

FJReport Tutorial

Version 0.4.1

1. What is FJReport

FJReport is named for Free Java Report.

FJReport is originally designed for desktop application. When I was writing an application something like Management Information System (MIS), I found that my customers need a "What You See Is What You Get" (WYSIWYG) editor to finish grid-style report and submit. Also, there are many circumstances that end users require a WYSIWYG editor to fill report and programmers need a fast report tool to design grid-style report. I can't find any free java package for such requirements. So I have to write a package myself.

FJReport is a light-weight WYSIWYG report package both for form-style report designing and filling. Unlike most other report designers which draw grids by cell melting, FJReport auto calculates out cells from lines. User draws lines directly on a page and let FJReport determine cells. Light-weight means that FJReport is a small package with basic functions provided. For easy deployment, it's not intent to include other library.

2. Basic Functions

Swing based report with graphic report template designer.

Paged table printing facility.

Some other useful swing components.

3. Make First Report

Download the release package of fjreport.0.4.1.jar.

To run the template designer,

```
java -jar fjreport.0.4.1.jar
```

It assume that installation of the java command is in your path. If it isn't, then you should either specify the complete path to the java command or update your path environment variable as described in the installation instructions for the Java 2 SDK.

The report has 4 states, which are line state, cell state, edit state, readonly state.

DRAWLINE_STATE.

Draw vertical or horizontal lines.

CELL_STATE.

Set cell's properties. Left double click on a cell to activate the cell edit dialog.

There are currently 9 cell types, empty, label, textfield, textarea, combobox, checkbox, datefiled, image, custom

When report state is switched from line to cell, it will update cells' position and

size according to the lines change. The algorithm performance is about $n*n$, fast enough.

EDIT_STATE.

It is for end user of your application.

User can finish form-style report in edit state. Edit components are embedded into cells to accept users' input. The class of edit component varies from JTextField to JPanel, according to cell's type.

What they see on the screen is what they get on a paper from printer. This is the main purpose of this package, to provide form-style report editor to user, something like MS Word.

READONLY_STATE

It looks like edit state, but can't accept user input. You can use it to show preview dialog.

In the east of the main content pane of template designer, there is a vertical toolbar containing some functional button. Click the line button to get into DRAWLINE_STATE,

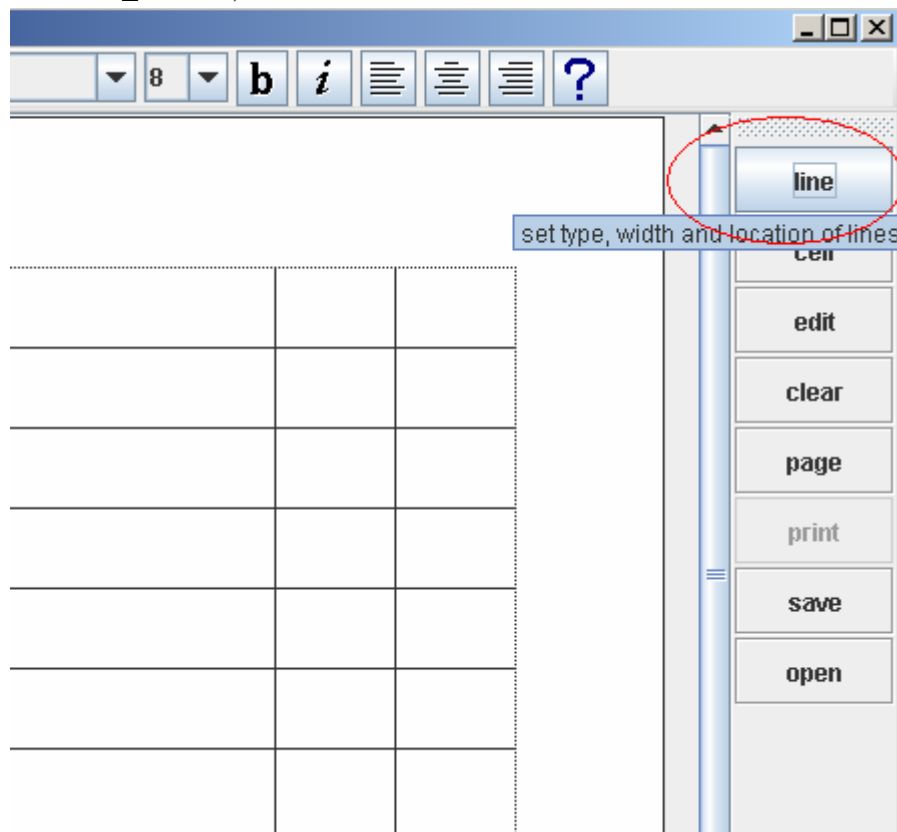


Figure 1. Line State of Template Designer

Right press mouse and drag, when right button is released, a new line will appear. Left click on a line to pick it. Left double click on it to set line properties. When a line is picked (high lighted with red color), left press mouse on it and

drag can move the line to the desired position. Left press mouse on its end point and drag (to vertical line, bottom point is end; to horizontal line, right point is end), the line length will change. Right click on it will remove the line.

Left double click on blank will create a horizontal line stretching from left border to right border with `line.startPoint.y = mouseEvent.y`, while right click on blank will create a vertical line stretching from top to bottom.

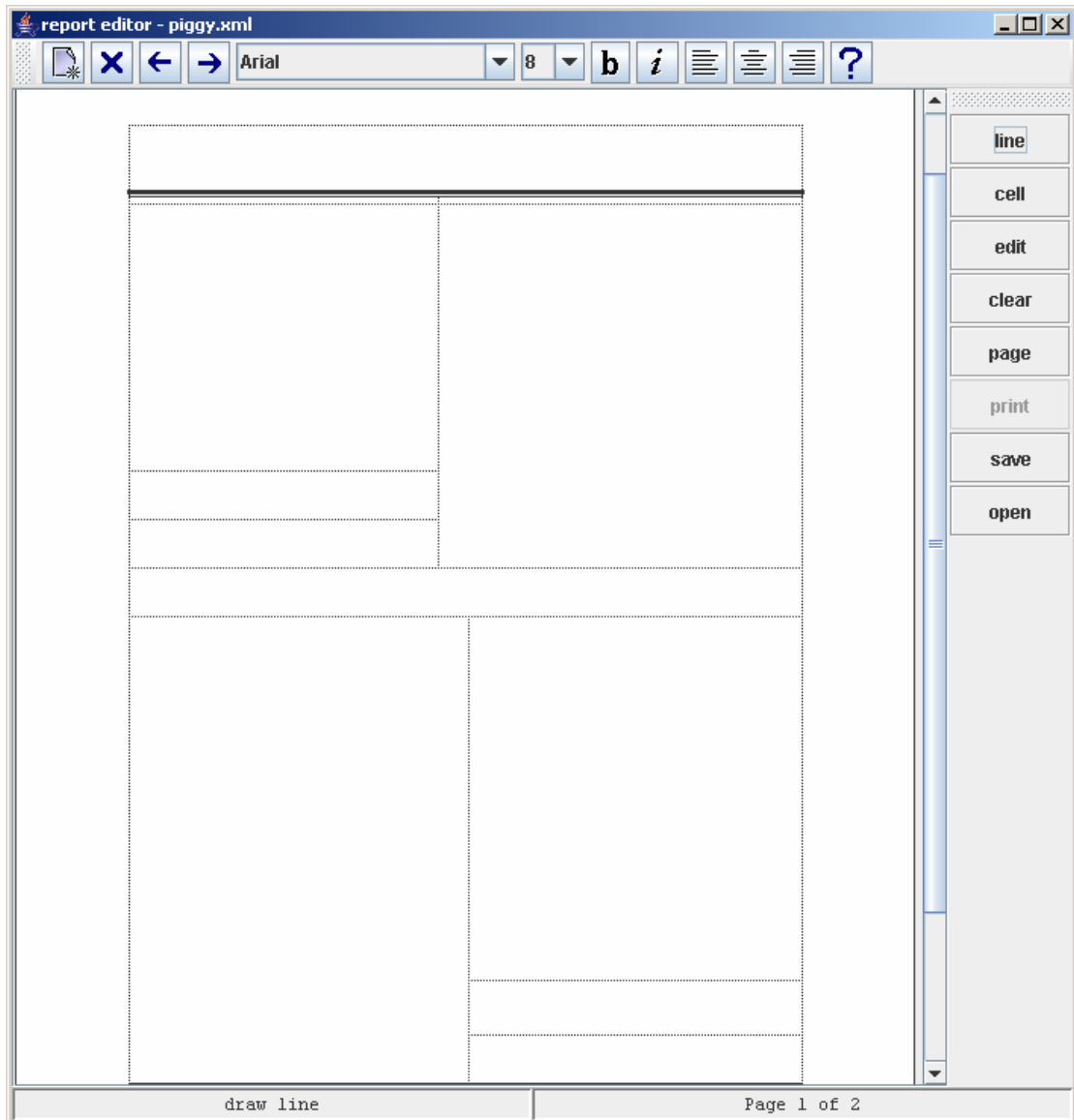


Figure 2. Piggy Report – Line State

Our first report, piggy report, would look like figure 2. You can also find `piggy.xml` at `net.sf.fjreport.samples.piggy.xml` in the source code package.

Click the Cell button to go to `CELL_STATE`.

Left double click on a cell to activate the cell property setting dialog.

Each cell except `TYPE_NULL` cell is identified by a string which is the cell's name. Each null cell has a value (mostly is instance of `String`) which is shown to end user. You can omit the Label cell's name, if you don't want to change the value of label string later in the programming.

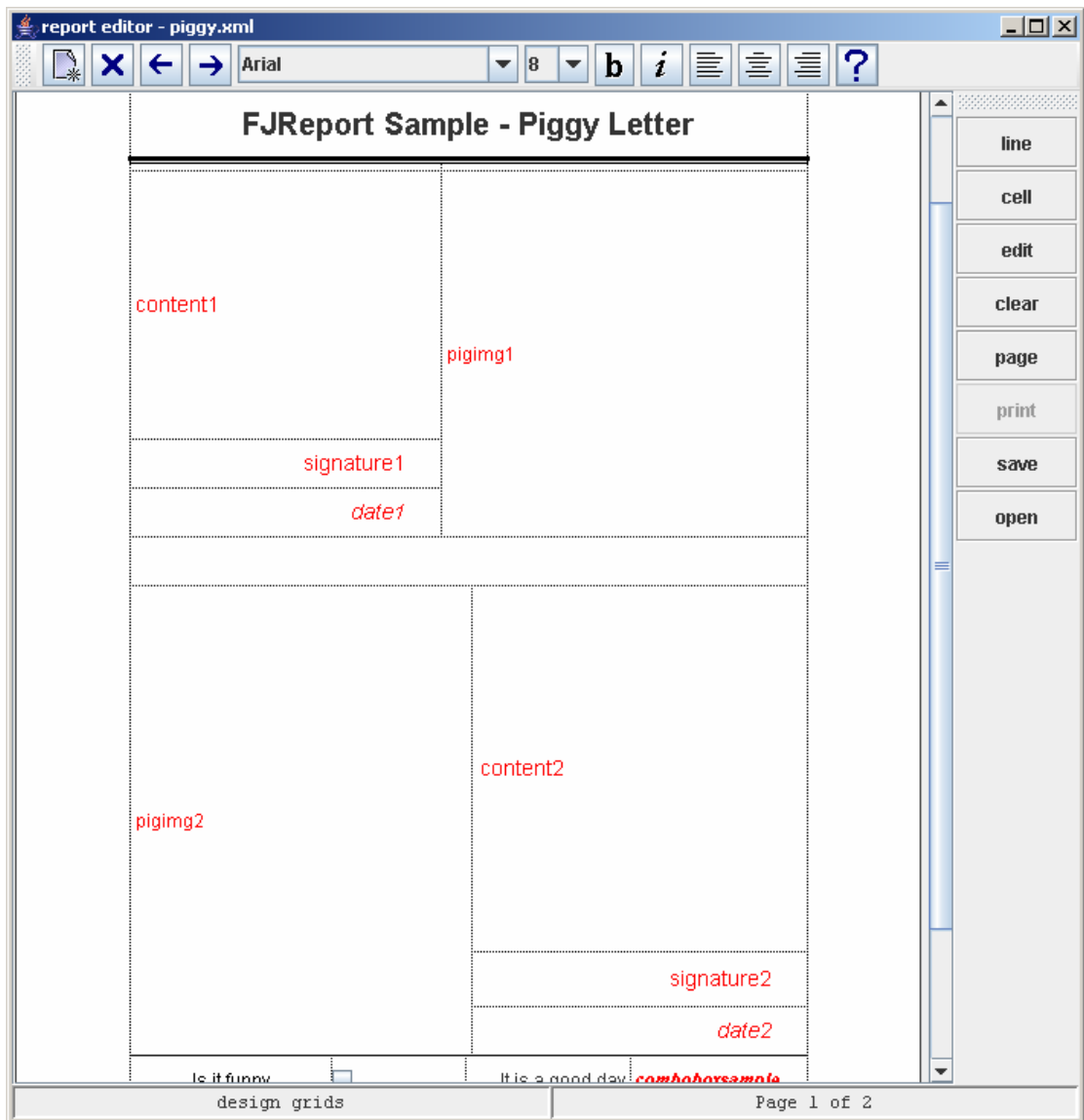


Figure 3. Piggy Report – Cell State

When designing complete, click save button to save your report template into a XML file.

4. Show It To End Users

```
import java.awt.Dimension;
import java.awt.Image;
import java.awt.MediaTracker;
import java.awt.Toolkit;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Date;
```

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
```

```

import javax.swing.JScrollPane;

import net.sf.fjreport.FJReport;
import net.sf.fjreport.control.PrintAction;

public class PiggyLetter extends JFrame {

    public PiggyLetter(){
        super("FJReport Sample - Piggy Letter");
        FJReport report = new FJReport();
        report.loadReport(PiggyLetter.class.getResource("") + "piggy.xml");

        report.setValue("pigimg1", PiggyLetter.class.getResource("") + "piggy1.jpg");
        report.setValue("pigimg2", PiggyLetter.class.getResource("") + "piggy2.jpg");

        report.setValue("content1",
            "Dear Mr Braver:\n"
            + "    Will you give us the pleasure of "
            + "your company on Thursday, March the "
            + "the eighth, at ten o'clock? We are "
            + "planning a small, costume dance."
            );
        report.setValue("signature1", "Piggy Dancer");
        report.setValue("date1", "2006-4-30");

        report.setValue("content2",
            "Dear Miss Dancer:\n"
            + "    Thank you for the good things you provided. "
            + "I'm even unwilling to leave. The wonderful music, "
            + "the romantic dance, the beautiful ladies, all these "
            + "give me lots of enjoyments. I really hope I could "
            + "have stayed still longer.\n"
            + "    Do not hesitate to inform me next time. I believe "
            + "we will have a good time."
            );
        report.setValue("signature2", "Piggy Braver");
        report.setValue("date2", new Date());

        report.setState(FJReport.EDIT_STATE);
        //report.setState(FJReport.READONLY_STATE);
        JScrollPane sc = new JScrollPane(report);

        report.setValue("chksample", true);
        report.setValue("comboboxsample", "Call my piggy");
    }
}

```

```

report.firstPage();

JPanel p = report.getEditToolBarPane();
JButton btnPrint = new JButton(new PrintAction(report));
btnPrint.setPreferredSize(new Dimension(28, 28));
p.add(btnPrint);
add(p, "North");
add(sc, "Center");

setDefaultCloseOperation(EXIT_ON_CLOSE);
setSize(700, 600);
setLocationRelativeTo(null);
setVisible(true);
}
public static void main(String[] args) {
    new PiggyLetter();
}
}

```

To Compile this class, you must have `fjreport.0.4.1.jar` included in your JDK class path. Also, you must have `piggy.xml`, `piggy1.jpg` and `piggy2.jpg` in the same directory of class `PiggyLetter` to run the demo. You can find these files in source package.

FJReport is easy to create, just one clause “`FJReport report = new FJReport();`”, It is inherited from `JPanel`, So, you should create a `JScrollPane` to show the whole report.

Each cell is identified by a string which is the cell's name. Each null cell has value (mostly is instance of `String`) which is shown to end user. The cell value can be access via `FJReport.getValue(String name)` and `FJReport.setValue(String name, Object value)`. Use `FJReport.getCellByName(String name)` to access a specific cell.

A report may contain many pages. Current page object can be accessed by `FJReport.currentPage`. The index of current page is `FJReport.getCurrentPageIndex()`. When page changes, a `StatusChangeEvent` will occur, which contain a string message “Page n of m”.

```

report.addChangeListener(new StatusChangeListener(){
    public void statusChange(int messageType, String message) {
        statusBar.setContent(messageType, message);
    }
});

```

FJReport implements `MultiPage` interface. You can get navigate buttons of FJReport instance by,

```

new JButton(new net.sf.fjreport.control.PrePageAction(report))
new JButton(new net.sf.fjreport.control.NextPageAction(report))
new JButton(new net.sf.fjreport.control.FirstPageAction(report))
new JButton(new net.sf.fjreport.control.LastPageAction(report))

```

The variable `report` is the FJReport instance.

The report running result would look like figure 4.

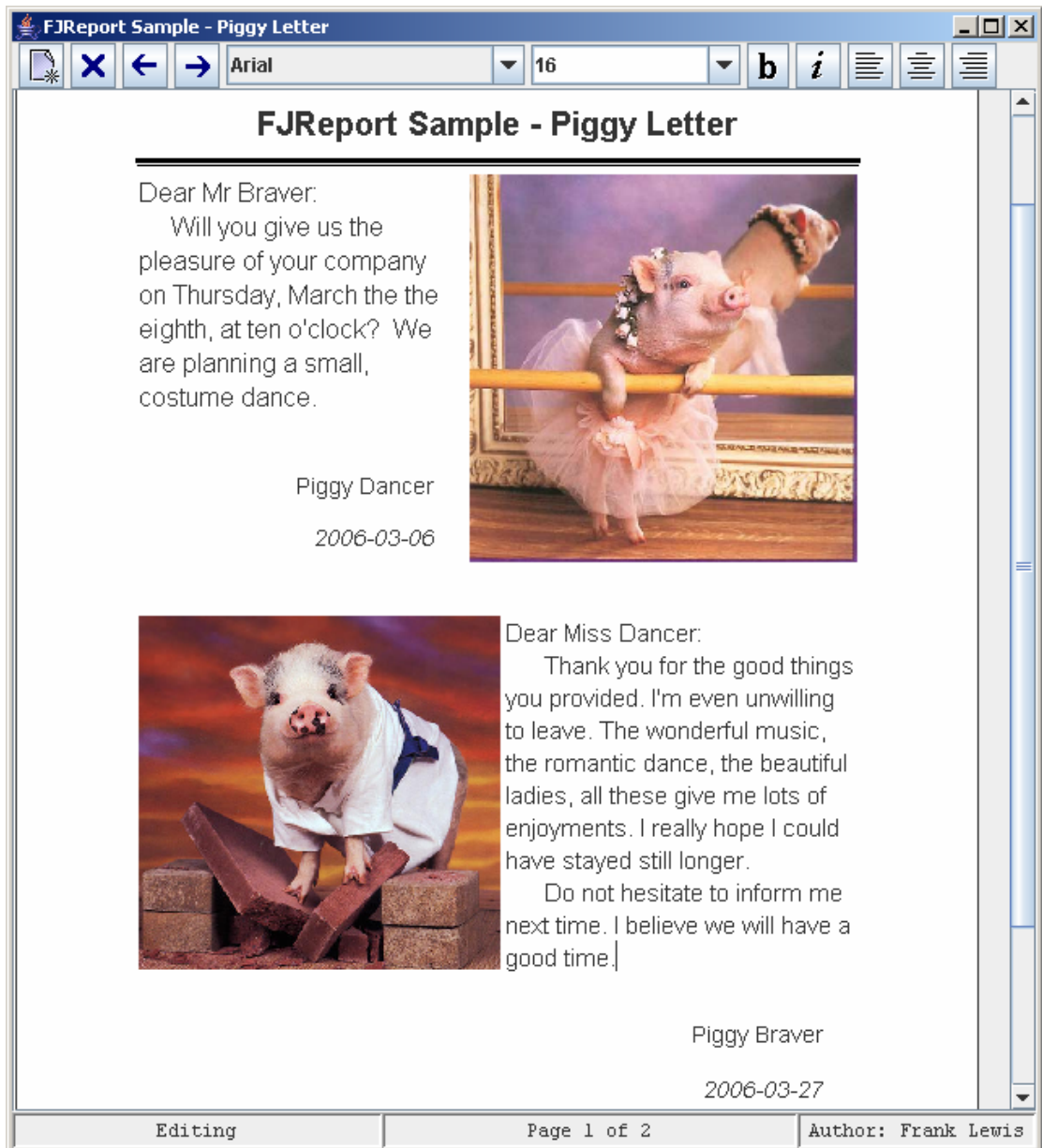


Figure 4. Piggy Letter – Running Snapshot

5. Other Components

DatePicker

See `net.sf.fjreport.datepicker`.

there are two classes, `FJDteField` and `FJDatePicker`.

StatusBar

See `net.sf.fjreport.statusbar`.

Paged jTable print

See `net.sf.fjreport.samples.TablePrintSample`

ModalDialog

See net.sf.util.ModalDialog

All include main() functions to show a demo.

6. Realse

0.4.1 release

some bugs fixed add French language translation

0.4.2 release

- a. fix landscape orientation page save/load error.
- b. fix cell font setting error.
- c. add numeric id to line and cell add Zoom class, zoom function is not yet completed

0.4.3 release

- a. fix a bug which can't keep old cells when state switch from drawline_state to cell_state
- b. lines and cells automatic adjust to changed page format

7. Future features

a.

only 3 states for FJReport

```
public static final int STATE_DESIGN = 1;  
public static final int STATE_EDIT = 2;  
public static final int STATE_PREVIEW = 3;
```

design state = drawing_state + cell_state
in preview state, line position and cell bounds can be adjusted by user

b.

support zoom

c.

support 1*n, m * n pages layout, for all 3 states

d.

support nested cell. very complicated report can be worked out.

e.

support dataset cells.

f.

an entity bean can be binded to a report

supposed a User bean like

```
class User {
    String name;
    String phone;
    String duty;
    ...
    List<Address> addresses;

    class Address {
        String city;
        String street;
        String house;
        String zip;
    }
}
```

A User object can be directly assigned to a designed report

```
report.bindEntity(user);
```

You will get an UI for inputting User, which is exactly what you get on your customer's printer or fax.

g.

facilitate report designing. can open *.class(compiled entity bean) file in FJReportEditor, find properties of the bean via java reflect mechanism. Then user can set a cell's name by double click the property item of the bean.

8. Notes

Currently, only I work for this project. I don't have much spare time, but I will continue its development.

Any help is appreciated, especially from experienced java guys.

Please feel free to contact me at fj_lewis@users.sourceforge.net. All comments are welcome.