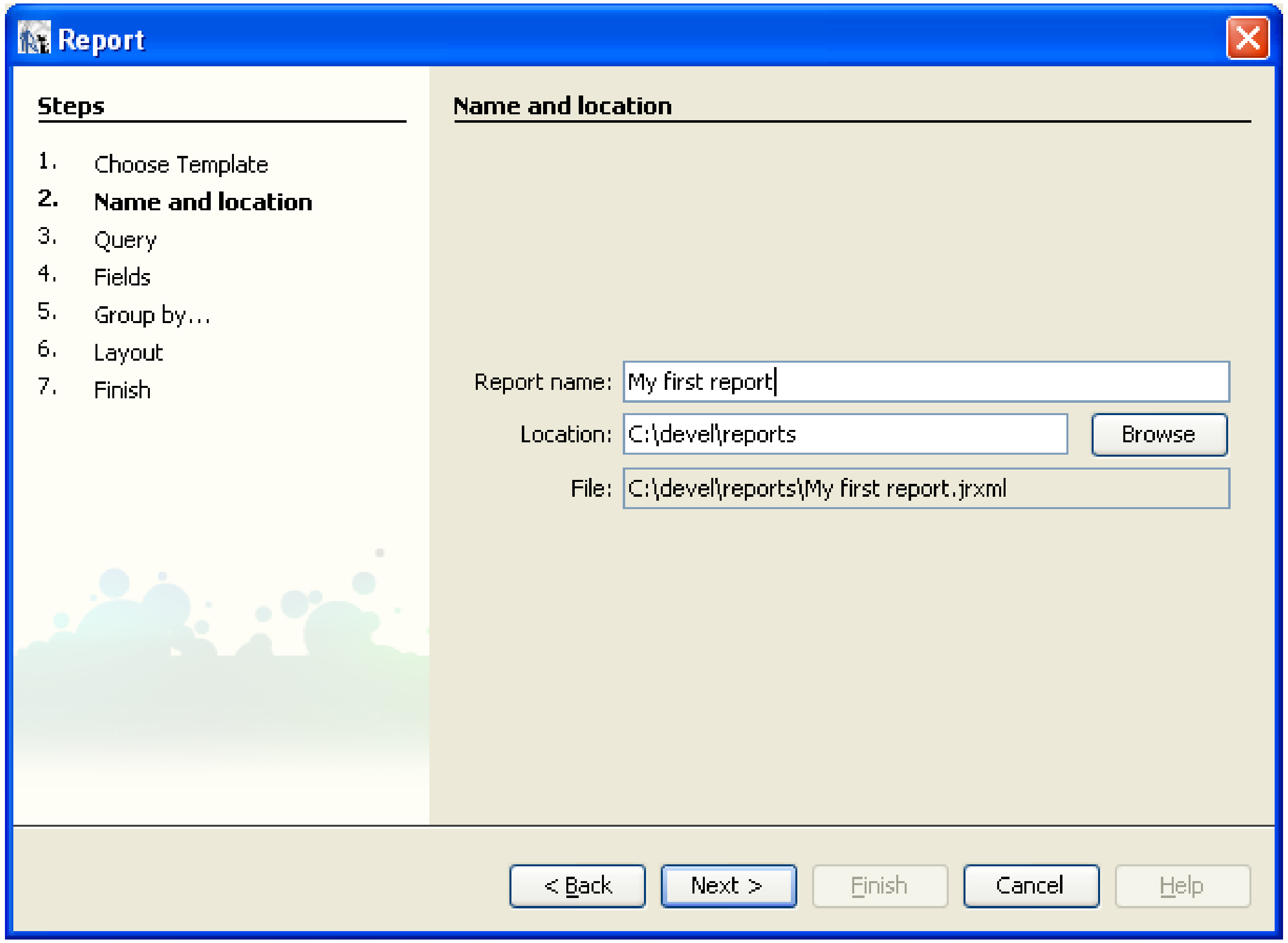
Jasper Studio Getting started

Presentation and installation
Data adapters
Fields
Report wizard
Dimensions and band
Simple elements
Parameters, expressions, variables

Report wizard

- jasperStudio offers a wizard to create a report.
- The wizard is based on **templates**; you can define your own templates
- 5 steps :
  - Selecting a template
  - Name and location of the report
  - Data source and SQL query
  - Fields used
  - Optional groups

Step 2 : File

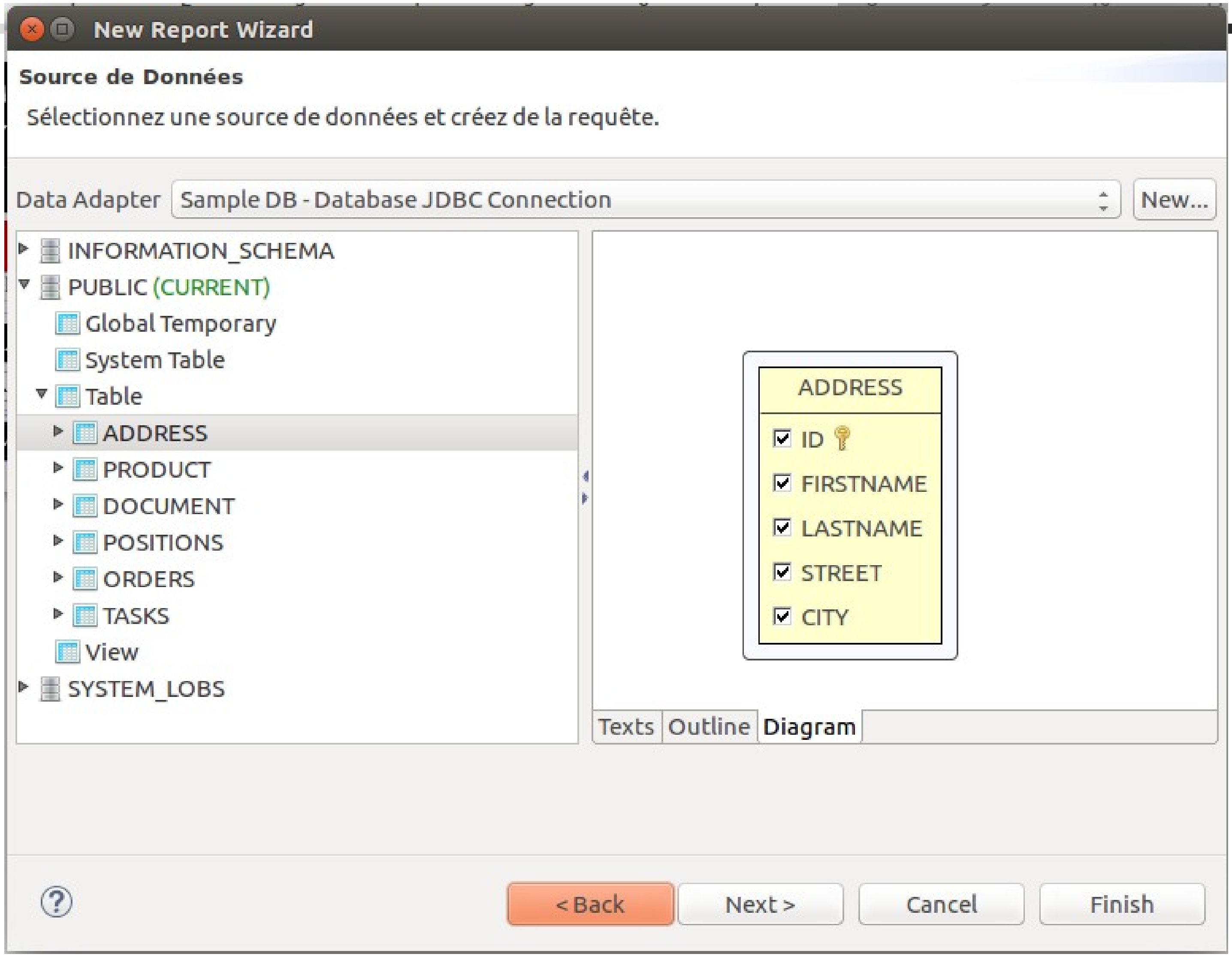


Report name: My first report

Location: C:\devel\reports

File: C:\devel\reports\My first report.jrxml

Step 3 : Query



New Report Wizard

Source de Données

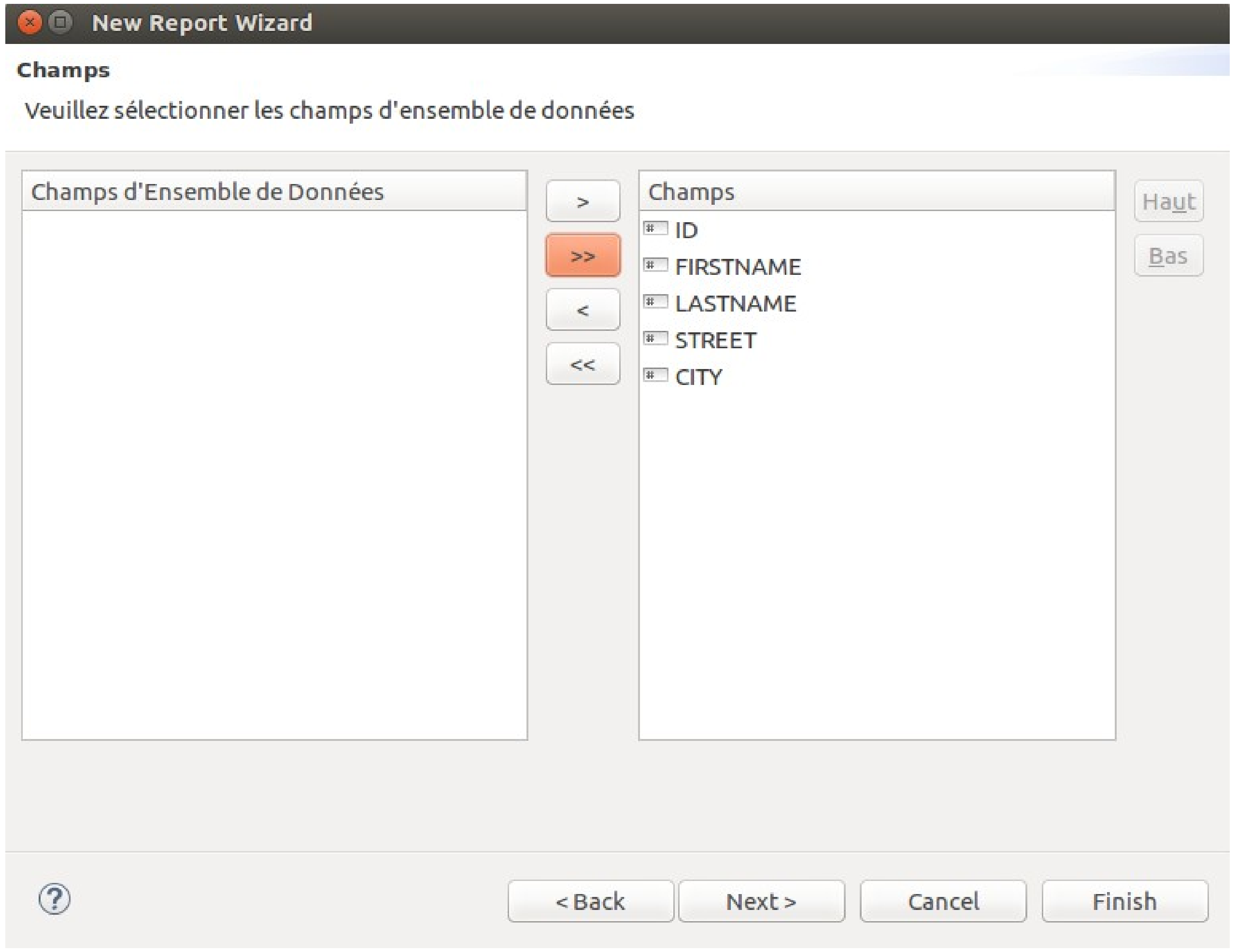
Sélectionnez une source de données et créez de la requête.

Data Adapter: Sample DB - Database JDBC Connection

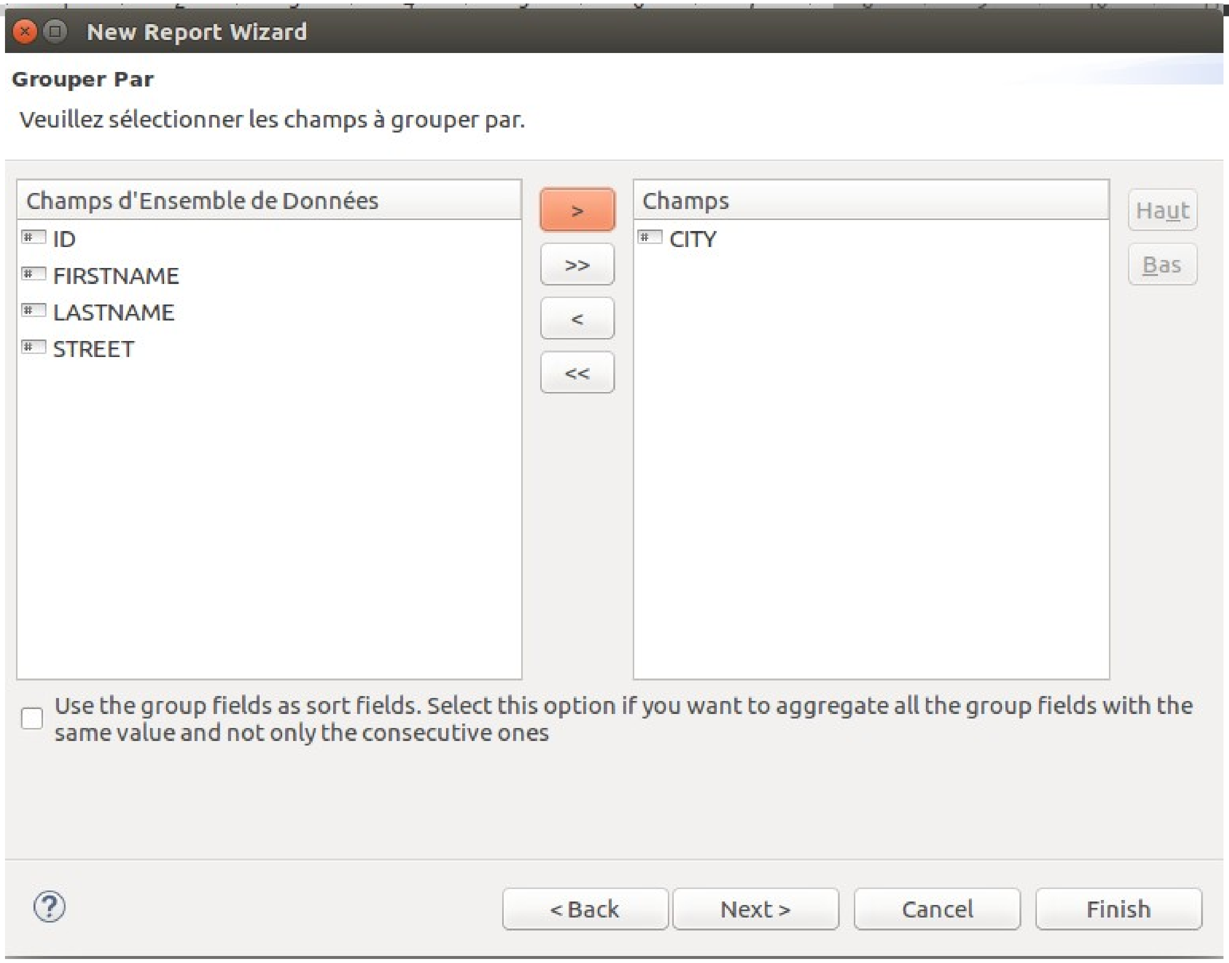
- INFORMATION_SCHEMA
- PUBLIC (CURRENT)
  - Global Temporary
  - System Table
  - Table
    - ADDRESS
    - PRODUCT
    - DOCUMENT
    - POSITIONS
    - ORDERS
    - TASKS
  - View
- SYSTEM_LOBS

ADDRESS
- ✓ ID
- ✓ FIRSTNAME
- ✓ LASTNAME
- ✓ STREET
- ✓ CITY

Step 4 : Fields



Step 5 : Groups



Grouper Par

Veuillez sélectionner les champs à grouper par.

Champs d'Ensemble de Données	Champs
ID	CITY
FIRSTNAME	
LASTNAME	
STREET	

☐ Use the group fields as sort fields. Select this option if you want to aggregate all the group fields with the same value and not only the consecutive ones

Jasper Studio Getting started

Presentation and installation
Data adapters
Fields
Report wizard
Dimensions and bands
Simple elements
Parameters, expressions, variables

Report’s properties

- The first properties of a report are its **dimensions**
  - Default unit is pixel and can be changed in preferences
- Attributes define the drawing area:
  - Page width and height.
  - Width and height of the different margins

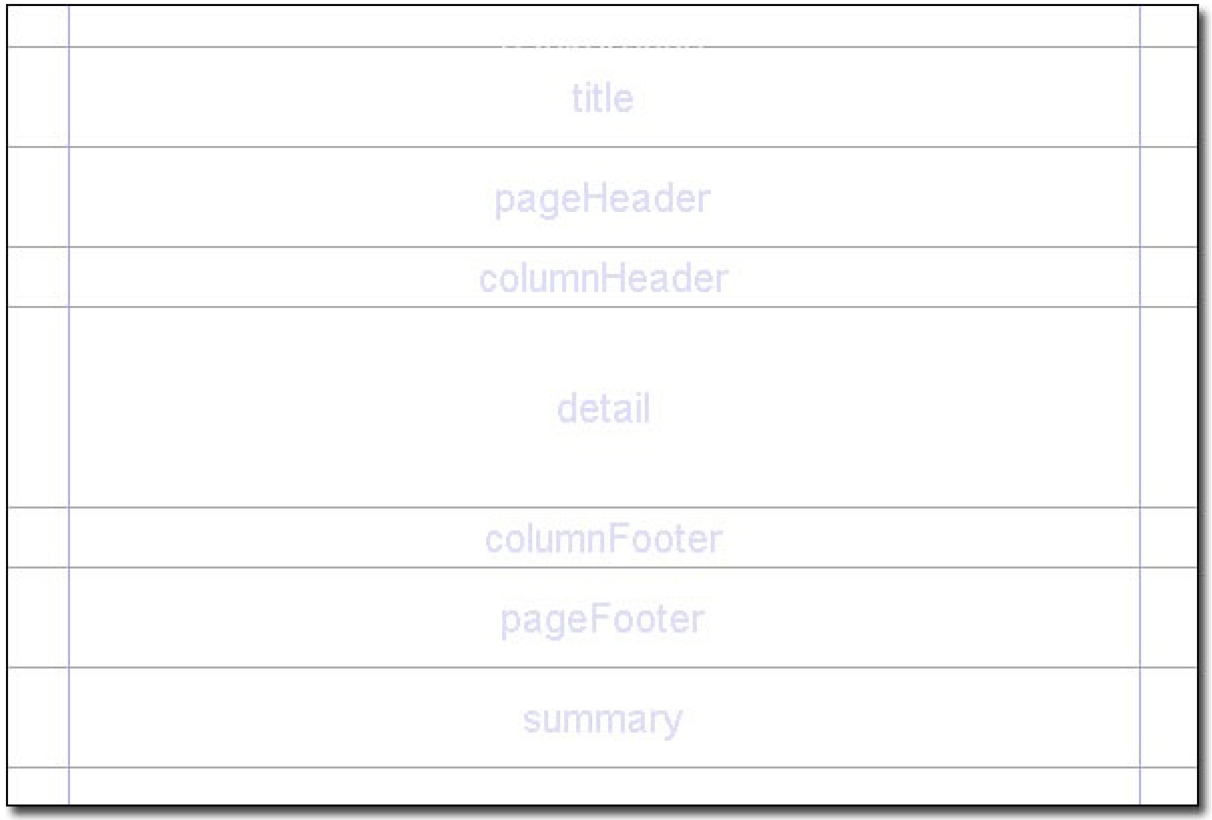
Bands

- A report is divided into **horizontal bands**.
  - Building a report consists of defining the content and dimensions of its bands.
  - Each band contains report elements such as lines, rectangles, images, or text fields.
  - The structure of a Jasper document is predefined with a set of well-defined bands
  - Bands can be added / removed from the report
  - One band corresponds to the ***<band>*** element in the *.jrxml* file

Report’s bands

- Available bands are:
  - ***<background>*** : Page background applied on all pages
  - ***<title>*** : Title appearing only once at the beginning
  - ***<pageHeader>***, ***<pageFooter>*** : Header and footer
  - ***<columnHeader>***, **<*columnFooter*>** : For multi-column reports, header and footer of column
  - ***<groupHeader>***, ***<groupFooter>*** :Header and footer of a group
  - ***<detail>*** : Printed for each row. Possibility to define several <*detail*> sections
  - ***<lastPageFooter>*** : Footer for the last page
  - ***<summary>*** : Printed at the end of the report
  - ***<noData>*** : Alternative text when there is no data

Default Bands



Band’s properties

- ***height***
  - The height of the band in the design view.
    It may increase when merging data (if the dynamic data exceeds the specified height).
    This is therefore the minimum height of the band at generation.
  - When this value is changed, *jasperStudio* checks if the value is acceptable based on the elements that the tape contains.

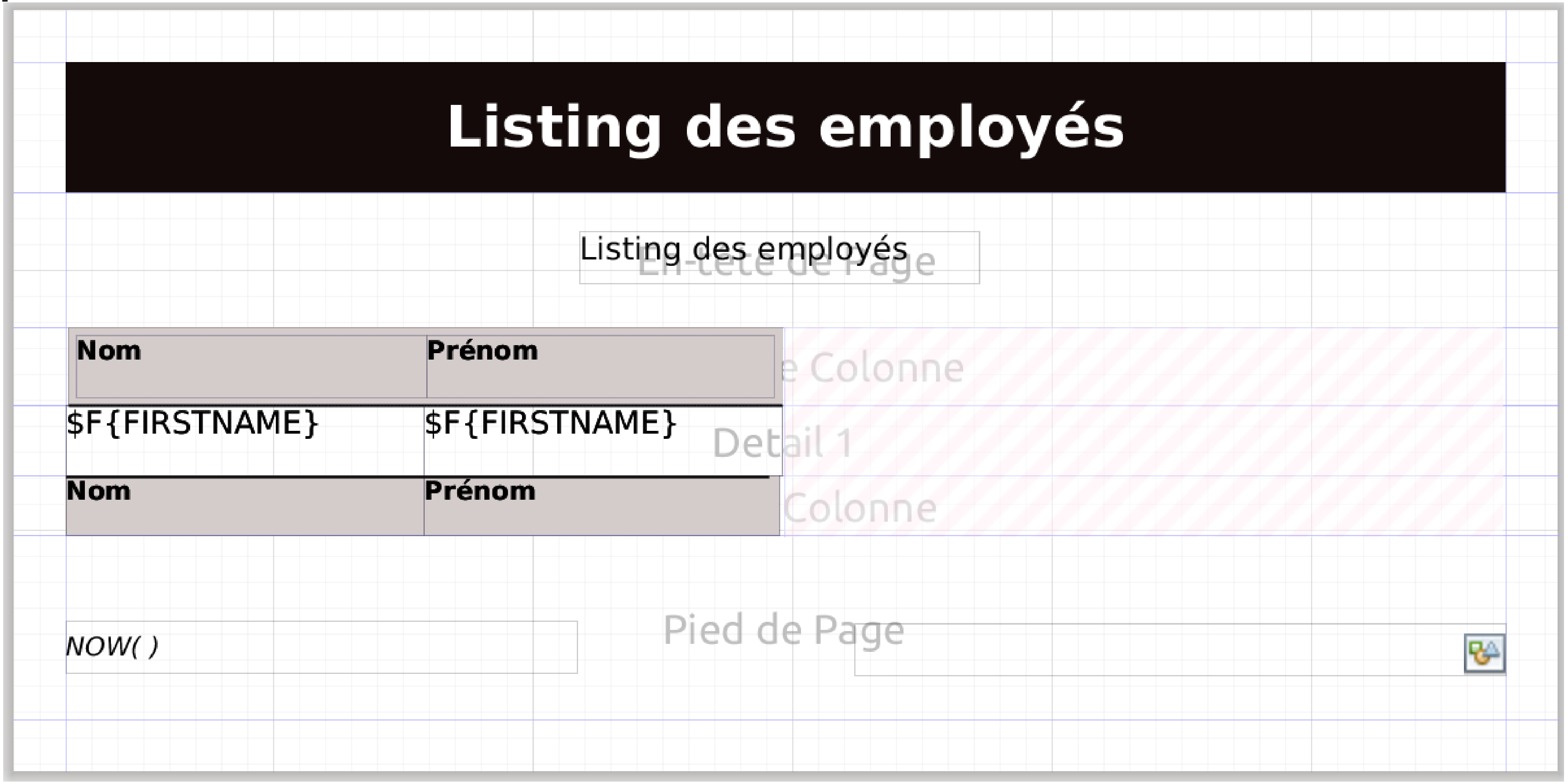
- ***printWhenExpression***
  - Boolean expression used to hide the band
  - Evaluated each time the band is used during generation.

- ***splitType***
  - ***prevent*** : The band is not divided. (Only if its size exceeds the size of a page)
  - ***immediate*** : The band can be divided.
  - ***stretch***: The band is divided into different pages only if its content exceeds the specified height.

Columns

- Bands can be organized in columns in order to optimize the available space.
- The report’s property ***columnCount*** is used to set the number of columns:
  - Column filling can then be vertical or horizontal (Orientation)
  - The sum of the margins, column widths, and each space between columns must be less than the page width, otherwise compilation errors may appear.
  - During design, the elements (controls, images, etc.) are placed only in the first column.

Design

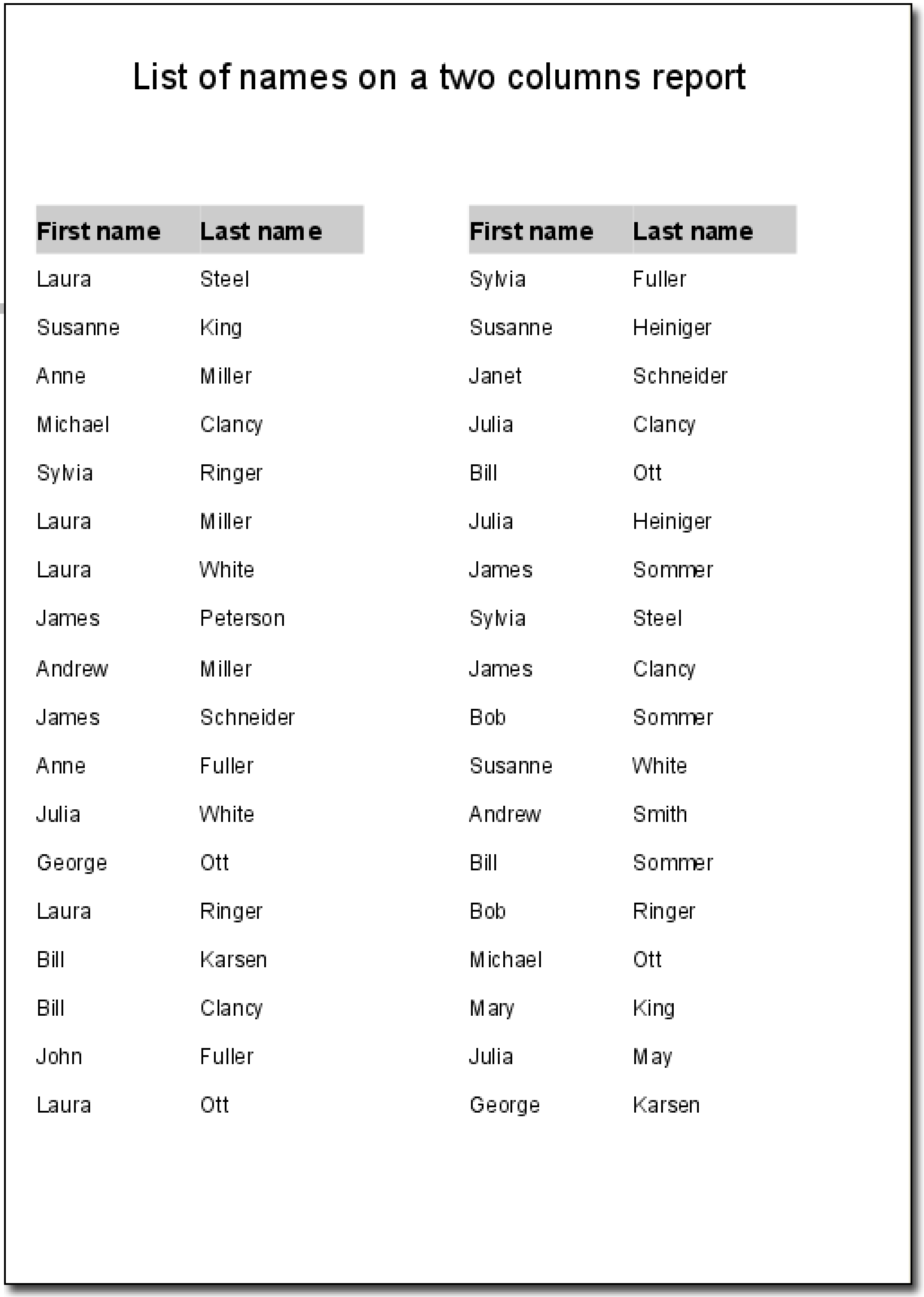

Listing des employés

Listing des employés

En-tête de Page

Nom	Prénom
$F{FIRSTNAME}	$F{FIRSTNAME}
Nom	Prénom

NOW( )



List of names on a two columns report

First name	Last name
Laura	Steel
Susanne	King
Anne	Miller
Michael	Clancy
Sylvia	Ringer
Laura	Miller
Laura	White
James	Peterson
Andrew	Miller
James	Schneider
Anne	Fuller
Julia	White
George	Ott
Laura	Ringer
Bill	Karsen
Bill	Clancy
John	Fuller
Laura	Ott

First name	Last name
Sylvia	Fuller
Susanne	Heiniger
Janet	Schneider
Julia	Clancy
Bill	Ott
Julia	Heiniger
James	Sommer
Sylvia	Steel
James	Clancy
Bob	Sommer
Susanne	White
Andrew	Smith
Bill	Sommer
Bob	Ringer
Michael	Ott
Mary	King
Julia	May
George	Karsen

Jasper Studio Getting started

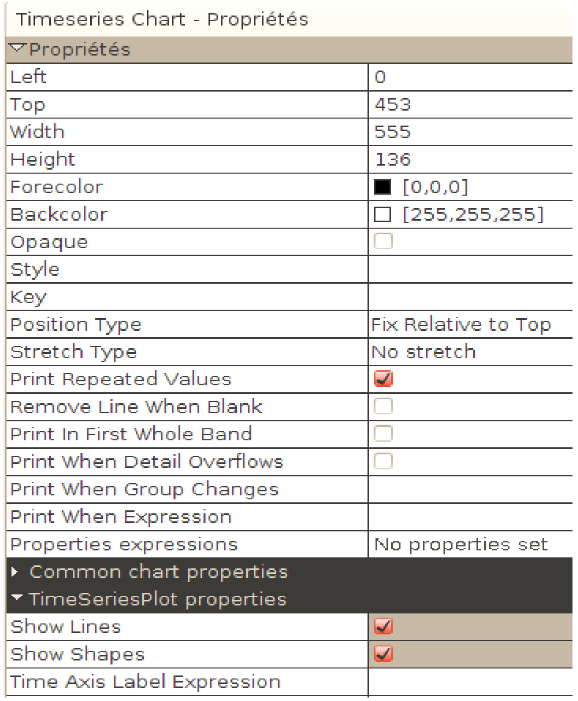
Presentation and installation
Data adapters
Fields
Report wizard
Dimensions and bands
Simple elements
Parameters, expressions, variables

Introduction

- Elements of a report are display objects (Text, Image, Shape, Graph, ...)
- They can be sized, aligned, formatted etc.
- Any content is created through these elements.
- *JasperStudio* palette provides :
  - Line, Rectangle, Ellipse, Static Text, Text Field, Image, Frame, Subreport, Table, List, Crosstab, Chart, Page Break or Column
- It is also possible to extend the components library

Inheritance of properties

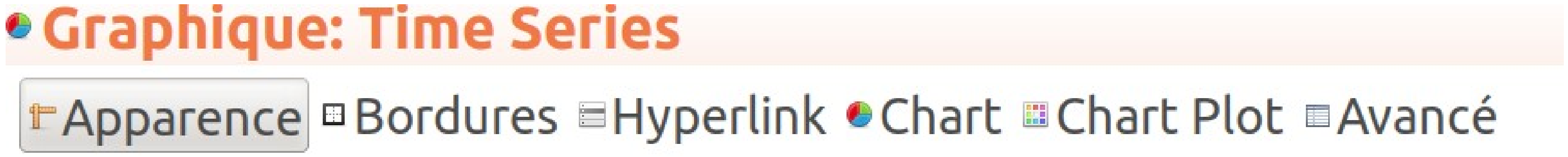
Elements are organized **hierarchically**; they all have common properties and each level of the hierarchy brings its specificities



Timeseries Chart - Propriétés

Propriétés

Left	0
Top	453
Width	555
Height	136
Forecolor	■ [0,0,0]
Backcolor	□ [255,255,255]
Opaque	☐
Style	
Key	
Position Type	Fix Relative to Top
Stretch Type	No stretch
Print Repeated Values	✓
Remove Line When Blank	☐
Print In First Whole Band	☐
Print When Detail Overflows	☐
Print When Group Changes	
Print When Expression	
Properties expressions	No properties set
Common chart properties	
TimeSeriesPlot properties	
Show Lines	✓
Show Shapes	✓
Time Axis Label Expression	



Graphique: Time Series

Size and position

- Elements are positioned via their ***top*** and ***left*** attributes
  - These values are relative to the parent container (band or frame)
- A component cannot overlap 2 bands

- The size of the element is specified by the ***width*** and height ***attributes***

Layout

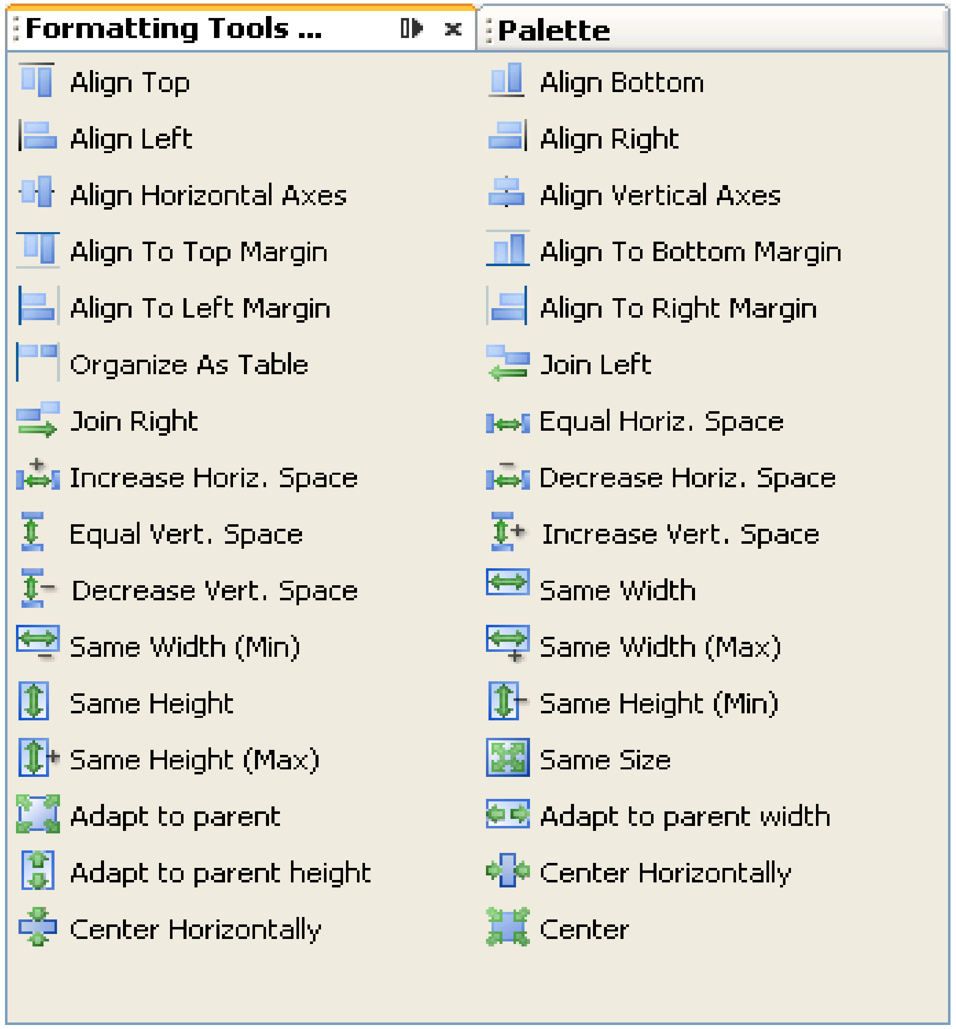
- Elements that may contain other elements such as bands, frames, table or crosstab cells are associated with ***layouts***.
- Layouts affect the size and positioning of items inside the container.

Layouts

JasperStudio provides 4 layouts :

- ***Free layout (default)*** *: The developer freely chooses the size and position of the sub-elements*
- ***Horizontal layout*** *:The elements are arranged horizontally in the container. They are all the same width*
- ***Vertical layout*** *: The elements are arranged vertically in the container. They are all the same height*
- ***Grid layout*** *: The container specifies a number of rows and columns and each element indicates its position with respect to this grid. Grid cells can be merged*

Formatting Tools



Align Top
Align Bottom
Align Left
Align Right
Align Horizontal Axes
Align Vertical Axes
Align To Top Margin
Align To Bottom Margin
Align To Left Margin
Align To Right Margin
Organize As Table
Join Left
Join Right
Equal Horiz. Space
Increase Horiz. Space
Decrease Horiz. Space
Equal Vert. Space
Increase Vert. Space
Decrease Vert. Space
Same Width
Same Width (Min)
Same Width (Max)
Same Height
Same Height (Min)
Same Height (Max)
Same Size
Adapt to parent
Adapt to parent width
Adapt to parent height
Center Horizontally
Center Horizontally
Center

z-index

- The order of the elements inside a band is important because it defines the printing order. This is the ***z-index***.
- If the postions of an element overlap with those of one of its predecessors, it may overlap it during printing.
  - In other situations, we can provide two elements at the same position but only one is displayed thanks to the *printWhenExpression* attribute.
- Some exporters, like the HTML exporter do not support overlapping elements and therefore ignore them.

Element’s common properties

- ***key*** *:* (Optional) Unique name of the element
- ***x, y, width, height*** *:* Positions
- ***foreground, background, opaque*** *:* Colors
- ***style*** *:* Style of the element.
- ***isRemoveLineWhenBlank*** *: Remove space if element is empty*
- ***properties*** : Arbitrary properties of the element.

PositionType

- The ***PositionType*** attribute determines how coordinates are changed when the height of the band changes .
- 3 values are possible:
  - ***FixRelativeToTop***: Default value, coordinates never change.
  - ***Float***: The element is gradually moved down by the previous elements which increases their size
  - ***FixRelativeToBot***: The distance between the element and the bottom of the band remains constant.

StretchType

- The ***StretchType*** attribute defines how to calculate the height of the element during printing.
- 3 values are possible:
  - ***NoStretch***: Default value. The height must not change
  - ***RelativeToBandHeight***: The height of the element is increased in proportion to the increase in the height of the band. (Useful for vertical lines simulating table borders)
  - ***RelativeToTallestObject***: The height of the element is changed according to the deformation of the largest element.

Print

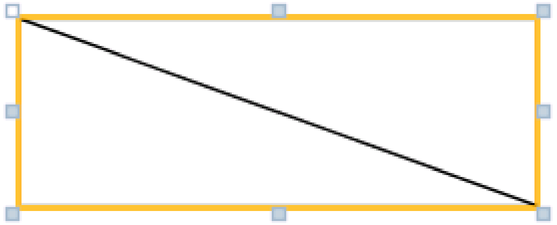
- ***isPrintRepeatedValues***: Print if value equal to the previous record
- ***isPrintInFirstWholeBand***: Ensures the printing of the element during a new page or new column.
- ***isPrintWhenDetailOverflows***: Printed on the next page, when the strip does not fit on the current page
- ***printWhenGroupChanges***: Printed when a group breaks up
- ***printWhenExpression***: Boolean expression conditioning the print

Shapes elements

- Shapes elements are:
  - The lines
  - Rectangles
  - Ellipses
- These elements have 2 additional attributes
  - ***pen***: The line drawing the shape
  - ***Fill***: The filling of the form

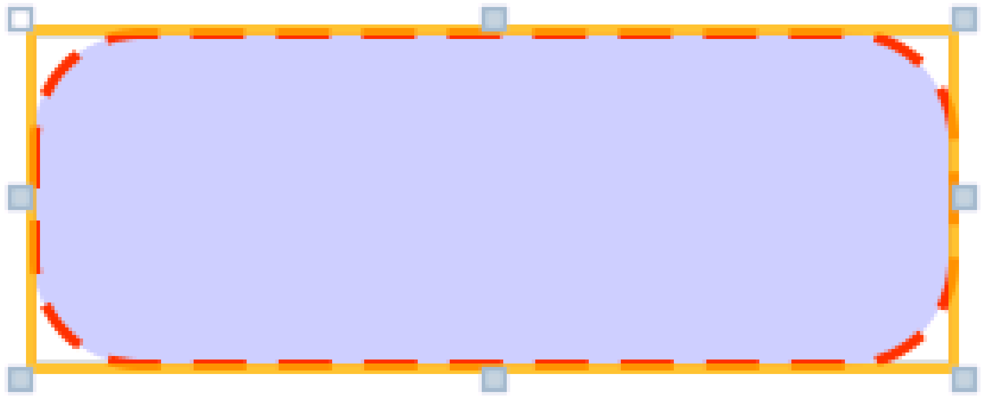
Line

- A line is defined by the diagonal of a rectangle
- The **direction** attribute determines the diagonal:
  - *TopDown*
  - *BotttomUp*



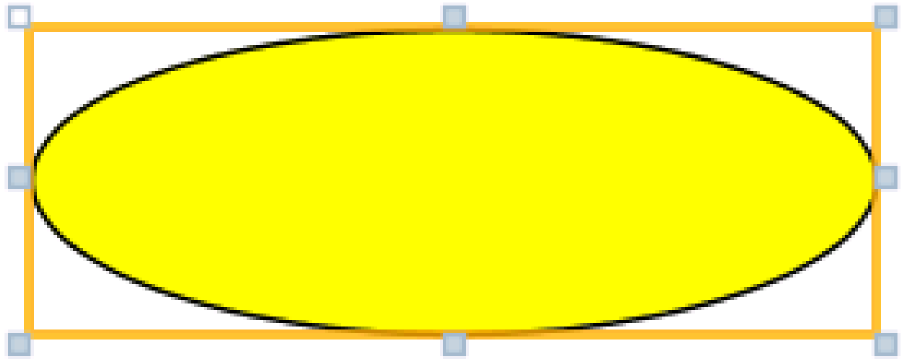
Rounded rectangle / rectangle

- With ***radius*** attribute, rectangles may have rounded corners



Ellipse

- Defined from a rectangle, this element has no specific attributes



Frame

- A ***frame*** is an element that contains and groups other elements. Optionally, it can define a border
- You can nest as many levels as you want.
- A frame allows to:
  - Move all the elements of the frame at the same time
  - Apply a particular layout
  - Apply a style (it will be applied to all the elements contained in the frame)
  - No need to play on the Z axis

Images

- It is the most complex object of graphic objects. It serves to :
  - Insert binary image file (JPEG, PNG, GIF)
  - Or define an Canvas area (Java) to draw or produce a binary image programmatically

ImageExpression

- Images are not stored with the report, only its reference via the ***<imageExpression>*** element is present.
- The ***class*** attribute of the tag indicates the type of the expression; the main values are:
  - ***java.lang.String*** : Location and name of the file.
    Ex : */home/public/images/logo-footer.gif*
  - ***java.io.File*** :
    Ex : *new java.io.File("c:\\myImage.jpg")*
  - ***java.net.URL*** :
    Ex : *new java.net.URL("http://127.0.0.1/test.jpg")*

Portability

- When defining an image location, the location must be valid in the development **AND** production environment
- The best practices to avoid problems in production are:
  - Use a **parameter** in the expression indicating the directory where the images are placed:
    $ P {MY_IMAGES_DIRECTORY} + “myImage.png”
    At generation time, the business application or jasperServer sets this parameter.
  - Use the **JAVA classpath.** (Recommended)
    In this case, the location expression is relative to the classpath and the images are included in the deployed archive.

These methods apply also to any kind of external resources (subreport, external font, scriptlet, bundle resource, etc.)

Loading attributes

isLazy: The image is only loaded when the report is run

isUsingCache: The image is kept in memory to be reused

onErrorType: Specifies the behavior if the image cannot be loaded:

- ***Error***: A Java exception is thrown.
- ***Blank***: The exception is ignored and the image is replaced with a white rectangle.
- ***Icon***: The exception is ignored and a replacement icon is displayed.

Other attributes

- ***Scale***: Defines how the image behaves in relation to the dimensions of the element:
  - ***Clip***: The image is not modified, and only the part corresponding to the dimensions of the element is displayed
  - ***FillFrame***: The image fills the frame
  - ***RetainShape***: The image is adapted keeping the proportions
  - ***RealHeight***: Keep the actual height of the image
  - ***RealSize***: Keep the actual size of the image
- Vertical and horizontal alignments (***halign*** and ***valign***)

Text

- Two elements are available for displaying text:
  - Static texts whose value does not change and is known at design time: ***<staticText>***
  - Text fields whose value is provided with an expression: ***<textField>***

Attributes

- ***Font***: jasperStudio presents all the fonts found in the system but pay attention to compatibility at the time of generation.
- ***Font size*** (integer), Bold and italics, Underline, Strikethrough
- Horizontal and vertical ***alignments***
- ***Rotation***: None, Left, Right, UpsideDown
- ***Line spacing***: Single, 1.5 or Double
- ***Markup***: Using a markup language (HTML or RTF)

Valid types for expression

- The type returned by the expression is indicated in the ***class*** attribute of the *textFieldExpression* element
- Valid types :
  - ***java.lang.String*** :
  - ***java.util.Date, java.sql.Timestamp, java.sql.Time*** :
  - ***java.lang.Boolean*** :
  - ***java.lang.Long, java.lang.Double, java.lang.Short, , java.lang.Float, java.math.BigDecimal, , java.lang.Integer*** : Numbers
  - ***java.lang.Byte*** :
  - ***java.io.InputStream*** :
  - ***java.lang.Object*** :

TextField attributes

- ***isBlankWhenNull*** : If the expression is null, the empty string is used (instead of "null")
- ***evaluationTime***, ***evaluationGroup*** : Determines when the expression is evaluated
- ***isStretchWithOverflow*** : The text field adapts to the content if exceeded
- ***pattern*** : The pattern to apply to format the value of the expression (Date or Number)
  - For example for a date :
    "*dd/MM/yyyy 'at' HH:mm:ss z*"
    04/07/2001 at 12:08:56 PDT

Fonts

Java and therefore JasperReport distinguishes 2 types of Font:

- The **physical fonts** installed on the system
- The **logical fonts**: *Serif, SansSerif, Monospaced, Dialog* and *DialogInput*. They do not have a file installed on the system but they are associated with one of the physical fonts. The mapping may depend on the system !!

Best practice

When using system fonts, you keep the risk that a font used during development is not present in the production environment.

The best practice is to embed the font files in the deployed archive.

This technique called **"Font Extension"** requires providing a jar file containing the font files, a properties file and an XML descriptor associating the font files with the locale and font style (italic, bold, etc.)

The jar can be generated directly from jasperStudio (***Windows → Preferences → Jasper → Font***)

See also the example in / demo / samples

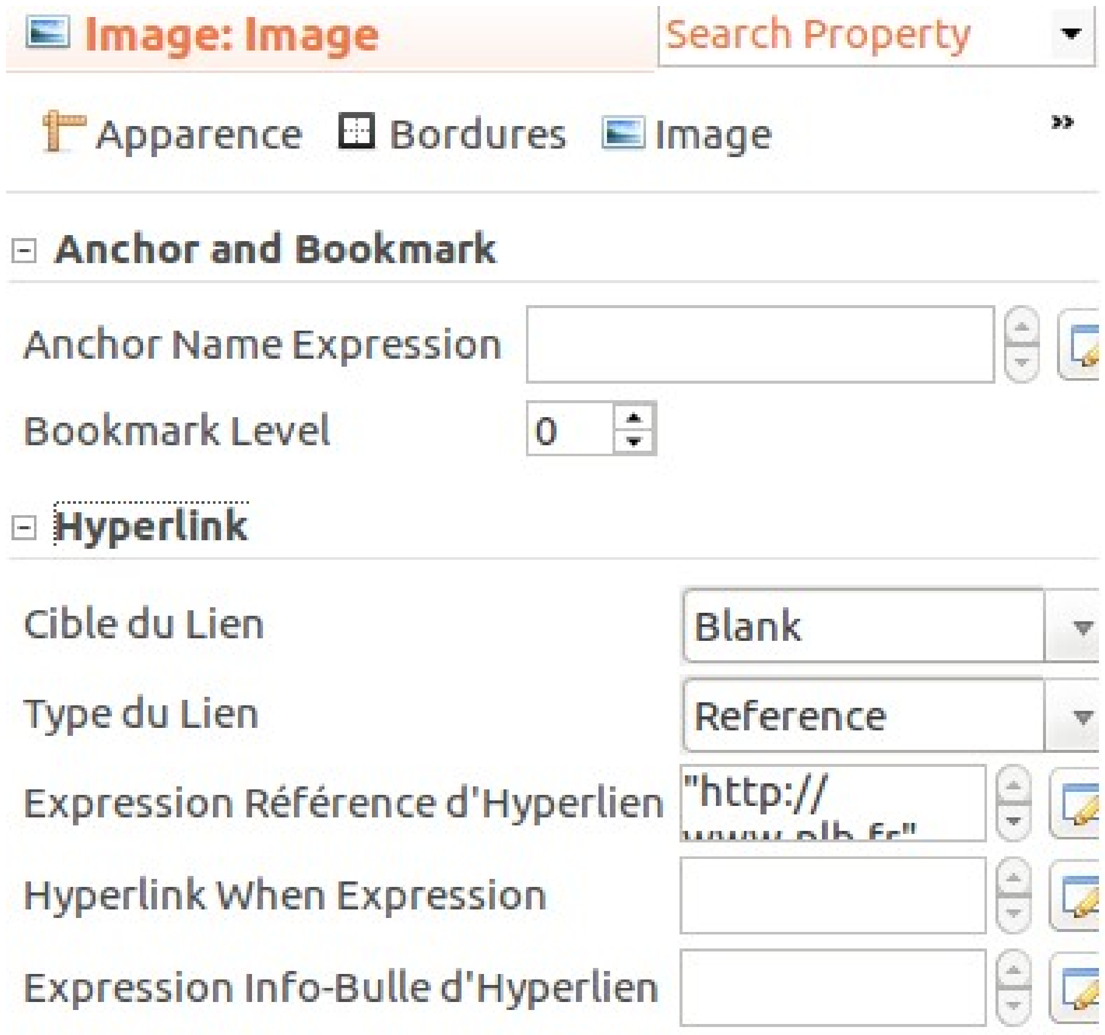
Adding font

- To use a specific *TrueType* font, it must be installed:
  Windows → Preferences → Font

Hypertext links

- Image, text and graphic elements can be used as **hypertext links** to external URLs or internal **anchors**
- Only HTML and PDF formats allow you to create Hypertext links
- The "Hyperlink" dialog window allows you to specify:
  - The expression of the anchor
  - A bookmark level (useful for PDF export)
  - The target of the link
  - The type of link
  - An expression for the tooltip

Anchor and Hyperlink properties



Anchor Name Expression	
Bookmark Level	0
Cible du Lien	Blank
Type du Lien	Reference
Expression Référence d'Hyperlien	"http://
Hyperlink When Expression	
Expression Info-Bulle d'Hyperlien	

Hyperlink types

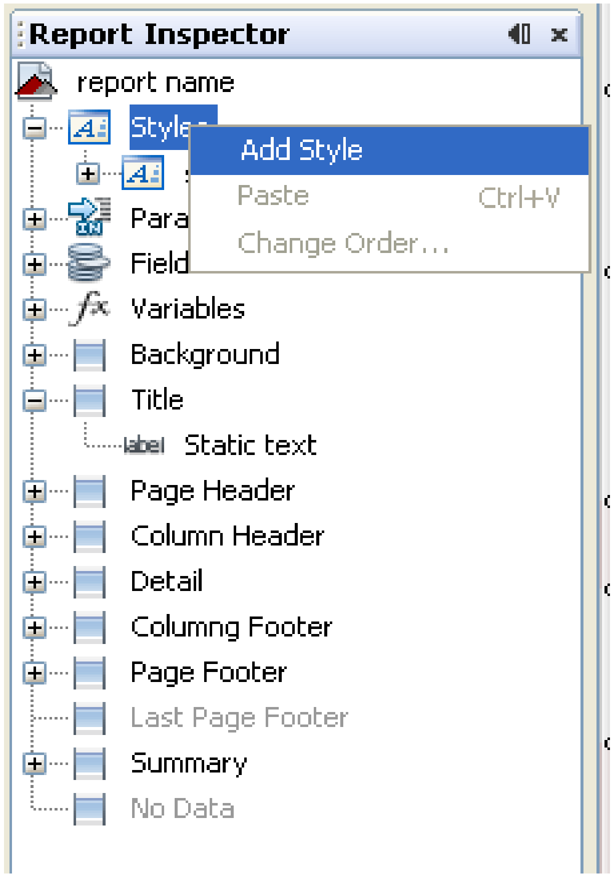
- Threre are different types of hyperlinks :
  - ***Reference*** : external URL
  - ***LocalAnchor*** : Link to an internal anchor
  - ***LocalPage*** : Link to a specific page number
  - ***RemoteAnchor*** : Link to an anchor in another report
  - ***RemotePage*** : Link to a specific page number in another report
  - ***ReportExecution*** : Used to link to a report deployed on JasperServer

Styles

- **Styles** are used to define a set of presentation properties
  Then, they can be applied to several report items

  => The modification of a style affect a set of elements of the report
- Elements can individually override properties defined by their style
- It is possible to define a **default style** for the report.

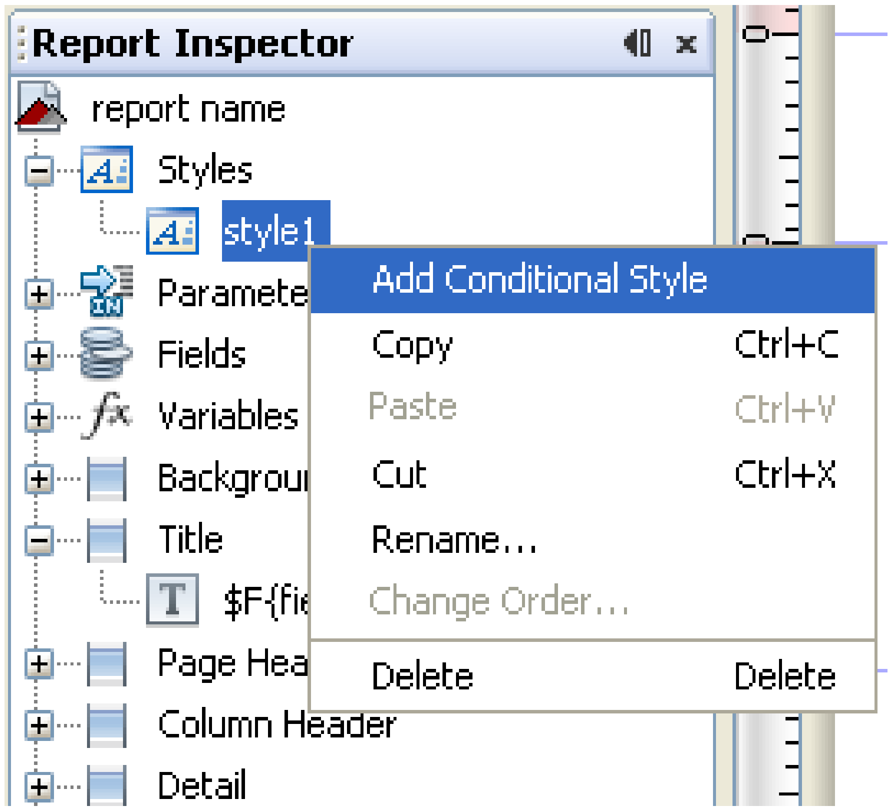
Adding style



Conditional Styles

- The applied style can dynamically change depending on the evaluation of an expression
  Example: Setting the text color in red according to a condition
- *jasperStudio* allows you to associate a style with a **conditional style** which, when certain conditions are met, will override certain properties of the style
- The condition can be expressed as an expression of all the properties of the report (parameters, fields, variables, ...)

Adding conditional style



Style Template

Instead of defining styles independently in each report, it is possible to reference a style library in a report

The ***Style Template File*** is a completely autonomous *.jrtx* file with can be edited with a specific editor

Jasper Studio Getting started

Presentation and installation
Data adapters
Fields
Report wizard
Dimensions and bands
Simple elements
Parameters, expressions, variables

Parameters

- ***Parameters*** are data passed during the generation of the dynamic document which does not come from the data source:
  - For example, presentation data such as the name of the user, the title of the report, ...
  - Parameters of the SQL query. The parameters then act as filters on the recovered data. Example: a period, an amount, etc ...
  - Parameters used for expression evaluation

Declaration

- The declaration of a parameter in a design consists in specifying its name and its type (Java class):
- In the *.jrxml file* :

```
<parameter name="ReportTitle" class="java.lang.String"/>
<parameter name="MaxOrderID" class="java.lang.Integer"/>
<parameter name="SummaryImage" class="java.awt.Image"/>
```

Additional attributes

- The ***defaultValueExpression*** element allows you to specify a default value.
  In this expression, it is not possible to use fields or variables.
- The ***IsForPrompting***, ***Description*** and ***properties*** attributes are not used directly by JasperReport but can be used by other external applications:
  - to determine if this parameter should be provided interactively.
  - Documentation
  - Stores a set of properties related to the parameter.
    For example, the description of the field in different languages

Using parameters in a query

- Parameters can be used in SQL queries:
  - To specify the value of a column in a *where* or *order by* clause (*java.sql.PreparedStatement)* :

*SELECT * FROM Orders WHERE CustomerID =* ***$P{OrderCustomer}***

  - Or as a *String* used to build the SQL query

*SELECT * FROM* ***$P!{TablesClause}***

Clause IN and NOT IN

- *JasperReports* provides specific syntax for use in IN and NOT IN clauses of SQL queries.
  Example :
  `SELECT * FROM ORDERS WHERE` **`$X{IN, SHIPCOUNTRY,myCountries}`**

- The **$ X {}** clause takes 3 parameters:
  - The type of the function to apply (IN or NOT IN)
  - The name of the field to be evaluated
  - The name of the parameter

Predefined parameters

- JasperReports also offers predefined parameters directly usable in reports
  - **REPORT_PARAMETERS_MAP**: A map containing all the predefined parameters.
  - **REPORT_CONNECTION**: Contains the JDBC connection provided by the application.
  - **REPORT_DATA_SOURCE**: Contains the instance of the data source provided by the application
  - **REPORT_MAX_COUNT**: An integer used to limit the size of the data source
  - **REPORT_SCRIPTLET**: Contains the instance of a Java code possibly provided by the application

Predefined parameters (2)

- **REPORT_LOCALE**: The locale (*java.util.Locale*) used for the generation
- **REPORT_RESOURCE_BUNDLE**: The bundle containing the localized labels of the report
- **REPORT_TIME_ZONE**: The time zone (*java.util.TimeZone*) to use for date formatting
- **REPORT_VIRTUALIZER**: The *JRVirtualizer* type instance used for voluminous report
- **REPORT_CLASS_LOADER**: The classloader to use during report generation
- **IS_IGNORE_PAGINATION**: Flag. If true the report is generated without page break

Variables

- A ***variable*** is an object constructed from an expression that can be reused in multiple places in a report by referring to its name.

```
<variable name="FirstLetter" class="java.lang.String">

     <variableExpression>
             <![CDATA[$F{ShipCountry}.substring(0, 1).toUpperCase()]]>
     </variableExpression>

</variable>
```

- In its expression, a variable can reference another variable if the latter has been declared before.

Agregations

- *JapsperReport* allows you to perform certain types of predefined aggregation calculations: *count, sum, average, lowest, highest, variance,* etc.
- For example :

```
<variable name="QuantitySum"
        class="java.lang.Double" calculation="Sum">
    <variableExpression>$F{Quantity}</variableExpression>
</variable>
```

Resetting variables

- It is possible to indicate the time when the variable will be reset via the ***resetType*** attribute :
  - ***report*** : Initialized once at the start of the report (default)
  - ***page*** : Reset on every page
  - ***column*** : Reset at each column
  - ***group*** : Reset for each group.
- For example, to calculate the total of a quantity field for each page

```
<variable name="QuantitySum" class="java.lang.Double"
            resetType="Page" calculation="Sum">
   <variableExpression>$F{Quantity}</variableExpression>
   <initialValueExpression>new Double(0) </initialValueExpression>
</variable>
```

Values taking part in the calculation

- By default, all the values of the *DataSet* are taken into account for the calculation of the variable.
- Sometimes, it is necessary to modify this behavior by playing on the ***IncrementType*** attribute which can take the following values :
  - ***None*** : For each row
  - ***Page*** : For each page
  - ***Column*** : For each colum
  - ***Group*** : For each break of a particular group
  - ***Report*** : At the end of the report

Predefined variables

- JasperReport provides predefined variables:
  - ***PAGE_NUMBER*** : The number of the current page
  - ***COLUMN_NUMBER*** : The number of the current column
  - ***REPORT_COUNT*** : Counter of rows
  - ***PAGE_COUNT*** : Counter of rows for the current page
  - ***COLUMN_COUNT*** : Counter of rows for the current column
  - ***GroupName_COUNT*** : Counter of rows for the current group
- See the JRVariable documentation for an exhaustive list

Expressions

- **Expressions** are used to express the value of certain elements of the report: variables, grouping criteria, or even just a visual parameter of the report.
- They can use operators and language methods
- The available languages are Java, Javascript and Groovy
- Expressions can refer to parameters, fields or variables or resourceKey

Expression notation

- The **$ {}** notation is used to reference fields, variables, parameters or resourceKey
  - **$ F {<fieldName>}** for a field
  - **$ V {variableName}** for a variable
  - **$ P {parameterName}** or
  - **$ P! {ParameterName}** for a parameter.
  - **$ R {resourceKey}** for localization

Expressions types

- The type of the expression is determined by its context of use (condition, calculation, text, etc.)
- If the language is Java, the supported types are: *Boolean, Byte, Integer, Long, Float, Double, String, Date* and *Object*
- In Javascript and Groovy, expressions are not typed

Operators

Operator	Description
+	Addition of 2 numbers or concatenation of 2 character strings
-	Substraction
*	Multiplication
/	Division
%	Modulo (Remainder of an integer division)
||	OR logical operator
&&	AND logical operator
== and !=	Equality or inequality test (Warning in Java, different from the equals () method

Conditional expression

- If-else expressions can be specified using the syntax:
  (<condition>) ? <value on true> : <value on false>
- Example :

```
(($F{name}.length() > 50) ? $F{name}.substring(0,50) :
    $F{name})
```

- This is a recursive syntax

```
(($F{name}.length() > 50) ?
    (($F{name}.startsWidth(“A”)) ? “AAAA” : “BBB”) :
    $F{name})
```

- It is still possible to delegate more complex expressions to a static method of a Java class available in the classpath

```
MyFormatter.toRomanNumber( $F{MyInteger}.intValue() )
```

Expressions of the report

- Different XML elements can define expressions:
  - ***<variableExpression>*** : Define the value of a variable. Is placed inside an element <*variable*>
  - ***<initialValueExpression>*** : Define the initial value of a variable. Is placed inside an element <*variable*>
  - ***<groupExpression>*** : Grouping criterion for groups. Is placed inside an element <*group*>*.*
  - ***<printWhenExpression>*** : Boolean expression acting on the print. Can be placed inside the elements <*band*>, <*reportElement*>*.*
  - ***<imageExpression>*** : Expression determining the image to insert. Is placed inside an element <*image*>
  - ***<textFieldExpression>*** : Expression determining a text value. Is placed inside an element <*textField*>

Evaluation of expressions

- By default, expressions are evaluated immediately during generation.
- We can modify this behavior by modifying the ***evaluationTime*** attribute which can take the following values:
  - ***Now***: Immediately (default)
  - ***Report***: At the end of the generation
  - ***Page***: At the end of the page
  - ***Column***: At the end of the column
  - ***Group***: At the end of the group
  - ***Band***: At the end of the band in which the expression is found
  - ***Auto***: When the expression depends on interdependent variables, we leave the choice to the engine to choose the right moment for the evaluation

Properties

- Properties are **key / value pairs** that can be associated with the report or with report items.
- They can be used for all kinds of purposes: configuring an export, specifying a theme for a graphic, encrypting a PDF, etc.
- *JasperReport* already uses a number of properties that can be reset to their default values.
  - For example: *net.sf.jasperreports.text.truncate.at.char* defines how a text is cut

Resource Bundle

- ***Resource bundles*** are used to internationalize a report, it consists of a set of files (1 per language) containing the translations of the texts appearing in the report.
- File names follow a naming standard :
  - They are suffixed by the country code and eventually the language (for example, _fr_FR for French / France)
  - They have the extension *.properties*
    Examples : *labels_fr_FR.properties* or *labels_fr.properties*
- The content of the file is made up of lines:
  resourceKey= translated_label
  - ***labels_fr.properties :***
    footer.confidential = Ces données sont confidentielles
    generated.by = Généré par {0}
  - ***labels_en.properties*** :
    footer.confidential = These data are confidential
    generated.by = Generated by {0}

Association with the report

- The association of a *ResourceBundle* with a report is performed by :
  - Either by using the ***resourceBundle*** attribute of the report
  - Either at runtime by providing a value for the predefined parameter **$P{REPORT_RESOURCE_BUNDLE}**.
- The generation language can be specified by the parameter **$P{REPORT_LOCALE}**.
  - In the generated output, the library preserves the text direction information so that documents generated in languages such as Arabic or Hebrew can be correctly rendered.

Usage in the report

- To localize a label, we use:
  - the syntax ***$ R {<bundleKey>}*** in expressions.
  - Or the ***msg ()*** method which offers functionality similar to the *java.text.MessageFormat* class by allowing parameters.

Examples

- *$R{footer.confidential}*

- *msg($R{generated.by},$P{USER_LASTNAME})*

Complex elements

Groups
Subreports
Datasets, List and Tables
Charts
Crosstab

Group

- **Groups** are used to group data in a report according to **break criteria** specified through expressions.
- Defining a group usually creates new bands:
  - a header (<groupHeader>)
  - a group footer (<groupFooter>)
- *Report → Right click → Create group*
- When merging data, JasperReport tests all defined group expressions to determine if a group break occurs.
- If so, it inserts the associated group header and footer.

Example

```
<group name="CITY">
   <groupExpression><![CDATA[$F{CITY}]]></groupExpression>
   <groupHeader>
         <band height="37">
           <frame>
              <reportElement mode="Opaque" x="0" y="7" width="555" height="24"
   forecolor="#B89F7D" backcolor="#000000"/>
              <textField isStretchWithOverflow="true">
                          <reportElement style="SubTitle"
   isPrintRepeatedValues="false" x="2" y="0" width="479" height="24"
   forecolor="#FFFFFF"/>
                    <textElement><font isBold="false"/></textElement>
                    <textFieldExpression class="java.lang.String">
                                <![CDATA[$F{CITY}]]>
                    </textFieldExpression>
              </textField>
          </frame>
       </band>
    </groupHeader>
    <groupFooter>
      <band height="6"/>
    </groupFooter>
</group>
```

Order

- The engine does not sort automatically the records.
- => You must therefore always add an **ORDER** clause in the SQL query
- In the case of an XML source, you must then explicitly ask JasperReport to perform the sort.

Nested groups

- There is no limit to the number of groups defined in a report.
- The order is important because the groups are contained within each other.
- When a group encounters a break, all the contained groups are reset.

Attributes of a group

- ***groupExpression*** : Break expression
- ***isStartNewColumn*** : If tr*ue*, a column break is performed on each group break.
  In the case of a report with only one column, a page break is performed
- ***isStartNewPage*** : If true, a page break is performed on each break
- ***isResetPageNumber*** : Page break + page number reset
- ***isReprintHeaderOnEachPage*** : The group header is repeated on each page
- ***minHeightToStartNewPage*** : Minimum height triggering the page break

Group variables

- When adding a group, *JasperReport* automatically creates a variable: ***<groupName> _COUNT***
- It is an iteration variable
- Used at the bottom of the group, it gives the number of records for the group

Complex elements

Groups
Subreports
Datasets, List and Tables
Charts
Crosstab

Introduction

- **Subreports** allow for complex report creation and reuse.
  - They are often used in "master-detail" type reports or when the structure of a report is not sufficient to describe the complexity of the document to be generated.
- A subreport is a normal report included in another report.
  => Any report can be used as a *subreport* in another report without modification.
- There is no limit for nesting subreports.
- Lists / tables provide a simplified alternative to subreports

Usage

- A subreport is designed completely independently of another report.
- It is included into the parent report with the ***<subReport>*** element which must provide:
  - **Reference** to the subreport .jrxml file
  - The **connection** to use
  - The **input parameters**
  - The **return values** that may be used in the parent report

Margins and dimensions

- The margins of a subreport are usually set to 0.
- When using it, it is possible to indicate a size to the <*subReport*> element
  - It is not this size used during generation : the size is defined in the subreport.
  - The <*subReport*> element therefore only allows you to position the subreport in the parent report
  - However, it is recommended to set the width of the element to the same value as the size of the subreport to "preserve" the WYSWIG aspect.

Reference

- The reference to the file is made via the ***<subReportExpression>*** sub-element and its *class* attribute which can take the following values:
  - ***java.lang.String***: The location of the Jasper file.
    Relative to file system or CLASSPATH or URL.
    The *isUsingCache* attribute avoid reloading the subreport during generation.
  - ***java.io.File***: Java expression returning a File object
  - ***java.net.URL***: A URL

Datasource

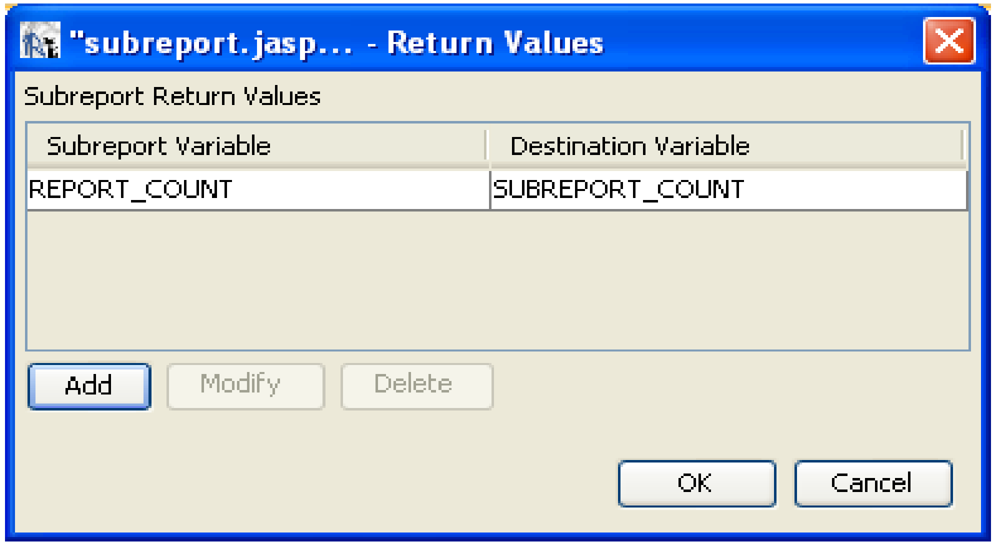
- 3 options :
  - The subreport uses an SQL query and requires the same JDBC connection used by the parent
  - The subreport uses another type of data source such as an XML file for example
  - The subreport does not use a data source and is used to simulate the inclusion of static document portions (header, footer, ...)

Parameters of a subreport

- Parameters of a subreport can be provided:
  - Either in the form of a Map by the ***<parametersMapExpression>*** element. Then, all the parameters of the parent can be passed with
    $ P {REPORT_PARAMETERS_MAP}
  - Or by using the ***<subreportParameter>*** element for each parameter. (these elements override the possibly specified map).

Return values

- It is possible to update variables of the parent report with the calculated variables of the subreport
  - Just define the mappings between the subreport variable and the parent variable.
  - Aggregation functions can also be applied



Subreport Return Values

Subreport Variable	Destination Variable
REPORT_COUNT	SUBREPORT_COUNT

Methodology

- Define a variable in the parent report whose calculation method is "***System***"
- Define the mapping between the created variable and one of the variables of the subreport.
  The types must match
- If the value of the variable is displayed in the same band as the subreport, then the **“Evaluation time”** attribute of the text field must be set to **“band”**

Complex elements

Groups
Subreports
Datasets, List and Tables
Charts
Crosstab

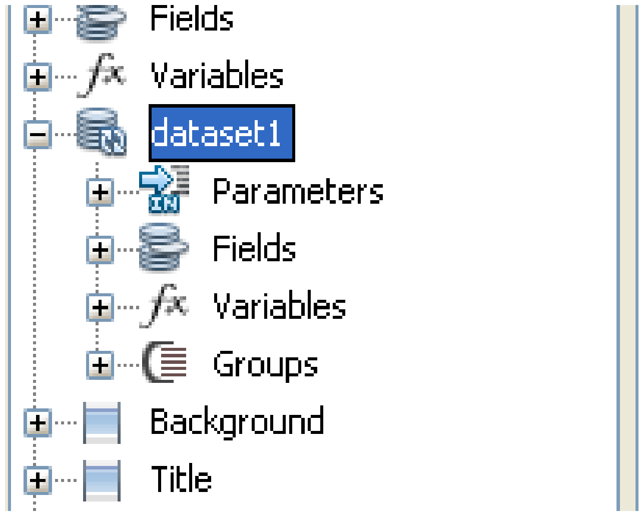
Datasets

- Charts or crosstabs often need to use data not provided by the main report query
- With ***datasets***, different graphs or crosstabs can be included without using sub-reports
- A dataset is a mix between :
  - A query
  - And a subreport : it contains parameters, fields, variables and groups but has no layout information.

<*subdataset*> element

- Datasets are declared with the ***<subdataset>*** element.
  - The number of datasets is not limited.
  - The main properties of the dataset are:
    - The query
    - The fields, parameters, variables and groups that will be usable by charts, crosstabs, ...

Outline


Fields
Variables
dataset1
Parameters
Fields
Variables
Groups
Background
Title

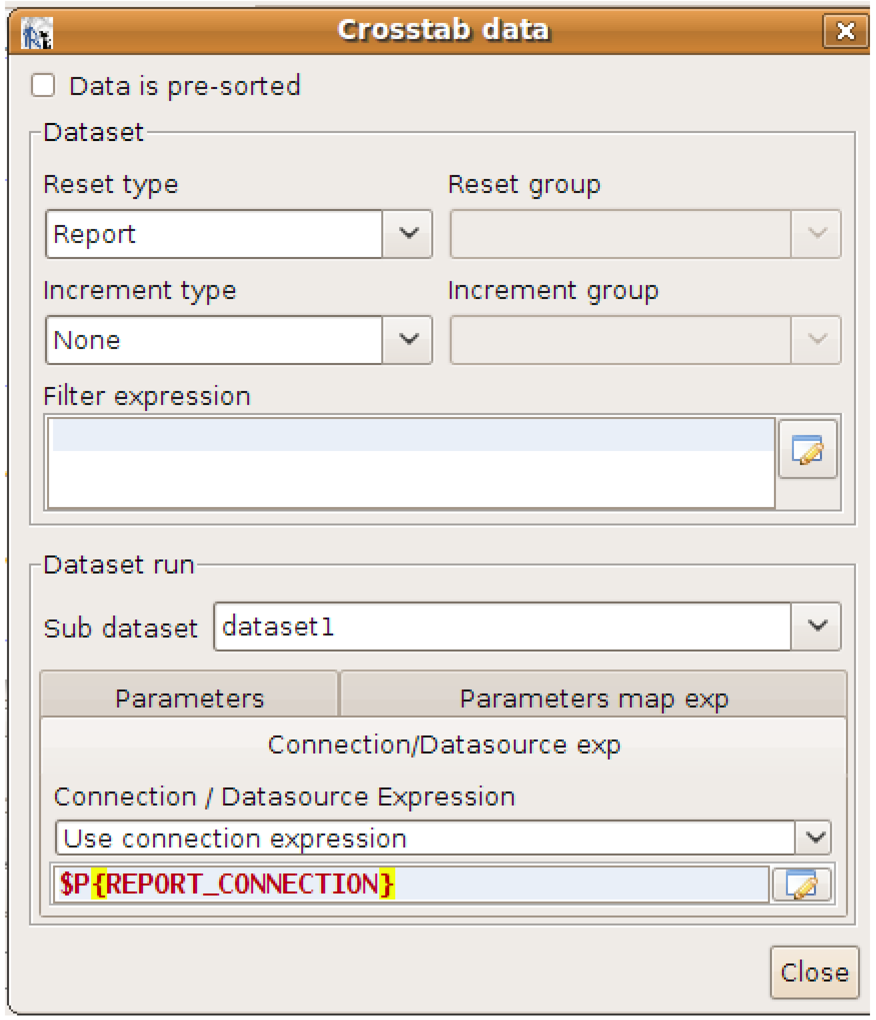
Dataset run

- A dataset can be associated with one or more iterative elements: list, table, chart or crosstab via a ***dataset run***
- The configuration of a dataset run is similar to the configuration of a subreport, you must specify:
  - Connection to the data source
  - Parameters mapping
  - Any return values
- All the fields, parameters and variables of the dataset can then be used in the iterative element

Dataset context

- Variables, parameters or groups of the dataset can only be used by iterative elements that use the dataset.
  They are not visible in the main report
- The groups are therefore only used for the calculation of the variables of the dataset. They do not cause additional print spaces in the main report

Dataset run



Crosstab data

- [ ] Data is pre-sorted

Dataset

Reset type	Reset group
Report	
Increment type	Increment group
None	
Filter expression	

Dataset run

Sub dataset: dataset1

Parameters | Parameters map exp | Connection/Datasource exp

Connection / Datasource Expression: Use connection expression

```
$P{REPORT_CONNECTION}
```

Close

Other properties

- Apart from the ***name*** attribute which must be unique in the report, a dataset can define:
  - ***scriptletClass*** : A specific scriptlet class
  - ***resourceBundle*** : Label localization files
  - ***whenResourceMissingType*** : behavior to adopt if a bundle key is missing
  - ***Filter*** and **sort** expressions

List

The **List** element is the most simple iterative element

- It does not offer variables or bands
- But it may have parameters

The usage of a list consists in "iterating" on another data set than the main data

The element can be placed in any section of the report, the display consists of printing items in the form of a list; the only latitudes are:

- Item size
- The fill direction of the list (Horizontal or Vertical)

Table

The **table** element displays tabular data

Using a table is generally easier than using individual text fields with borders.

- A separate XML DTD is associated with the table element http://jasperreports.sourceforge.net/sample.reference/table/index.html

Table structure

Inside a table, **sections** are declared in order to group the content:

- table header and footer
- header and footer
- header and footer
- as well as an unlimited number of nested group headers and footers.

Each section is made up of a list of **columns**.

- These columns can be grouped
- They can be hidden depending on a boolean condition

Each column can be associated with a **cell** that contains one or more display elements.

Table Wizard

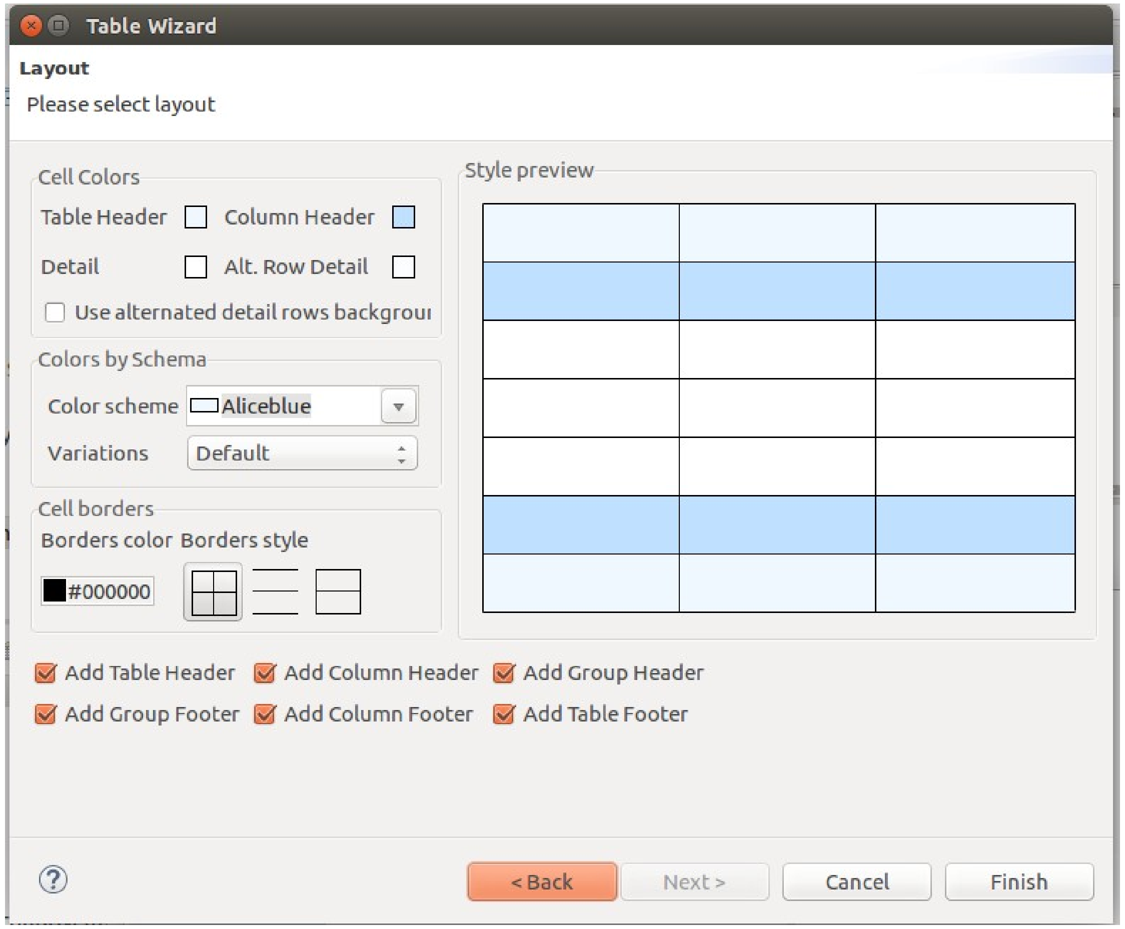


Table Wizard

Layout

Please select layout

Cell Colors

Table Header ☐ Column Header ☐

Detail ☐ Alt. Row Detail ☐

☐ Use alternated detail rows backgrou

Colors by Schema

Color scheme: Aliceblue

Variations: Default

Cell borders

Borders color: #000000

Borders style

Style preview

☑ Add Table Header ☑ Add Column Header ☑ Add Group Header

☑ Add Group Footer ☑ Add Column Footer ☑ Add Table Footer

Cells

Finally, the result of a table is a series of **cells** behaving like frames where content can be placed.

Some cells are

- In header or footer sections
- In detail ou group section

Cells can be removed if not neededd

Their Layout can be edited (Vertical by default)

Deferred evaluations are possible

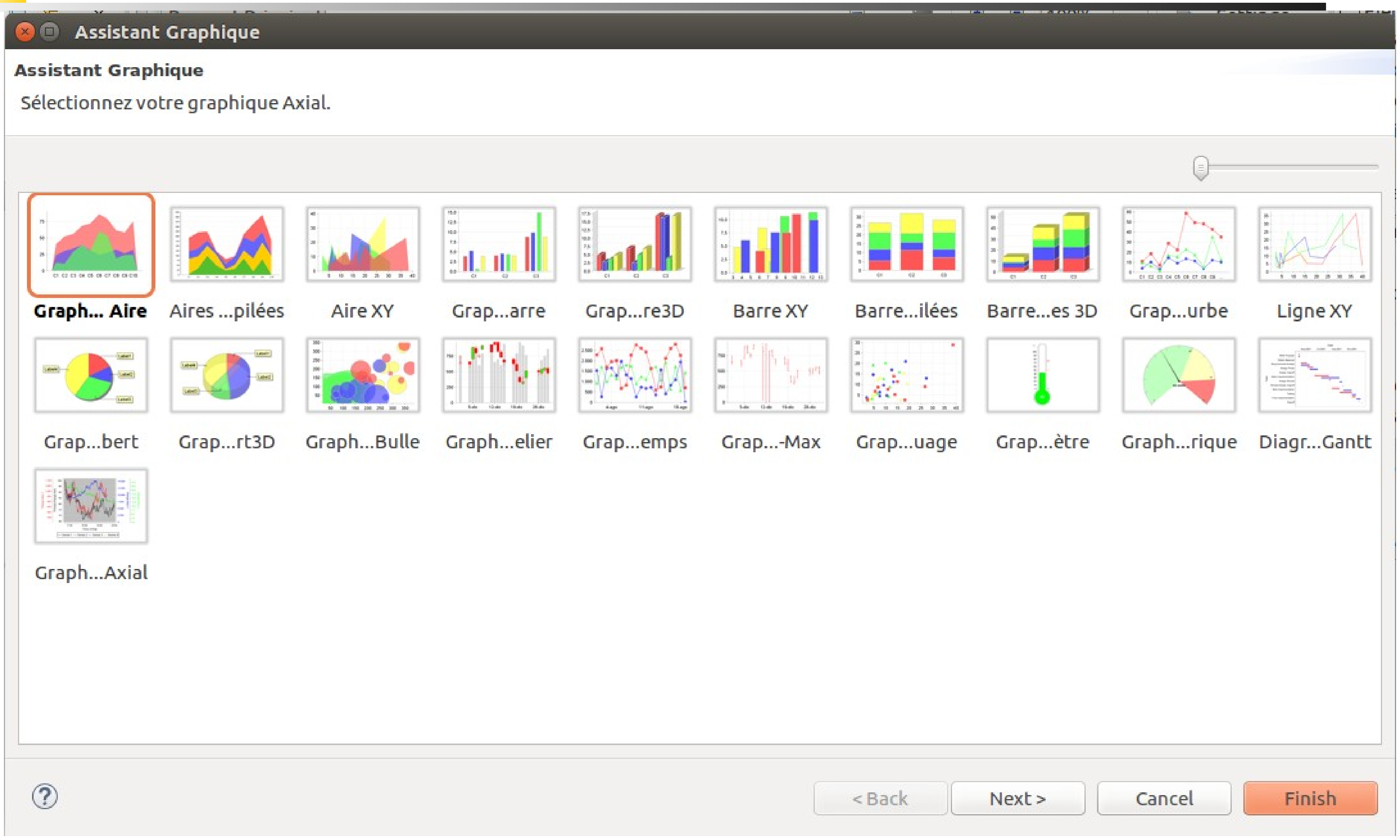
Complex elements

Groups
Subreports
Datasets, List and Tables
Charts
Crosstab

Charts

- JasperReports offers support for creating charts based on the *JFreeChart* library
- The most comprehensive documentation is available in the JasperReport Ultimate Guide and the JFReeChart site
- When adding a chart element, 4 types of information must be configured:
  - The **type** of chart
  - The **dataset** used (main dataset or a subdataset)
  - The **expressions** of the values of the graph (depending on the type of graph)
  - The **rendering** parameters (colors, layout, etc.) also depend on the type of graphic

Charts



Assistant Graphique

Assistant Graphique

Sélectionnez votre graphique Axial.

Dataset

- When defining the dataset, attributes common to all types of dataset can be defined:
  - ***ResetType*** and ***ResetGroup*** are used to periodically reset the dataset.
    Useful if the dataset takes a parameter (parameter of a group for example)
  - By default, all rows are displayed in the chart and crosstab. We can change this behavior via the following attributes:
    - ***IncrementType*** and ***IncrementGroup*** specify when new values should be added to the dataset.
    - The expression ***IncrementWhen*** returns a boolean that determines whether the current record should be added to the data set. (default all)

Common chart properties

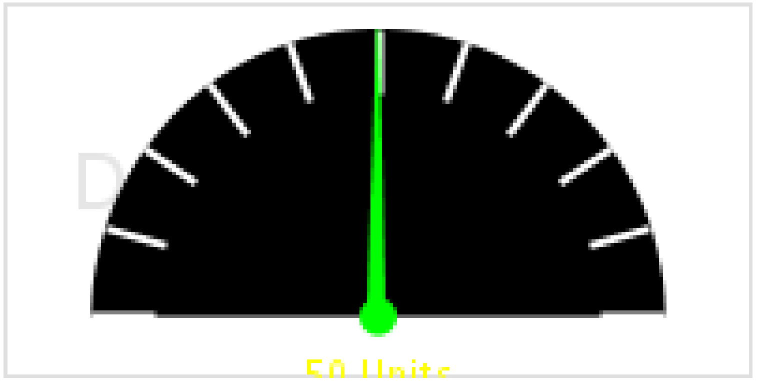
- Many properties are common to all types of charts:
  - *evaluationTime* and *evaluationGroup*: The time when the expressions in the graph are evaluated
  - The title and subtitle (expression, font, color, position)
  - The legend (font, color, position)
  - The rendering type (Vector, image)
  - The orientation of labels
  - ...

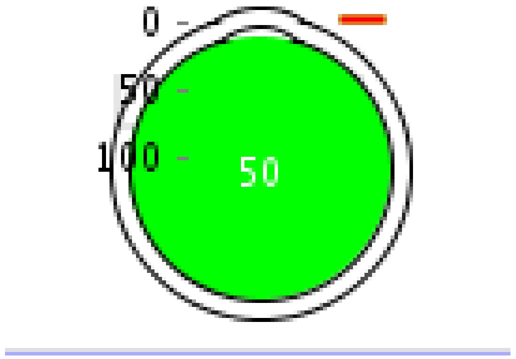
Chart's data

- The data in a chart are very dependent on the type of chart
  - Some may ask to specify the values in abscissa, ordinate, others expressions to evaluate the size of a pie chart,….
- But, in general, there are 2 types of values:
  - 1 to group the data
  - 1 numeric value
- In addition, graphs allow you to define several series of numerical values giving rise to several plots on the same graph.

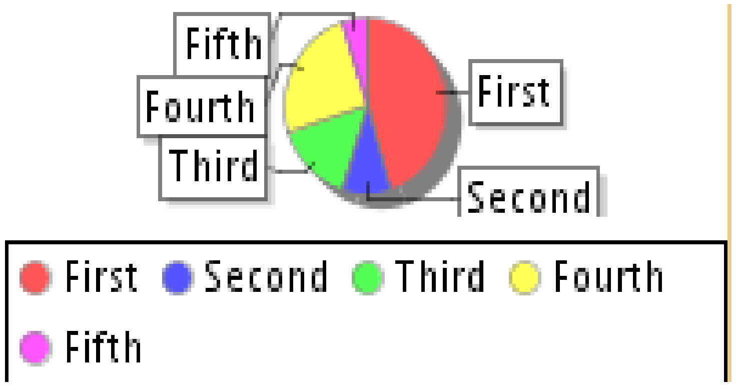
Meter

- "Meter" type charts display only one value (***valueExpression*** attribute)
  - Intervals are used to specify whether this value is judged good, average or bad (Render settings)

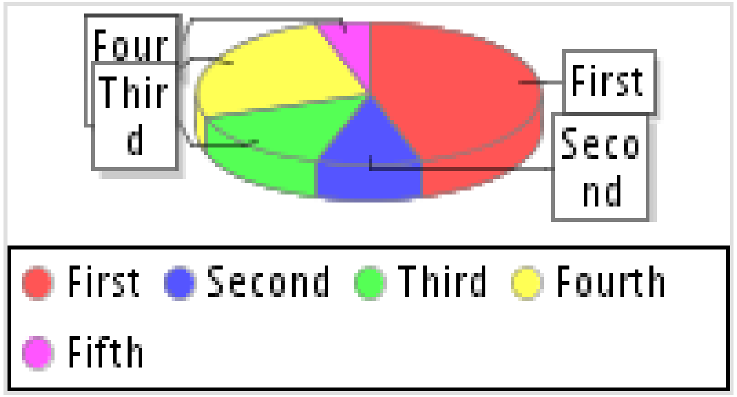




Pies



Fifth
Fourth
Third
First
Second
● First ● Second ● Third ● Fourth
● Fifth

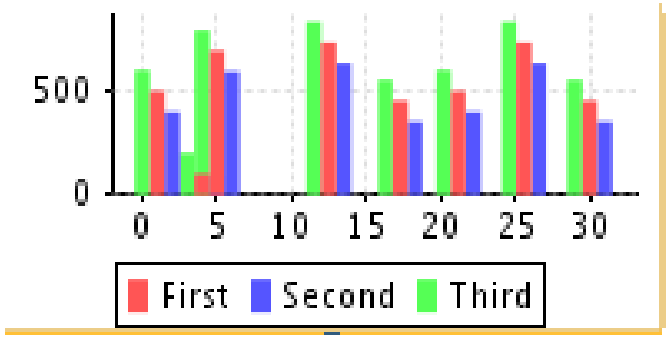


Four
Thir d
First
Seco nd
● First ● Second ● Third ● Fourth
● Fifth

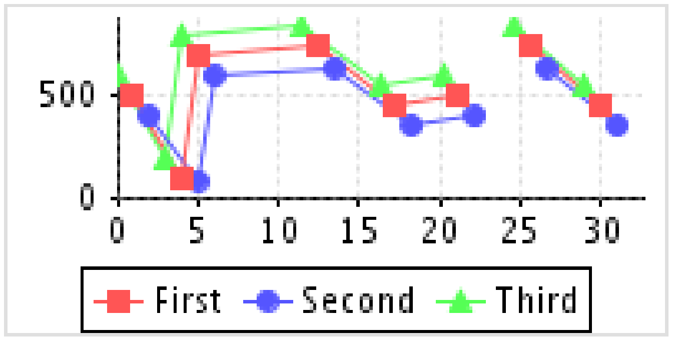
Pie's data

- For *pieDataSet,* you must specify:
  - ***serieExpression :*** Constant or expression used to create several pie charts
  - ***keyExpression*** : An expression that identifies a section of the pir
  - ***valueExpression*** : The size of the section
  - ***labelExpression*** : The label of a section
- It is also possible to define Hypertext links on each section, to limit the number of sections or to indicate a minimum display threshold

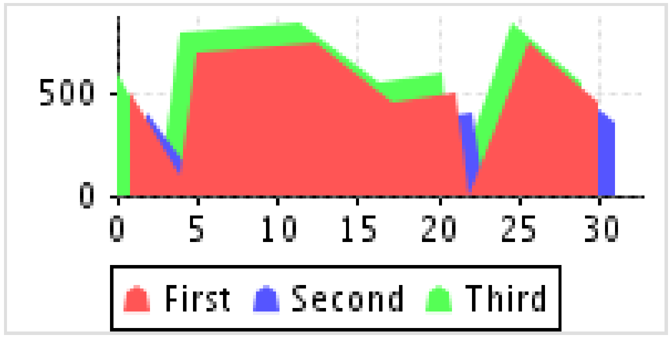
X/Y Charts



500
0
0
5
10
15
20
25
30
🟥 First
🟦 Second
🟩 Third



500
0
0
5
10
15
20
25
30
First
Second
Third



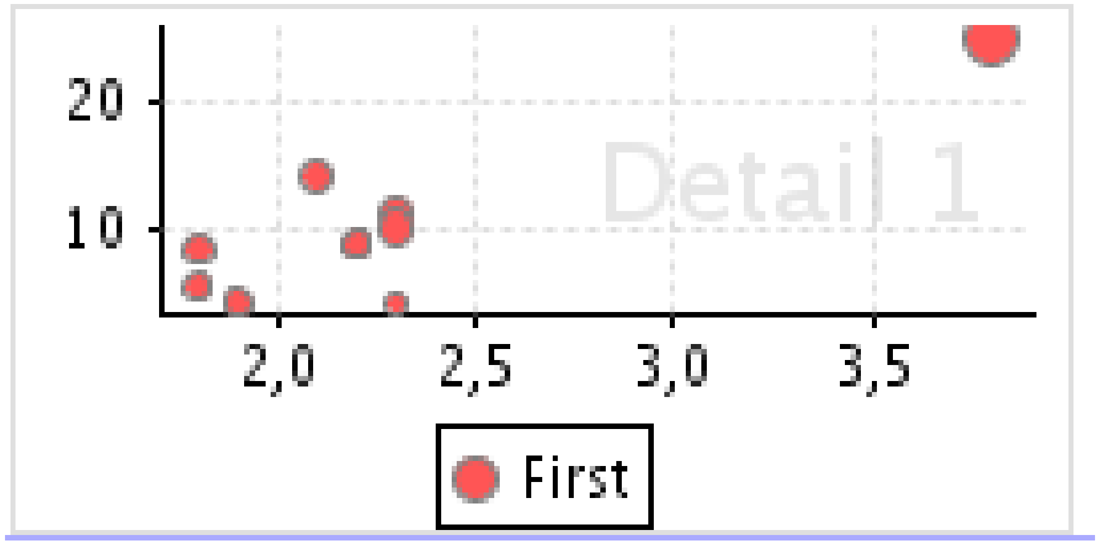
500
0
0
5
10
15
20
25
30
First
Second
Third

XY Charts

- It is possible to draw several series on the same graph (different colors)
- Attributes are :
  - ***seriesExpression*** : The expression allowing to distinguish the series.
    (If only one series, indicate a constant)
  - ***xValueExpression*** : The X axis
  - ***yValueExpression*** : The Y axis

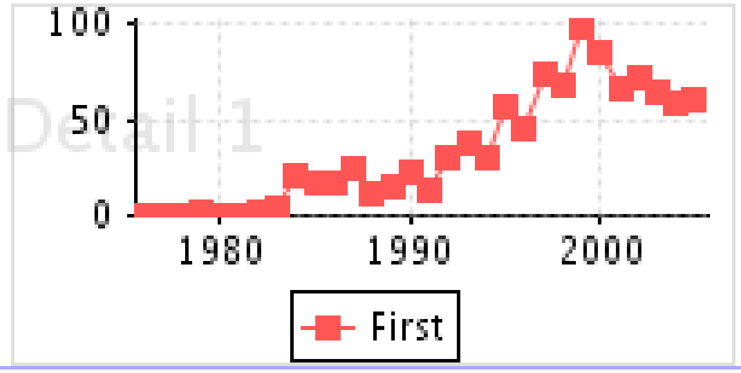
Charts XYZ

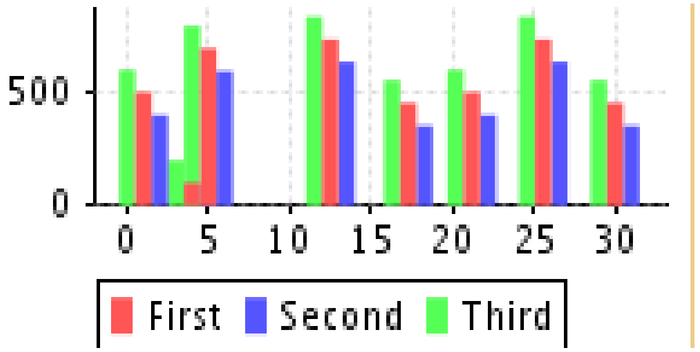
- XYZ Charts add the ***zValueExpression***



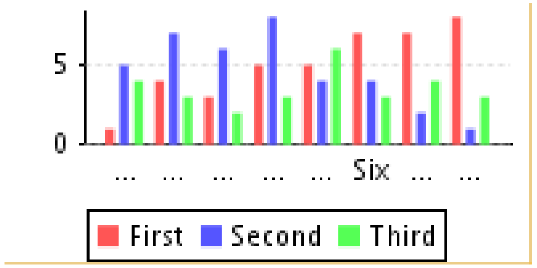
TimeSeries

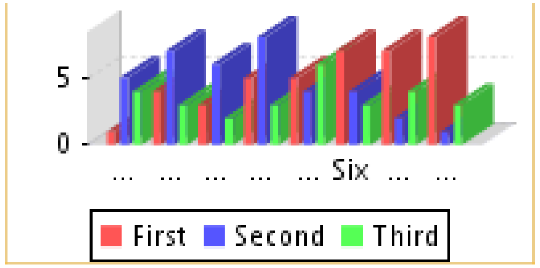
- The x-axis is suited for dates

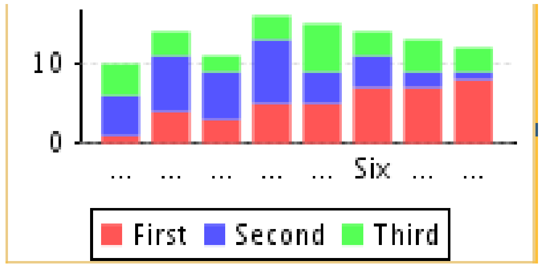


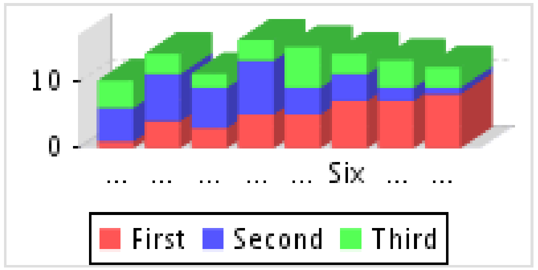


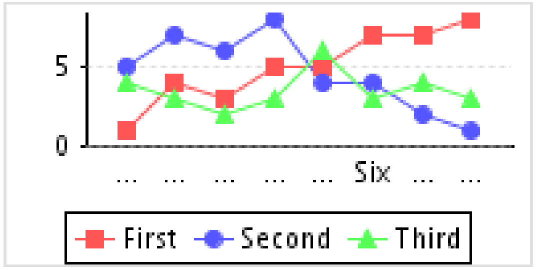
Category

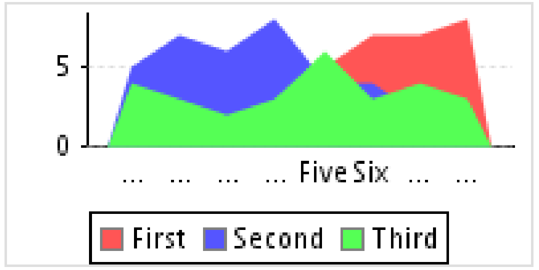

5
0
... ... Six ... ...
First Second Third

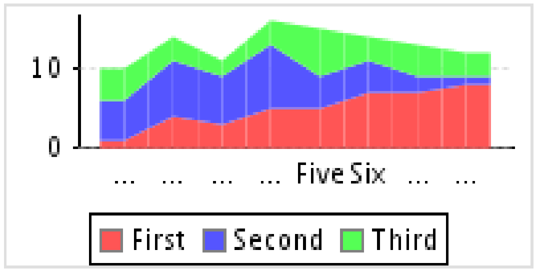

5
0
... ... Six ... ...
First Second Third


10
0
... ... Six ... ...
First Second Third


10
0
... ... Six ... ...
First Second Third


5
0
... ... Six ... ...
First Second Third

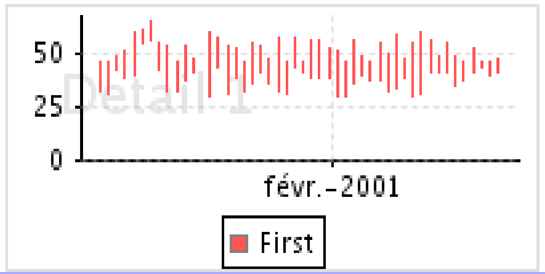

5
0
... ... Five Six ... ...
First Second Third

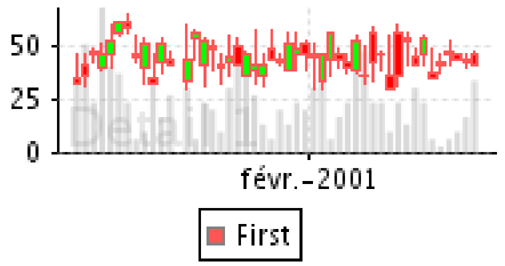

10
0
... ... Five Six ... ...
First Second Third

CategoryDataSet

- The *categoryDataSets* are suitable for comparing different groups.
  The x-axis corresponds to a particular value in a group.
  - ***seriesExpression*** : Used to identify a series (color)
  - ***categoryExpression*** : The groups
  - ***valueExpression*** : The value. (Y-axis)

High/Low charts



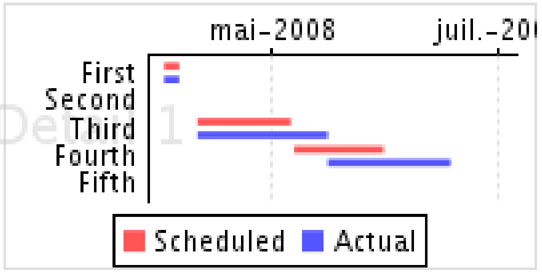


highLowDataSet

- This finance-oriented dataset allows for the same x-axis (a date), to display a high value and a low value, an opening value and a closing value.
- Attributes are :
  - ***seriesExpression*** : Colors
  - ***dateExpression*** : X-axis
  - ***highExpression*** : Highest value
  - ***lowExpression*** : Lowest value
  - ***openExpression*** : Openning value
  - ***closeExpression*** : Closing value
  - ***volumeExpression*** : Volume

GanttDataSet

- Suitable for Gant charts.
- Attributes are :
  - ***TaskExpression*** and ***subTaskExpression*** : For tasks
  - ***startDateExpression*** and ***endDateExpression*** : For planning
  - ***percentExpression*** : For the percentage of completion of a task



Specific drawing attributes

- Specific presentation attributes can be indicated depending on the type of chart:
  - Thermometer: Colors of Mercury
  - Pie 3D: 3D factor
  - Bar, XY Bar, Stacked Bar: Display captions
  - 3D Bar, 3D Stacked Bar: Display of legends + 3D effect (x offset and y offfset)
  - Line, XY Line, Scatter Plot, Time series: Display of lines and shapes
  - Bubble: Scale
  - High Low Open Close: Display colors
  - Candlestick: Volume display
  - ....

Complex elements

Groups
Subreports
Datasets, List and Tables
Charts
Crosstabs

Crosstabs

- **Crosstabs** are specific tables in which the numbers of rows and columns are not known at esign time.
  - For example, sales of different products by years:
    - Neither the number of years nor the number of products are known at design time
- They are used to display aggregated data with multiple levels of column and row groupings
- Common calculations are totals, percentages, averages

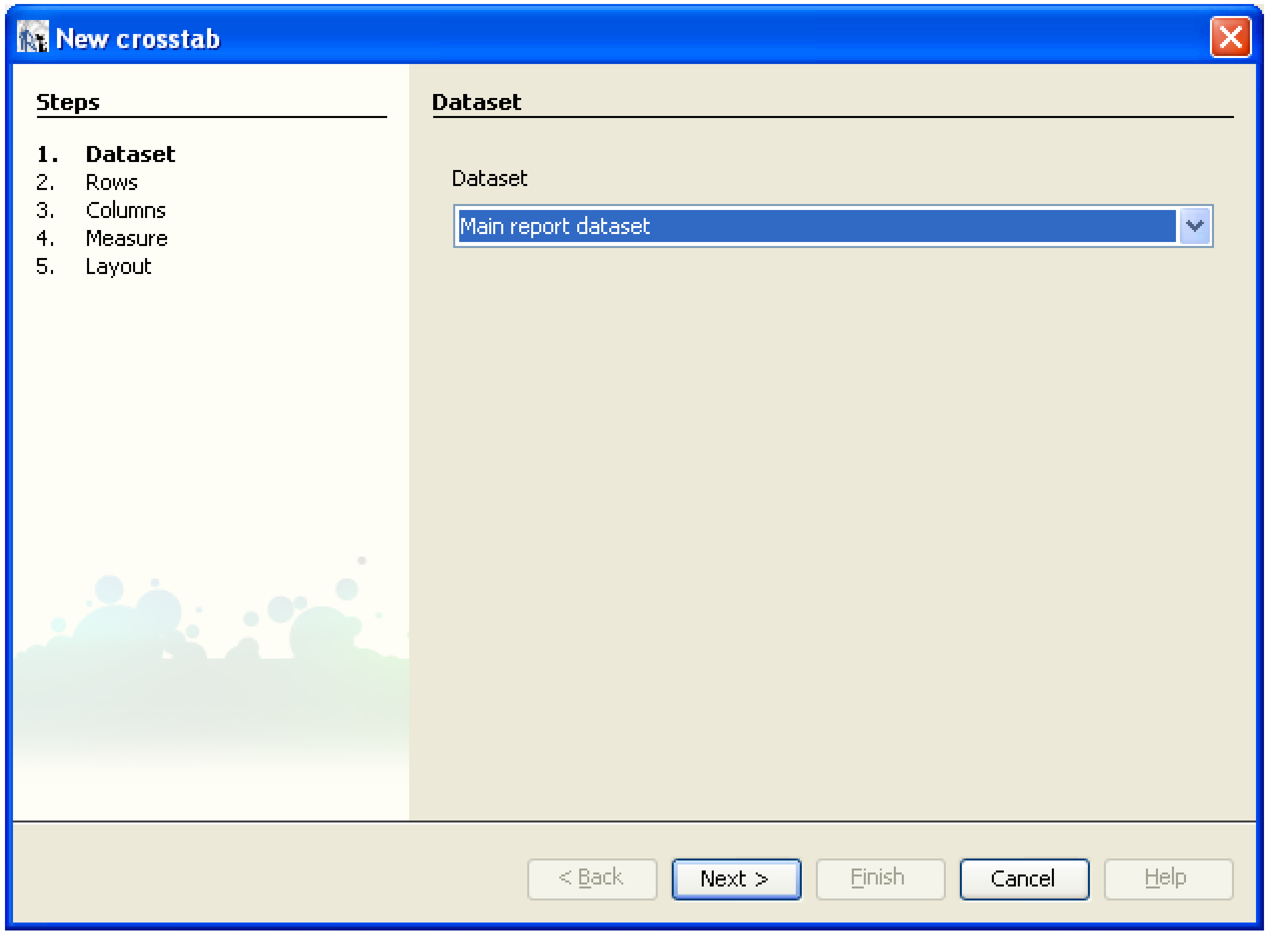
Example

	2007	2008	2009
Fraises	45	21	39
Cerises	40	25	42
Pommes	35	36	38

Wizard

- A wizard with 5 steps ease the creation of a crosstab
  - Definition of the dataset
  - Definition of column groupings
  - Definition of row groupings
  - Definition of the measurement carried out
  - Colors and possible addition of totals at the beginning or end of row / column

Step 1 : Dataset



Steps

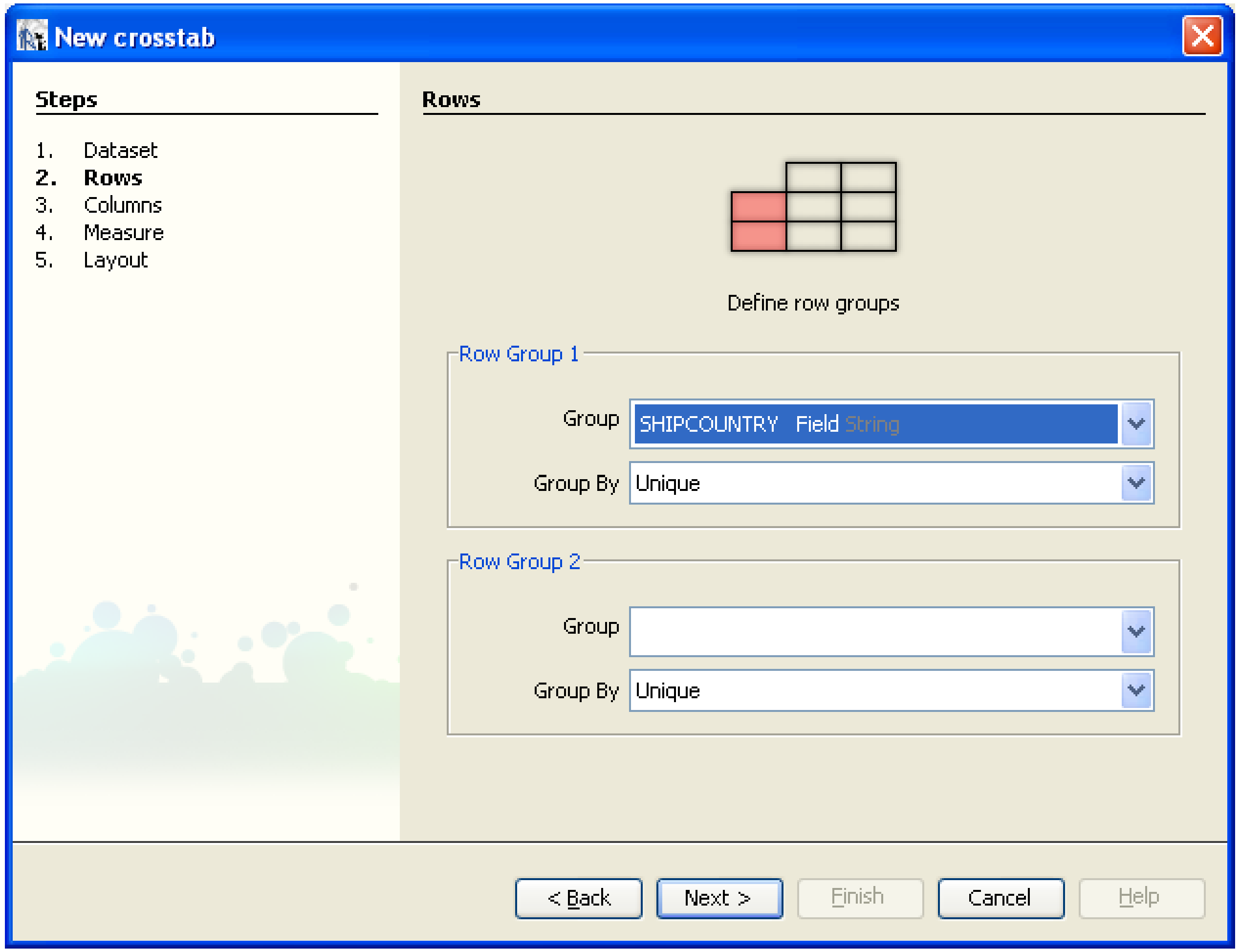
1. **Dataset**
2. Rows
3. Columns
4. Measure
5. Layout

Dataset

Dataset

Main report dataset

Step 2 : Rows


New crosstab

Steps

1. Dataset
2. **Rows**
3. Columns
4. Measure
5. Layout

Rows

Define row groups

Row Group 1

Group: SHIPCOUNTRY Field String

Group By: Unique

Row Group 2

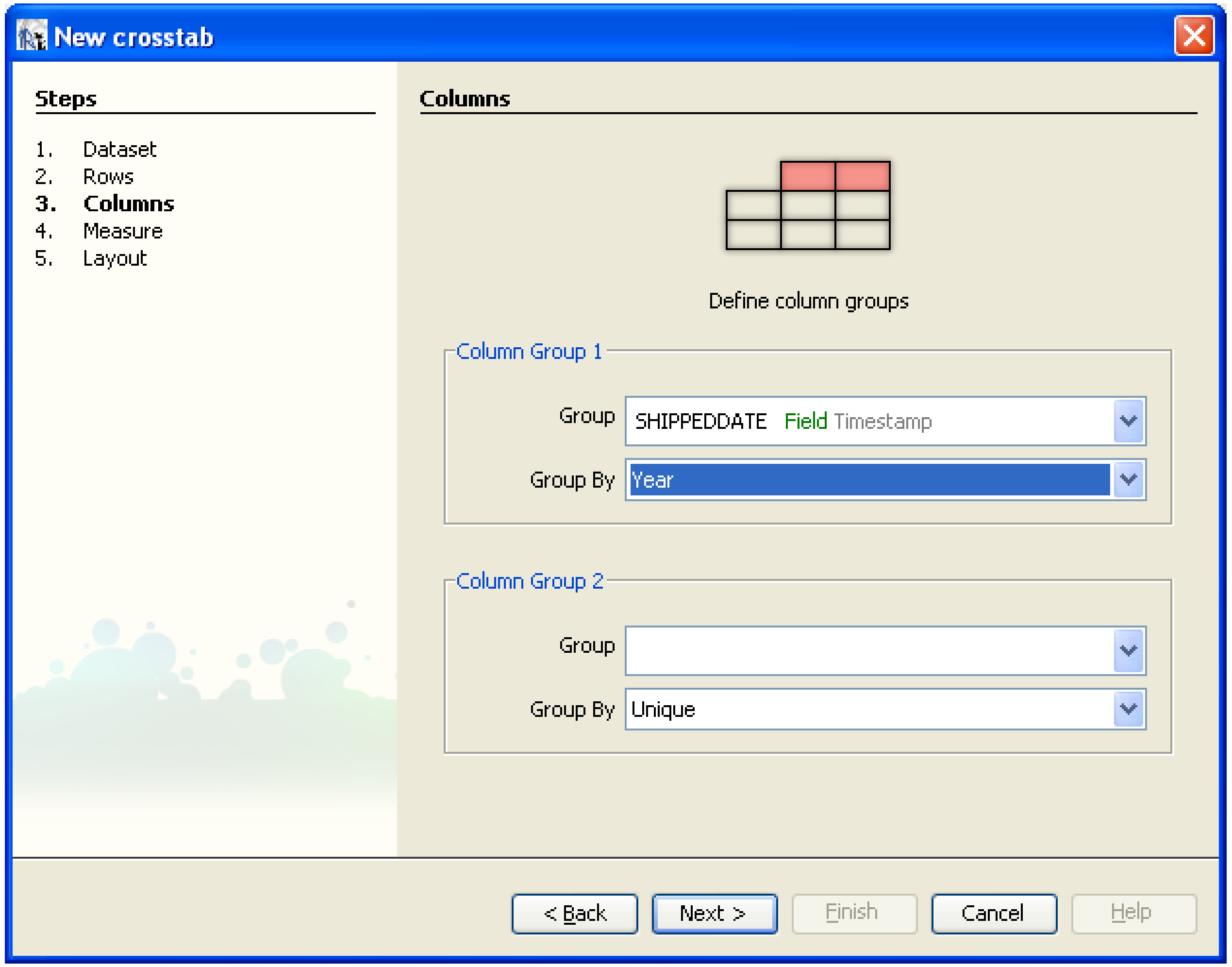
Group:

Group By: Unique

Sorting

- Unlike using groups in the main report, it is not necessary to explicitly sort the dataset according to the grouping criteria
  - This is done automatically and can be turned off if the data is already sorted correctly.

Step 3 : Columns



New crosstab

Steps

1. Dataset
2. Rows
3. **Columns**
4. Measure
5. Layout

Columns

Define column groups

Column Group 1

Group: SHIPPEDDATE Field Timestamp

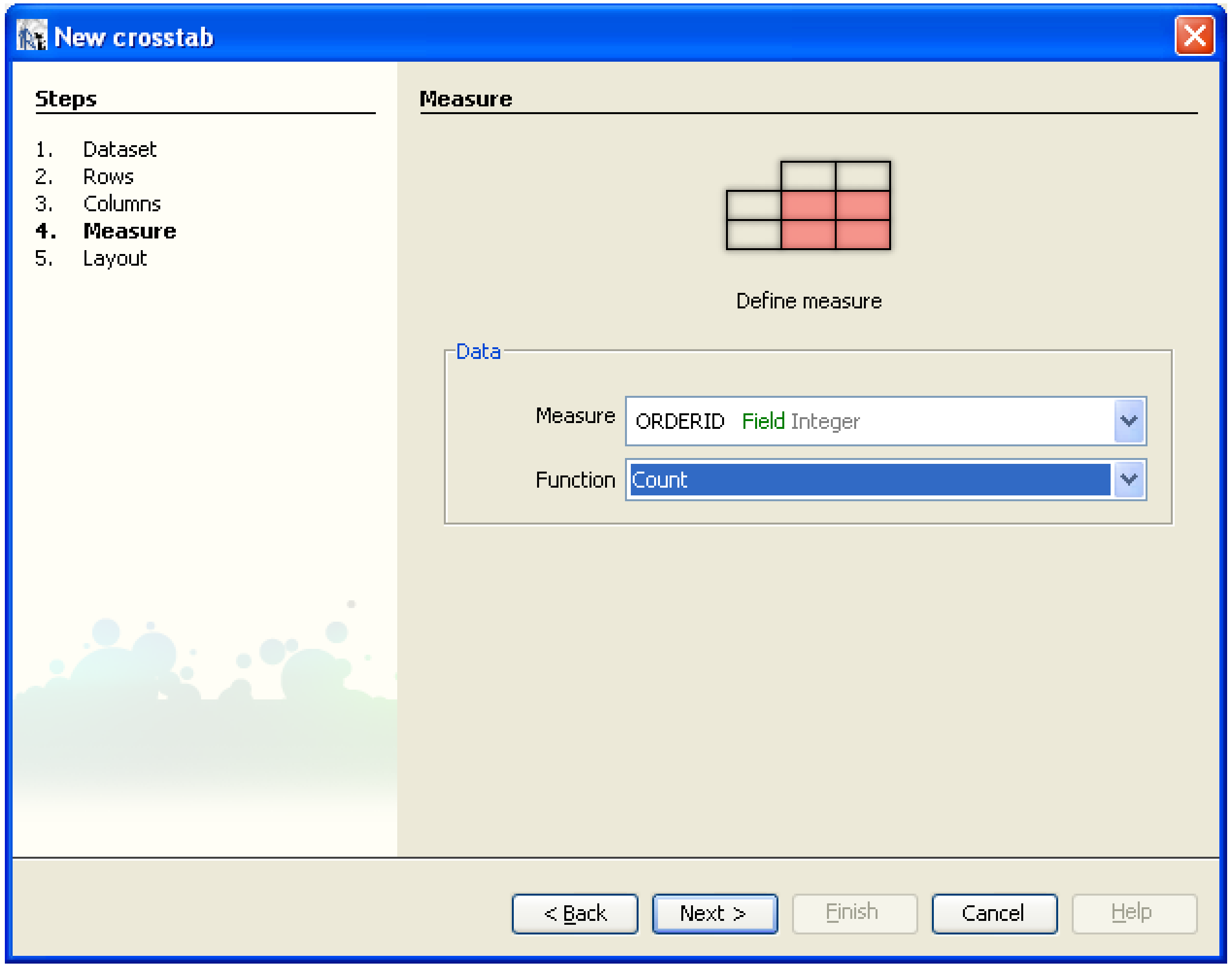
Group By: Year

Column Group 2

Group:

Group By: Unique

Step 4 : Measures



New crosstab

Steps

1. Dataset
2. Rows
3. Columns
4. **Measure**
5. Layout

Measure

Define measure

Data

Measure: ORDERID Field Integer

Function: Count

Measure

A measure is similar to a variable. In general, it is the result of an aggregation function defined by:

- Its **name**
- Its **class**
- Its **expression**
- The **calculation.** If the proposed types of calculation are not sufficient, developers can provide an Incrementer type class via a factory

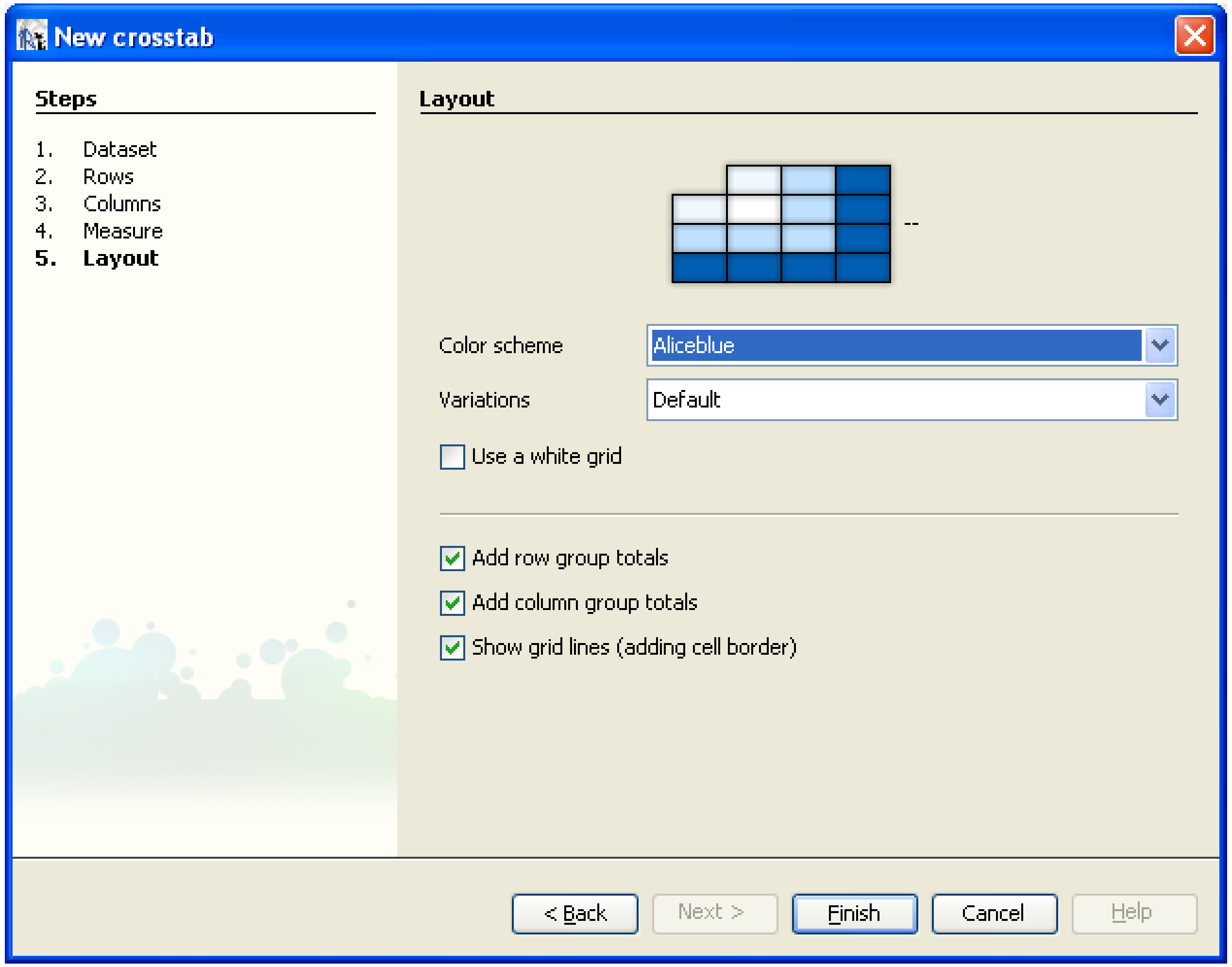
Example :

```
<measure name="ORDERIDMeasure"
  class="java.lang.Integer" calculation="Count">

<measureExpression>
  <![CDATA[$F{ORDERID}]]>
  </measureExpression>

</measure>
```

Step 5 : Layout and Totals



Steps

1. Dataset
2. Rows
3. Columns
4. Measure
5. **Layout**

Layout

Color scheme: Aliceblue

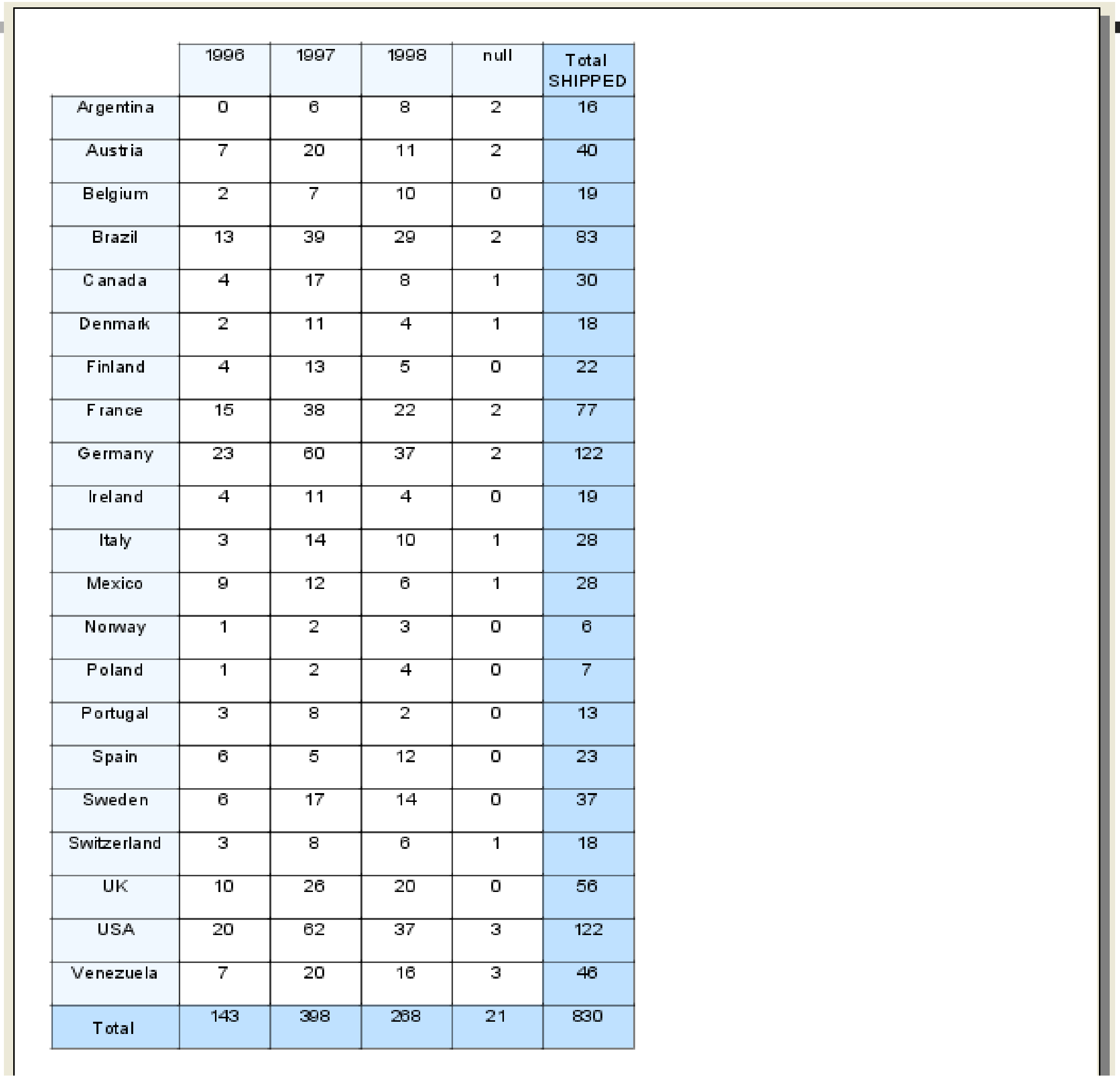
Variations: Default

- [ ] Use a white grid
- [✓] Add row group totals
- [✓] Add column group totals
- [✓] Show grid lines (adding cell border)

Cells

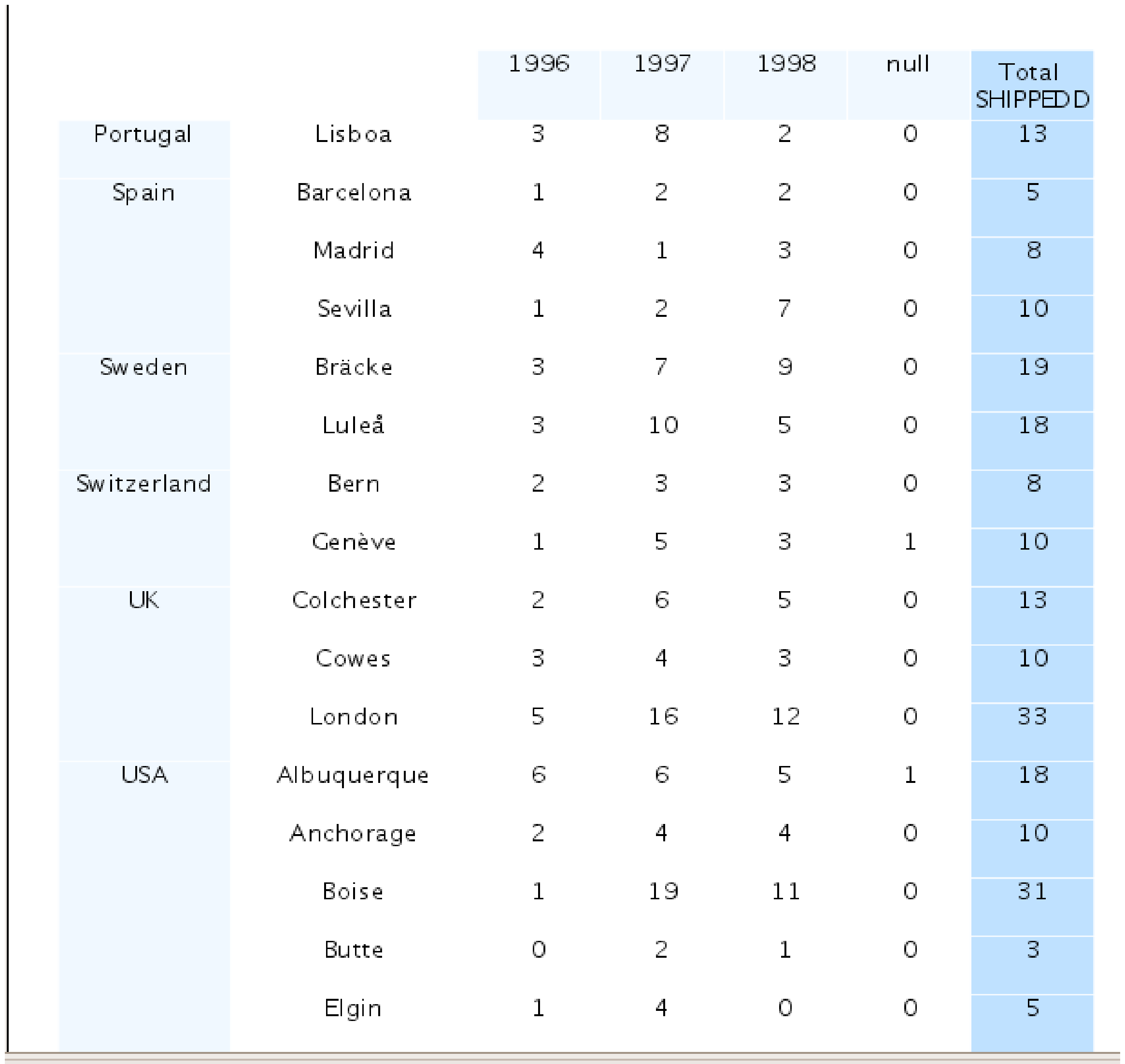
- Depending on the number of grouping of rows and columns; the number of cells in the crosstab varies.
- We can distinguish :
  - Header cells displaying titles or group names
  - Cells showing totals
  - Cells displaying measured values
- From the point of view of jasperStudio, each cell contains elements whose value is defined by expressions (Character strings, variables, ...)

Example



	1996	1997	1998	null	Total SHIPPED
Argentina	0	6	8	2	16
Austria	7	20	11	2	40
Belgium	2	7	10	0	19
Brazil	13	39	29	2	83
Canada	4	17	8	1	30
Denmark	2	11	4	1	18
Finland	4	13	5	0	22
France	15	38	22	2	77
Germany	23	60	37	2	122
Ireland	4	11	4	0	19
Italy	3	14	10	1	28
Mexico	9	12	6	1	28
Norway	1	2	3	0	6
Poland	1	2	4	0	7
Portugal	3	8	2	0	13
Spain	6	5	12	0	23
Sweden	6	17	14	0	37
Switzerland	3	8	6	1	18
UK	10	26	20	0	56
USA	20	62	37	3	122
Venezuela	7	20	16	3	46
Total	143	398	268	21	830

Example 2 rows/1 columns



		1996	1997	1998	null	Total SHIPPEDD
Portugal	Lisboa	3	8	2	0	13
Spain	Barcelona	1	2	2	0	5
	Madrid	4	1	3	0	8
	Sevilla	1	2	7	0	10
Sweden	Bräcke	3	7	9	0	19
	Luleå	3	10	5	0	18
Switzerland	Bern	2	3	3	0	8
	Genève	1	5	3	1	10
UK	Colchester	2	6	5	0	13
	Cowes	3	4	3	0	10
	London	5	16	12	0	33
USA	Albuquerque	6	6	5	1	18
	Anchorage	2	4	4	0	10
	Boise	1	19	11	0	31
	Butte	0	2	1	0	3
	Elgin	1	4	0	0	5

Attributes of a group

- ***totalPosition*** : Defines the presence of a row (or column) for totals.
- ***order*** : The sort order in the group (ascending or descending)
- ***comparatorExpression*** : Returns an instance of *java.util.Comparator* which can be used to sort data

Crosstab attributes

- ***isRepeatColumnHeaders*** : Column headers are repeated during a page change
- ***isRepeatRowHeaders*** : Line headers are repeated during a page change
- ***columnBreakOffset***: The space between two parts of the crosstab, when the crosstab exceeds in width

Crosstab variables

- A crosstab generates a set of variables.
  - Only these variables can be used in cell expressions
  - The variables consist of the aggregations of the measures for all the dimensions of the table.
  - They are useful for creating derived metrics.

Derived measures

- For example, to calculate a percentage, we can divide the measurement by the approved measurement for all dimensions.
- The expression would be:

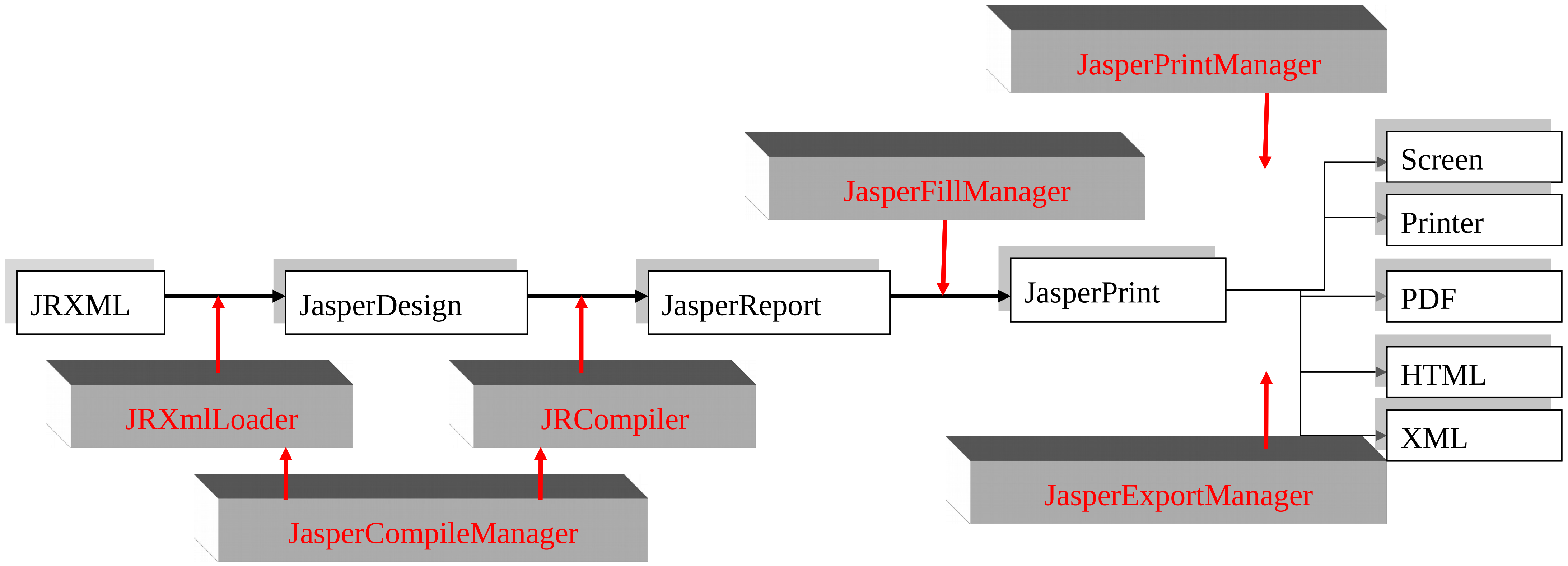
```
new Double(
   $V{ORDERIDMeasure}.doubleValue()
   /
   $V{ORDERIDMeasure_ORDERDATE_ALL.doubleValue() )
```

  – Et Le pattern : #,##0.00 %.

Java and JasperReport

API
Scriptlets
Build tools

Lifecycle of a report



Main classes

The *JasperReport* library exposes a set of facade classes that contain different static methods that simplify the API

These classes are present in the package *net.sf.jasperreports.engine* :

- ***JRXmlLoader :*** Load a JRXML file
- ***JasperCompileManager*** : Compile JRXML
- ***JasperFillManager*** : Fill compiled files with dynamic data
- ***JasperPrintManager*** : Print the final document
- ***JasperExportManager*** : Export a document in PDF, HTML or XML

Loading JRXML

The ***JRXmlLoader*** class allows you to build a JasperDesign object from an XML file

```
static JasperDesign  load(java.io.File file)
static JasperDesign  load(java.io.InputStream is)
static JasperDesign  load(java.lang.String sourceFileName)
```

Example :

```
     JasperDesign design = JRXmlLoader.load("/public/myReport.jrxml");
```

Compilation

- The compilation of the JRXML file is performed by the *compileReport ()* methods of the ***JasperCompileManager*** class.
  - There are different types of compilation depending on whether you use Java, Groovy or Javascript
  - The result is an object of type JasperReport.
  - This object can be serialized to disk to materialize as a *.jasper* file
  - The *.jasper* file can be part of the application deployment
  - *JasperReport* object or *.jasper* file is required for producing the final document

JasperCompileManager Methods

- ***JasperCompileManager*** provides methods:
  - taking as input parameter an *InputStream*, a file location or a *JasperDesign* object
  - Returning an *outputStream*, a *JasperReport* object, or generating a .jasper file as output

```
static JasperReport  compileReport(java.io.InputStream inputStream)
static JasperReport  compileReport(JasperDesign jasperDesign)
static JasperReport  compileReport(java.lang.String sourceFileName)
static void compileReportToFile(JasperDesign jasperDesign, java.lang.String destFileName)
static String  compileReportToFile(java.lang.String sourceFileName)
static void    compileReportToFile(java.lang.String sourceFileName, java.lang.String destFileName)
static void    compileReportToStream(java.io.InputStream inputStream, java.io.OutputStream outputStream)
static void    compileReportToStream(JasperDesign jasperDesign, java.io.OutputStream outputStream)
```

Deployment

In the majority of cases, JRXMLs files do not change when the final application is run, it only dynamically supplies data and parameters to predefined reports.

In this case, the JRXML files can be considered as the source code of the application and their compilation must logically be done during the construction of the application.

The compiled *.jasper* files are then included in the deployed application

Embedded JDT compiler

- In more complex cases, the reports can be modified at execution time. It is therefore recommended to use the JDT compiler for several reasons:
  - It does not need to use temporary files (the JDK based compiler does).
  - It uses the application classloader to resolve classes while the JDK compiler requires the notion of classpath.
- Using the JDT-based compiler requires you to package your jar with ***jdt-compiler.jar***

Data merging

- The ***JasperFillManager*** class
  - Takes as input a file compiled in its different formats, a set of parameters, a data source
  - Return a JasperPrint object, file or OutputStream object that can be exported in different formats

```
static JasperPrint     fillReport(java.io.InputStream inputStream,
                                  java.util.Map parameters,
                                  java.sql.Connection connection)

static JasperPrint     fillReport(JasperReport jasperReport,
                                  java.util.Map parameters,
                                  JRDataSource dataSource)

static JasperPrint     fillReport(java.lang.String sourceFileName,
                                  java.util.Map parameters,
                                  JRDataSource dataSource)
```

- Example

```
JasperFillManager.fillReport(myReport, parametersMap, jdbcConnection);
```

Exporters

- The ***exporters*** classes allow you to export a generated report to a specific document format. They are located in the *net.sf.jasperreports.engine.export* package
- ***JasperExportManager*** class allows exporting to PDF, HTML and XML
- For other formats or for specific needs, other classes are available:
  - *JRHtmlExporter*, *JRPdfExporter* allow fine mapping control from AWT fonts to HTML or PDF fonts
  - *JRXls* Export to Excel,
  - *JRCsvExport* to csv format

JasperExportManager Methods

- The ***JasperExportManager*** class:
  - Takes as input a *JasperPrint* object, an InputStream or a file
  - Generate a file, an OutputStream, a byte array in a visualization format

```
static void   exportReportToHtmlFile(JasperPrint jasperPrint, java.lang.String destFileName)
static byte[] exportReportToPdf(JasperPrint jasperPrint)
static void   exportReportToPdfFile(JasperPrint jasperPrint, java.lang.String destFileName)
static void   exportReportToPdfStream(java.io.InputStream inputStream, java.io.OutputStream
       outputStream)
static java.lang.String  exportReportToXml(JasperPrint jasperPrint)
static java.lang.String  exportReportToXmlFile(java.lang.String sourceFileName, boolean
       isEmbeddingImages)
```

- Example :

```
byte[] pdfContent = JasperExportManager.exportToPdf(myPrint);
```

Viewer

- *JasperReport* provides several ways to directly view a generated report. (JasperPrint)
  - Inside a SWING type application, the *JPanel*: *JRViewer* type component can be used.
  - The stand-alone ***JasperViewer*** application is a Swing application containing the *JRViewer* component

- But, most of the time, the *.jrprint* file is exported in an office format that can be used by the end user

Servlet Example

```
public void service(HttpServletRequest request,HttpServletResponse response)
   throws IOException, ServletException {

response.setContentType("application/pdf");

try {
  JasperReport report =
   JasperCompileManager.compileReport(request.getParameter("jrxml"));

  JasperPrint print =
   JasperFillManager.fillReport(report,request.getParameterMap(),connection);

  JasperExportManager.exportReportToPdfStream(print,
   response.getOutputStream());
} catch (JRException e) {
  e.printStackTrace();
}
```

Java and JasperReport

API

Scriptlets

Build tools

Scriptlets

- **Scriptlets** provide an alternative to expressions when more complex calculations are required.
- Their methods can be used in expressions
- They can provide Java code which will be executed at specific generation time
  - Ex: start of a new page, end of a group

Scriptlets

- A scriptlet is a class that extends ***JRAbstractScriptlet*** or ***JRDefaultScriptlet.***
  The name of the class must be specified in the **scriptletClass attribute** of the <*jasperReport*> element or in <*scriptletClasses*> element.

- A scriptlet may implement some methods called by the engine at specific time during generation:
  - *beforeReportInit (), afterReportInit (), beforePageInit (), afterPageInit (), beforeGroupInit (), afterGroupInit (), etc.*

- It is also possible to retrieve an instance of the scriptlet class using the **REPORT_SCRIPTLET** predefined parameter. We can then as part of an expression called a specific method of the class.

Java and JasperReport

API
Scriptlets
Build tools

Ant

- As compilation is usually done during the build process, an Ant task is provided by JasperReport
- This task implemented by the ***JRAntCompileTask*** class is very similar to the predefined task <javac>
- Defining this task in an ant *build.xml* file is as follows:

```
<taskdef name="jrc"
     classname="net.sf.jasperreports.ant.JRAntCompileTask">
  <classpath>
    <fileset dir="./lib"> <include name="**/*.jar"/> /fileset>
  </classpath>
</taskdef>

<jrc srcDir= "src/jrxml/*" targetDir= "build/jasper">
```

Example

```
<target name="build-report" description="Compile les rapports Jasper">
  <echo message="Building report with ${jasper_home}" />

  <path id="cp">
    <fileset dir="${jasper_home}">
      <include name="**/*.jar" />
    </fileset>
  </path>

  <taskdef name="jrc" classname="net.sf.jasperreports.ant.JRAntCompileTask">
    <classpath refid="cp" />
    <classpath path="${classes.model.dir}" />
  </taskdef>

  <jrc srcdir="./reports" destdir="./reports" tempdir="./classes"
                                               keepjava="true" xmlvalidation="true">
    <classpath refid="cp" />

    <include name="**/*.jrxml" />
  </jrc>
</target>
```

Maven dependencies

Dependencies:

```
<dependency>
  <groupId>net.sf.jasperreports</groupId>
  <artifactId>jasperreports</artifactId>
  <version>6.11.0</version>
</dependency>
```

Optionnaly, if Groovy is used :

```
<dependency>
  <groupId>org.codehaus.groovy</groupId>
  <artifactId>groovy-all</artifactId>
  <version>2.5.8</version>
  <type>pom</type>
</dependency>
```

Plugin Maven

There is a Maven plugin (unofficial) to compile JRXML reports

```
<plugin>
    <groupId>com.alexnederlof</groupId>
    <artifactId>jasperreports-plugin</artifactId>
    <version>2.8</version>
    <executions>
        <execution>
            <phase>compile</phase>
            <goals>
               <goal>jasper</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <sourceDirectory>src/main/resources/</sourceDirectory>
        <outputDirectory>${project.build.directory}/jasper</outputDirectory>
    </configuration>
</plugin>
```

Gradle

```
plugins {
  id "com.github.gmazelier.jasperreports" version "0.4"
}


./gradlew clean compileAllReports
```

https://github.com/gmazelier/gradle-jasperreports

Resources

- “Ultimate Guides” provided by JasperSoft (*JasperReport* et *iReport*)
- “JasperReports for Java Developer ", David R. Heffelfinger, 2006

- Forums et Wikis
- http://community.jaspersoft.com

Thank U!!!

- Thank You for Your Attention

Annex

DataAdapters
Telmplates and wizards
Report Books
Progammatically design a report

Interface JRDataSource

- L'interface *JRDataSource* définit deux méthodes :
  - ***public boolean next()***
    Cette méthode doit retournée *true* si le curseur est positionné correctement sur l'enregistrement suivant. Chaque fois que le moteur invoque cette méthode. Les valeurs des champs sont remplis avec les valeurs de l'enregistrement courant et les expressions sont réévaluées.
    En conséquence, les entêtes d'un nouveau groupe peuvent être imprimées, un saut de page peut s'effectuer, ...
    Si cette méthode retourne *false*, les bas de groupe, bas de colonne, la section de dernière page et le résumé sont imprimés.
  - ***public Object getFieldValue(JRField jrField)***
    Cette méthode retourne la valeur du champ passé en paramètre pour l'enregistrement courant.

Collection de JavaBeans

- Un ensemble de JavaBeans implémente un *JRDataSource*
- Un JavaBean est une classe Java qui expose ses attributs avec des méthodes getter :
  `public <returnType> getXXX()`
  XXX est le nom du champ;
- Si le type de *XXX* est également un JavaBean. Il est possible d'utiliser la notation :
  XXX.YYY
- Le flag *Use Field description* permet d'utiliser la description du champ plutôt que son nom.
- La collection ou le tableau de JavaBeans doit être retourné par une méthode *static* d'une classe fournie (factory).

Exemple JavaBean

```
public class PersonBean {
  private String name = "";
  private int age = 0;

  public PersonBean(String name, int age){
    this.name = name;
    this.age = age;
  }

  public int getAge() {
    return age;
  }
  public String getName(){
    return name;
  }
}
```

Exemple Factory

```
public class SampleFactory {

  public static java.util.Collection generateCollection() {
    java.util.Vector collection = new java.util.Vector();
    collection.add(new PersonBean("Ted", 20) );
    collection.add(new PersonBean("Jack", 34) );
    collection.add(new PersonBean("Bob", 56) );
    collection.add(new PersonBean("Alice",12) );
    collection.add(new PersonBean("Robin",22) );
    collection.add(new PersonBean("Peter",28) );
    return collection;
  }
}
```

XML DataSource

- A la différence du format « table » voulu par Jasper, un document XML est organisé hiérarchiquement en forme d'arbre
- Une expression *XPath* est alors nécessaire pour extraire un ensemble de nœuds du document.
- L'expression *XPath* peut être fournie :
  - dans le rapport; dans ce cas il est possible d'utiliser les paramètres du rapport dans l'expression *XPath*
  - Au moment de la création de la datasource.
- Au moment de la création du datasource, il est également possible de spécifier des patterns Java pour convertir les chaînes de caractères XML en dates ou nombres

Champs XML

- La syntaxe *XPath* est également utilisée pour définir les champs. L'expression étant évaluée à partir du nœud courant. Elle se spécifie dans la description du champ.
- *IReport* propose un outil visuel pour la sélection des champs

Nom	Description	
id	@id	L'attribut id
lastname	lastname	La valeur du noeud-enfant lastname
categoryName	ancestor::category/ @name	L'attribut name du noeud ancêtre category

Documents XML et sous-rapport

- L'organisation hiérarchique des documents est très adaptée à la notion de sous-rapport de JasperReport
- Une *JRXmlDataSource* expose 2 méthodes supplémentaires :
  - *public JRXmlDataSource dataSource(String selectExpression)*
  - *public JRXmlDataSource subDataSource(String selectExpression)*

La première méthode applique une expression *XPath* à partir du nœud racine du document XML. La seconde applique l'expression sur le nœud courant.

L'expression pour spécifier les données d'un sous rapport est alors :

```
((JRXmlDataSource)
$P{REPORT_DATA_SOURCE}).subDataSource(xExpression)
```

CSV DataSource

- Les fichiers CSV sont un format naturel pour une source de données Jasper.
- Les noms des champs sont :
  - Soit obtenus à partir de la première ligne du fichier
  - Soit spécifié un à un
- Il est possible également de spécifier le caractère délimiteur

JREmptyDataSource

- JasperRepoprt fournit une datasource spéciale permettant d'effectuer des itérations sur des enregistrements dont la valeur de champ est *null*
- L'interface iReport permet de spécifier le nombre d'itérations voulues

Hibernate

- Avec Jasper, il est possible d'utiliser une requête HQL.
- La connexion hibernate (*hibernate.cfg.xml*) et les fichiers de mapping (**.hbm.xml*) doivent être dans le classpath.
- La requête est exprimée alors via HQL.
- En fonction du résultat retourné par la requête, les champs du rapport sont alors des entités Hibernate ou des objets simples Java.
- Il est possible de naviguer dans les objets entités avec la notation « . »

Datasources personnalisées

- Fournir une classe implémentant l'interface *JRDataSource*
- Fournir une factory ayant une méthode statique retournant la datasource personnalisée

Gabarits et assistant

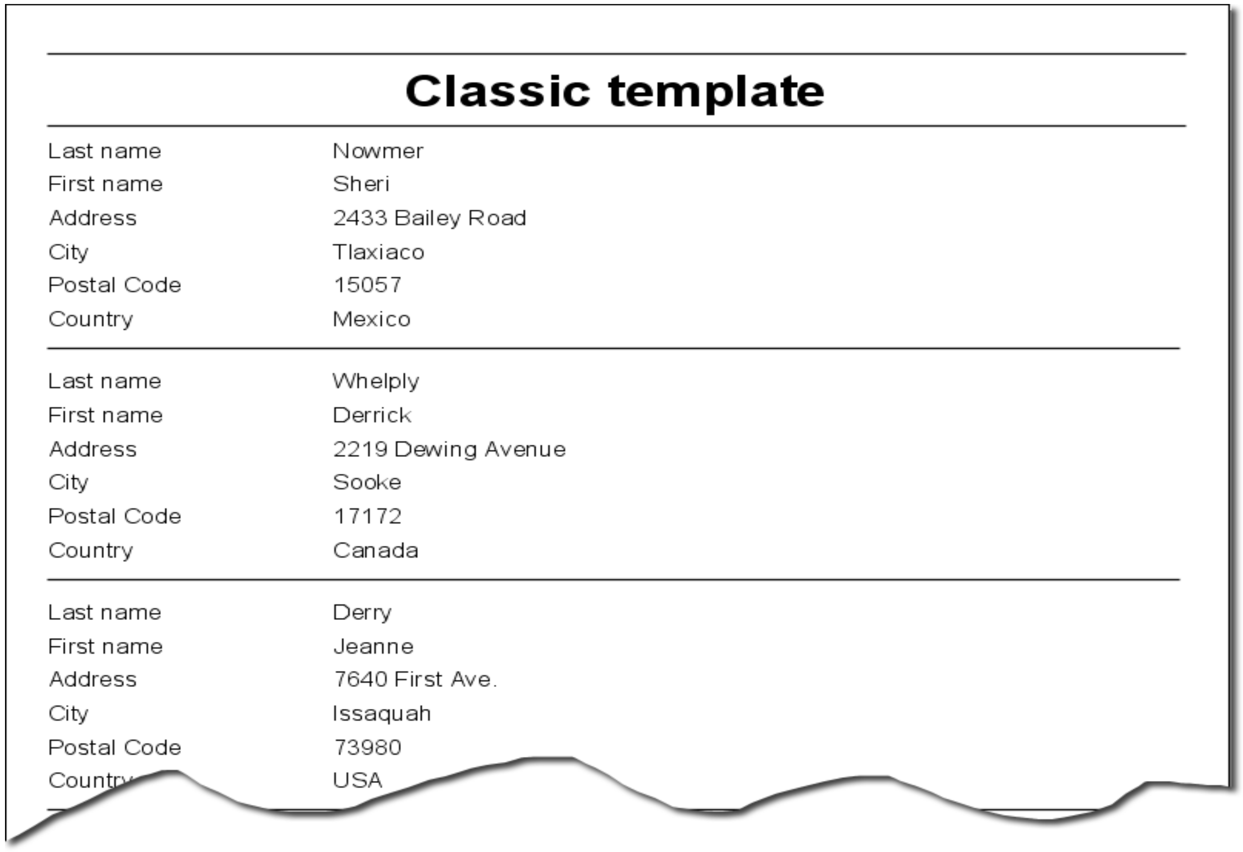
Template

- Avec *jasperStudio*, il est possible de définir des **gabarits de rapport** (template) permettant la réutilisation de disposition et de présentation
  - Ces gabarits peuvent être associés à des assistants pour faciliter alors la création de rapport.
  - *JasperStudio* proposent quelques gabarits dans sa distribution.
- Les fichiers gabarits sont de simples fichiers JRXML stockés dans le répertoire de son choix
  Windows → Preferences → Resource Folder → Template location
- Lorsqu'un nouveau rapport est créé en utilisant un template, l'utilisateur a le choix entre :
  - Démarrer l'assistant associé au template qui lui permettra de renseigner les informations prévues
  - D'utiliser directement le fichier JRXML du template

Assistant

- L'assistant est capable de créer à partir d'une liste de champs sélectionnés par l'utilisateur d'ajouter des éléments textes permettant d'afficher les enregistrements.
- Ces derniers peuvent également être groupés.
- Deux types de disposition sont disponibles :
  - **Colonnes** : Pour chaque enregistrement, on obtient deux colonnes pour le label du champ et sa valeur
  - **Tabulaire** (défaut) : Les enregistrements sont affichés en tableau, en utilisant les column header comme titre de colonne

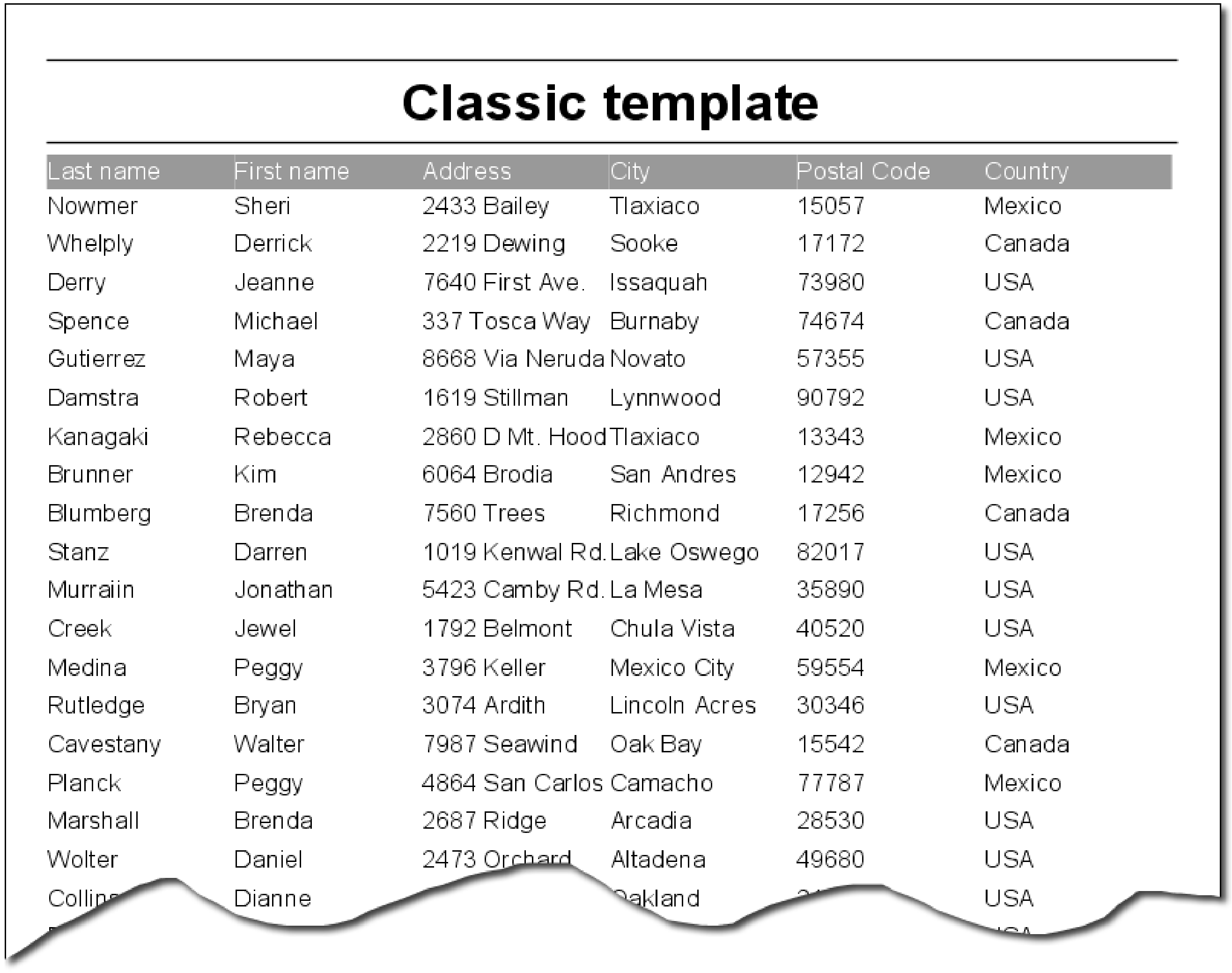
Colonne



Classic template

Last name	Nowmer
First name	Sheri
Address	2433 Bailey Road
City	Tlaxiaco
Postal Code	15057
Country	Mexico
Last name	Whelply
First name	Derrick
Address	2219 Dewing Avenue
City	Sooke
Postal Code	17172
Country	Canada
Last name	Derry
First name	Jeanne
Address	7640 First Ave.
City	Issaquah
Postal Code	73980
Country	USA

Tableau



Classic template

Last name	First name	Address	City	Postal Code	Country
Nowmer	Sheri	2433 Bailey	Tlaxiaco	15057	Mexico
Whelply	Derrick	2219 Dewing	Sooke	17172	Canada
Derry	Jeanne	7640 First Ave.	Issaquah	73980	USA
Spence	Michael	337 Tosca Way	Burnaby	74674	Canada
Gutierrez	Maya	8668 Via Neruda	Novato	57355	USA
Damstra	Robert	1619 Stillman	Lynnwood	90792	USA
Kanagaki	Rebecca	2860 D Mt. Hood	Tlaxiaco	13343	Mexico
Brunner	Kim	6064 Brodia	San Andres	12942	Mexico
Blumberg	Brenda	7560 Trees	Richmond	17256	Canada
Stanz	Darren	1019 Kenwal Rd.	Lake Oswego	82017	USA
Murraiin	Jonathan	5423 Camby Rd.	La Mesa	35890	USA
Creek	Jewel	1792 Belmont	Chula Vista	40520	USA
Medina	Peggy	3796 Keller	Mexico City	59554	Mexico
Rutledge	Bryan	3074 Ardith	Lincoln Acres	30346	USA
Cavestany	Walter	7987 Seawind	Oak Bay	15542	Canada
Planck	Peggy	4864 San Carlos	Camacho	77787	Mexico
Marshall	Brenda	2687 Ridge	Arcadia	28530	USA
Wolter	Daniel	2473 Orchard	Altadena	49680	USA
Collin	Dianne		Oakland		USA

Groupes

- L'utilisateur peut choisir de grouper les données. (4 groupes au maximum)

  => Les entêtes et bas de groupe sont alors créés pour chaque groupe

  => Pour chaque groupe, l'assistant positionne l'expression et ajoute un label et un champ texte permettant d'afficher la valeur.

- L'expression est obligatoirement réduite à un nom de champ

Mise au point

- Pour mettre un point un gabarit compatible avec l'assistant. Il est nécessaire de respecter certaines **règles de nommage**.
- Le gabarit définit à priori :
  - Le maximum de groupe possible (4)
  - L'entête de colonne
  - La bande de détail
- Ces bandes sont analysées par l'assistant afin de remplacer ou de créer certains éléments textes.
- Les autres bandes ne sont pas modifiées par l'assistant

Groupes

- Les éléments des entêtes de groupes sont analysés et si une de leur valeur correspond à une valeur prédéfinie, elle est remplacée soit par le nom du critère de groupe (Textes statiques) soit par la valeur de l'expression (Champs textes)
  - Textes statiques : *GroupLabel, Group Label, Label, Group name, GnLabel* (avec n le n° de groupe)
  - Champs textes : *GroupField, Group Field, Field, GnField*

Colonne

- L'entête de colonne est analysée par l'assistant lors d'une disposition en table.
- Si il trouve des textes statiques contenant des chaînes prédéfinies, l'assistant créera un label pour chaque champ.
- Les valeurs prédéfinies étant :
  - *DetailLabel*, *Label* ou *Header*

Détail

- Si le rapport à une disposition en table, l'assistant analysera la section détail afin de rechercher les champs textes contenant des valeurs prédéfinies.
- Il créera alors un champs texte pour chaque champs des enregistrements. Les valeurs prédéfinies étant :
  - *“DetailField”*, *“Field”*
- Si le rapport est en colonne, l'assistant recherchera dans la bande détail, les champs statiques avec les mêmes critères que l'entête de colonne.

Type de disposition

- Pour choisir entre la disposition colonne et la disposition table, il suffit de spécifier dans le gabarit la propriété ***template.type*** qui peut prendre 2 valeurs :
  - *tabular*
  - *columnar*

Exemple *cherry.jrxml*

```
<jasperReport  name="Cherry" language="groovy" pageWidth="595" pageHeight="842" columnWidth="535"
 leftMargin="20" rightMargin="20" topMargin="20" bottomMargin="20">
<style name="Row" mode="Transparent" fontName="Times New Roman" pdfFontName="Times-Roman">
<conditionalStyle><conditionExpression><![CDATA[$V{REPORT_COUNT}%2 == 0]]></conditionExpression>
<style style="Row" mode="Opaque" backcolor="#F0EFEF"/>
</conditionalStyle></style>
<group name="Group1">
<groupExpression><![CDATA[(int)($V{REPORT_COUNT}/15)]]></groupExpression>
<groupHeader>
<band height="37">
<frame><reportElement mode="Opaque" x="0" y="7" width="555" height="24" forecolor="#B89F7D"
 backcolor="#000000"/>
<textField isStretchWithOverflow="true">
<reportElement style="SubTitle" isPrintRepeatedValues="false" x="2" y="0" width="479" height="24"
 forecolor="#FFFFFF"/>
<textFieldExpression class="java.lang.String"><![CDATA["GroupField"]]></textFieldExpression>
</textField>
</frame>
</band>
</groupHeader>
...
```

Report Books

Introduction

- Un **report book** est un unique fichier *.jrxml* qui englobe plusieurs rapports dans une unique objet.
- Le livre a ses paramètres, ses variables, sa requête et ses champs
- Le livre introduit la notion de ***master*** qui correspond au niveau livre :
  - MASTER_PAGE : Le numéro de la « partie » du livre
  - Réinitialisation MASTER : Réinitialisation d’une variable à un changement de « partie »

Apports

Les report book permettent de :

- Générer un sommaire avec les différentes signets des parties du livre (i.e les sous-rapports associés)
- Paginiation spécifique incluant le n° de la partie

Création programmatique de rapport

Introduction

- Dans certains cas, l'application Java finale peut vouloir mettre au point un design Jasper à la volée.
- L'API de JasperReport permet alors toutes les opérations sur une instance de *JasperDesign* :
  – Création
  – Ajout de champs, variables, groupes
  – Manipulation de bandes, sections
  – Ajouts d'éléments

Cas d'utilisation

- Il est, cependant fastidieux de créer un *JasperDesign* complet à partir de zéro.
- En général, on part d'un fichier JRXML incomplet que l'on complète avec les informations récoltées de l'interface utilisateur. Par exemple, la requête les champs, les groupes, etc..

Interface et classe d’implémentation

- L’API fournit deux types d’objets Java
  - Des **interfaces** qui permettent d’accèder en lecture aux éléments du rapport. Exemple, *JRGroup, JRBand, JRField*
  - Des **implémentations** qui peuvent elles être instanciées et modifiées Exemple : *JRDesignGroup*, *JRDesignBand*, *JRDesignField*
- Les méthodes disponibles renvoient des interfaces qu’il est nécessaire de *caster* afin de pouvoir les manipuler

Hiérarchie

- La classe *JasperDesign* hérite de *JRBaseReport*.
  - ***JRBaseReport*** encapsule toutes les données concernant la structure du rapport (*JRSection* et *JRBand*)
  - ***JasperDesign*** complète la classe parente avec tous les concepts extérieurs à la structure : champs, variables, groupes, scriptlets, styles, ...

JRBaseReport

- *JRBaseReport* permet d’accéder aux éléments structurels du rapport :
  - ***JRSection*** : Une section peut contenir plusieurs bandes. Il existe trois types de sections dans un rapport
    - La section détail : *JRSection getDetailSection()*
    - Les sections relatives aux groupe : *groupHeaderSection* et *groupFooterSection* obtenues à partir de la classe *JGroup*
  - ***JRBand*** : Une bande contenant des éléments
    - *JRBand getTitle();*
    - *JRBand getBackground();*
    - *JRBand getColumnHeader() , getColumnFooter()*
    - *JRBand getPageHeader(), getPageFooter(), getLastPageFooter()*
    - *JRBand getSummary()*
    - *JRBand getNoData()*

JRDesignBand

- A partir d’une instance de *JRDesignBand*, il est possible d’ajouter des éléments dans une bande du rapport via les méthodes :
  - *addElement(JRDesignElement element)*
  - *addElementGroup(JRDesignElementGroup group)*
- ***JRDesignElement*** est un élément simple qui ne contient pas d’autre élément
- ***JRDesignElementGroup*** est un container d’éléments (Ex *JRDesignBand*)

JRDesignElement

- Les classes implémentant *JRDesignElement* correspondent à tous les éléments qu'il est possible d'ajouter dans une bande :
  - *JRDesignTextElement, JRDesignCrosstab, JRDesignGraphicElement, JRDesignChart, JRDesignFrame, JRDesignSubreport, …*
- Sur chaque élément, il est possible d'accéder à ses propriétés spécifiques (police, radius, dataset, etc.)

Méthodes de lecture JasperDesign

- En dehors de la structure, des méthodes de type *getter* permettent de récupérer toutes les autres propriétés du design :
  - Toutes les expressions utilisées :
    - `Collection getExpressions(), JRExpression getFilterExpression()`
  - Les champs :
    - `List getFieldsList(), Map getFieldsMap()`
  - Les paramètres :
    - `List getParametersList(), Map getParametersMap()`
  - Les variables :
    - `List getVariablesList(), Map getVariablesMap()`
  - Les groupes :
    - `List getGroupsList(), Map getGroupsMap()`

Méthodes de lecture JasperDesign

- Les scriptlets :
  - `List getScriptletsList(), Map getScriptletsMap()`
- Les styles
  - `List getStylesList(), Map getStylesMap()`
- Les tableaux croisés :
  - `List getCrosstabs()`
- Les datasets :
  - `JRDesignDataset getMainDesignDataset(), Map getDatasetMap(), JRDataset[] getDatasets(), List getDatasetsList()`

Méthode d'ajout d'éléments

- Méthodes d’ajouts :
  - `void addField(JRField field)`
  - `void addParameter(JRParameter parameter)`
  - `void addGroup(JRDesignGroup group)`
  - `void addVariable(JRDesignVariable variable)`
  - `void addScriptlet(JRScriptlet scriptlet)`
  - `void addStyle(JRStyle style)`
- Les méthodes *remove* équivalentes sont bien évidemment accessibles
- La méthode pour spécifier la requête : `setQuery(JRDesignQuery query)`

Expressions

- En dehors de leurs attributs spécifiques, les éléments ont en général besoin de définir leur expression et la classe associée.
- La classe peut être précisée soit par une *String* (*classname*), soit directement par une instance de *java.lang.Class*
- L’expression est elle fournie par une instance de ***JRExpression***

Composition d’une expression

- Une expression est découpée en morceau (***JRChunkExpression***)
- Chaque morceau étant constitué d’une chaîne de caractère et d’un type.
- Les types possibles étant :
  - *JRExpressionChunk.TYPE_TEXT*
  - *JRExpressionChunk.TYPE_PARAMETER*
  - *JRExpressionChunk.TYPE_FIELD*
  - *JRExpressionChunk.TYPE_VARIABLE*
  - *JRExpressionChunk.TYPE_RESOURCE*