JasperReports Tutorial

> The Struts 2 JasperReports plugin is a bridge from Struts 2 to JasperReports and does not include JasperReports itself, which must be downloaded separately.

JasperReports is one of the leading open-source Java reporting libraries. It compiles `.jrxml` (XML source) to `.jasper` (compiled) files, which in turn can be transformed into several output types including PDF, HTML, CSV, and XLS.

In the following example, we will use the framework to create a PDF with a list of persons. Our action will be used to create a List with `Person` objects, and our JasperReports Result will use this list to fill our template, and return the PDF.

Our Person class

We start by defining a simple `Person` POJO class.

com.acme.test.Person.java

```
package com.acme.test;

public class Person {

    private Long id;

    private String name;

    private String lastName;

    public Person() {
    }

    public Person(String name, String lastName) {
        this.name = name;
        this.lastName = lastName;
    }

    public Person(Long id, String name, String lastName) {
        this.id = id;
        this.name = name;
        this.lastName = lastName;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

JasperReports libraries

Before we can continue, we need to add the JR libraries to our classpath. You can download the JR project here: http://www.sourceforge.net/projects/jasperreports
Save the jasperreports-X-project.zip to your harddisk, and extract the files.

We need the following files:

- dist/jasperreports-X.jar
- lib/commons-*.jar (all the commons - except maybe for commons-logging)
- lib/itext-X.jar
- lib/jdt-compiler.jar

Copy these jars over to your S2_WEBAPP/WEB-INF/lib directory, and add them to your classpath.

Creating the Action

com.acme.test.action.JasperAction

```
package com.acme.test.action;

import java.util.ArrayList;
import java.util.List;

import net.sf.jasperreports.engine.JasperCompileManager;

import com.acme.test.Person;
import com.opensymphony.xwork.ActionSupport;

public class JasperAction extends ActionSupport {

    /** List to use as our JasperReports dataSource. */
    private List<Person> myList;

    public String execute() throws Exception {

        // Create some imaginary persons.
        Person p1 = new Person(new Long(1), "Patrick", "Lightbuddie");
        Person p2 = new Person(new Long(2), "Jason", "Carrora");
        Person p3 = new Person(new Long(3), "Alexandru", "Papesco");
        Person p4 = new Person(new Long(4), "Jay", "Boss");

        // Store people in our dataSource list (normally they would come from a database).
        myList = new ArrayList<Person>();
        myList.add(p1);
        myList.add(p2);
        myList.add(p3);
        myList.add(p4);

        // Normally we would provide a pre-compiled .jrxml file
        // or check to make sure we don't compile on every request.
        try {
            JasperCompileManager.compileReportToFile(
                    "S2_WEBAPP/jasper/our_jasper_template.jrxml",
                    "S2_WEBAPP/jasper/our_compiled_template.jasper");
        } catch (Exception e) {
            e.printStackTrace();
            return ERROR;
        }

        return SUCCESS;
    }

    public List<Person> getMyList() {
        return myList;
    }
}
```

Our JasperAction creates a list of several People. The JasperCompileManager compiles the jrxml template to a .jasper file.

> Again, don't use this in production code. You should of course either provide compiled templates, or do some sort of checking to avoid compiling the template on every request. But for our demonstration, or development, this suits our needs just fine.

Our Jasper template

JR uses XML configuration to define templates which are compiled to .jasper files. These templates define the resulting report.
This is a handwritten version - for more complex versions I seriously suggest taking a look a the various GUI designers.

our_jasper_template.jrxml

```
<?xml version="1.0"?>
<!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report Design//EN" "http://jasperreports.sourceforge.net
/dtds/jasperreport.dtd">
<jasperReport name="jasper_test">
  <!-- Our fields from the Person class. -->
  <field name="name" class="java.lang.String"/>
  <field name="lastName" class="java.lang.String"/>
  <title>
    <band height="50">
      <staticText>
        <reportElement x="0" y="0" width="180" height="15"/>
        <textElement/>
        <text><![CDATA[Struts 2 JasperReports Sample]]></text>
      </staticText>
    </band>
  </title>
  <pageHeader>
    <band/>
  </pageHeader>
  <columnHeader>
    <band height="20">
      <staticText>
        <reportElement x="180" y="0" width="180" height="20"/>
        <textElement>
          <font isUnderline="true"/>
        </textElement>
        <text><![CDATA[NAME]]></text>
      </staticText>
      <staticText>
        <reportElement x="360" y="0" width="180" height="20"/>
        <textElement>
          <font isUnderline="true"/>
        </textElement>
        <text><![CDATA[LASTNAME]]></text>
      </staticText>
    </band>
  </columnHeader>
  <detail>
    <band height="20">
      <textField>
        <reportElement x="180" y="0" width="180" height="15"/>
        <textElement/>
        <textFieldExpression><![CDATA[$F{name}]]></textFieldExpression>
      </textField>
      <textField>
        <reportElement x="360" y="0" width="180" height="15"/>
        <textElement/>
        <textFieldExpression><![CDATA[$F{lastName}]]></textFieldExpression>
      </textField>
    </band>
  </detail>
  <columnFooter>
    <band/>
  </columnFooter>
```

```
  <pageFooter>
    <band height="15">
      <staticText>
        <reportElement x="0" y="0" width="40" height="15"/>
        <textElement/>
        <text><![CDATA[Page:]]></text>
      </staticText>
      <textField>
        <reportElement x="40" y="0" width="100" height="15"/>
        <textElement/>
        <textFieldExpression class="java.lang.Integer"><![CDATA[$V{PAGE_NUMBER}]]></textFieldExpression>
      </textField>
    </band>
  </pageFooter>
  <summary>
    <band/>
  </summary>
</jasperReport>
```

Save this file in `S2_WEBAPP/jasper/` as 'our_jasper_template.jrxml'.

Most important: we declared the fields name and lastName (two properties from our `Person` class). This means we will now be able to use these fields in our Jasper template.

We define two columnheaders (NAME and LASTNAME), and then add our fields to the detail band (for a better explanation, look at the JR tutorial). This 'detail' band will iterate over our List of People. This is the default behaviour of JR - so if you want to display more information from the Person, add them to this band.

In the detail band we use the

```
$F{name}
```

expression. JasperReports will ask Struts to retrieve the `name` field value from a `Person` object; the `lastName` field is handled the same way.

The rest is markup to define the layout.

> Use a logger (commons-logging, log4j, ...) to watch `org.apache.struts2.views.jasperreports` in debug mode, if you have any troubles.

Registering the Action

Using the JasperReports plugin requires adding the JasperReports result type as well as normal action configuration.

struts.xml

```
<package name="default" namespace="/" extends="jasperreports-default">
    <action name="myJasperTest" class="com.acme.test.action.JasperAction">
        <result name="success" type="jasper">
            <param name="location">/jasper/our_compiled_template.jasper</param>
            <param name="dataSource">myList</param>
            <param name="format">PDF</param>
        </result>
    </action>
    ...
</package>
```

To use the JasperReports result type we must either (a) extend the `jasperreports-default` package that defines it or (b) manually define the JasperReport `jasper` result type ourselves.

In the above example we extend the `jasperreports-default` package; we can define the `jasper` result type manually by defining it the same way the JasperReport plugin does:

Manually defining the "jasper" result type

```
<result-types>
    <result-type name="jasper" class="org.apache.struts2.views.jasperreports.JasperReportsResult"/>
</result-types>
```

We configure our JasperAction with the name 'myJasperTest' - this means that we can execute this Action by sending a request to `myJasperTest.action` in our browser.

```
<action name="myJasperTest" class="com.acme.test.action.JasperAction">
```

When our JasperAction executes correctly, we will use the Result type registered with the name 'jasper'. As discussed above the "jasper" result type is available from either extending the "jasperreports-default" package or by defining the result type manually.

```
<result name="success" type="jasper">
```

The "location" parameter defines the location of the compiled jasper file, which will be filled by Struts 2 with our dataSource:

```
<param name="location">/jasper/our_compiled_template.jasper</param>
```

The "dataSource" parameter defines the action property containing the collection of objects to use in our report. In this case it's the `myList` property which we manually filled with some `Person` objects.

```
<param name="dataSource">myList</param>
```

The "format" parameter specifies the output format of the report. Possible values include PDF, CSV, XLS and HTML.

```
<param name="format">PDF</param>
```

Conclusion

You should now be able to execute http://localhost:8080/YOUR_WEBAPP/myJasperTest.action - and you should see a nice list of names.
Struts provides probably the most elegant way to deal with JasperReport files; specify the location of the .jasper file, specify what dataSource you want to use, and there you go.

Back to Tutorials