Spring Workbook Configuration

Custom Data Solutions

Contents

Part I
Overview

1 Introduction

The Spring Workbook Configuration (SWC) is a newly constructed tool to simplify the way to generate workbooks by defining a SQL query, passing style information for the workbook, and passing layout information to produce an excel file. This file externalization provides for a simpler way to create these workbooks by isolating the three aforementioned factors into a communal .XML file. This new tool does no require additional java coding and compilation. It is an easy-to-use tool that can be used by non-java developers to create an excel workbook.

1.1 Assumption

In order to use this tool, you will need to be familiar with SQL, PL/SQL, some knowledge of XML and basic knowledge of the principles of Spring.

2 Process Flow

In order for this process to take place there must be a seperate properties file that is defined that is referenced by the XML file. This is set up as a place to externally store commonly used logic that is then referenced in the XML file and used multiple times. For example, if you are looking to use the current year more than one time throughout your code, you should define it in the properties file and reference is in the XML file as dollarsignyear. Here is an example properties file:

```
outputFile=/tmp/test_dataset.xls
deleteExistingOutputFile=true
year=2009
month=1
```

3 How To Use This Tool?

Initially, the Spring packages need to be listed at the top of the XML file in order to have access to the proper utilities and provide proper validation for the XML file. This is set up similarly to how a java class lists the packages that will be referenced in that class. These utilities and frameworks are essential to having this spring class work properly. Here is what that utility listing would look like:

```
 <?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:util="http://www.springframework.org/schema/util" xmlns:xsi="http://www.w3.org/2001/XMLSch
        xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
           http://www.springframework.org/schema/util
           http://www.springframework.org/schema/util/spring-util-2.5.xsd">
```

Within the .XML file being created the database connection needs to be set up properly before the following steps are taken. The connection can be created by defining the datsource bean. The connection information can be changed to whatever schema and database that needs to be connected to. This information is similar in format to connecting to a database through Toad and SQL Developer so this should look familiar. This is the java method to connect to oracle. The sample datasource bean is listed below:

```
        <!--
                see spring documentation for using datasources
        -->
        <bean id="datasource" class="org.apache.commons.dbcp.BasicDataSource"
                destroy-method="close">
                <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
                <property name="url" value="jdbc:oracle:thin:@db1:1521:prod" />
                <property name="username" value="roprod" />
                <property name="password" value="*****" />
                <property name="initialSize" value="1" />
                <property name="maxActive" value="10" />
                <property name="accessToUnderlyingConnectionAllowed" value="true" />
        </bean>
        <bean id="sqlRunner"
                class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate">
                <constructor-arg ref="datasource" />
        </bean>
```

The sqlRunner is the object that runs the query in th enext step. It references the datasource which is why it is listed with it here rather than with the query.

3.1 Create the SQL Query

The first step to run this workbook generation tool is to create the SQL query (or multiple queries) to gather the data required for the workbook. This should look familiar to people who have created and generated SQL queries before. The select statement will identify the columns to be used from the specified table and the where clause defines the bind variables to be used in the next step. The query identifies what parameters need to be used.
Here is an example of a SQL Query to get data for a workbook:

```
        <bean id="salesQuery" factory-bean="sqlRunner" factory-method="queryForRowSet">
                <constructor-arg><value><![CDATA[
        select
                VP_DST_NBR,
                VP_CST_NBR,
                VP_ITEM_NBR,
                SHIP_DT,
                INV_NO,
                POSTED_CASES,
                EXT_NET_AMT
        from
                kllg.vp_sales_dtl
        where
                POSTED_CASES <= :CASES_MAX and
                POSTED_CASES >= :CASES_MIN and
                YR = :YEAR and MTH = :MONTH
                ]]></value></constructor-arg>
                <constructor-arg ref="salesParameters" />
        </bean>
```

3.2 Derive Bind Variables From Query As Per User Requirements

The bind variables for the workbook need to be set up and called as per the ’where’ clause of the select statement above. Each entry key will define the value of the bind variable. These parameters are given by the User as to what needs to be the range of data for the workbook. This pairs the key values and the name values together. Here is an example:

```
 <util:map id="salesParameters" map-class="java.util.LinkedHashMap">
```

```
                <entry key="MONTH">
                        <value>1</value>
                </entry>
                <entry key="YEAR">
                        <value>2009</value>
                </entry>
                <entry key="CASES_MIN">
                        <value>75</value>
                </entry>
                <entry key="CASES_MAX">
                        <value>99</value>
                </entry>
</util:map>
```

As you can see, the entry names match the items from the SQL statement and then values are assigned to each entry.

3.3 Execute the Query and Get the Dataset

The final step of the SQL section is to execute the query to get the dataset. The dataset is defined at the end of the XML document in the secion titled 'Bringing It All Together and Defining the Document'. The dataset stores all of the data into memory before it is written to the workbook. This is beneficial in the example of the crosstabbing worksheet in terms of having to store the data and then rearrange it properly. The following bean associates the dataset with the query:

```
<bean id="salesDataset" class="org.javautil.dataset.DisassociatedResultSetDataset"
                factory-method="getDataset">
                <constructor-arg ref="salesQuery" />
</bean>
```

If a result set from a procedure is required take a look at the examples here:
http://static.springsource.org/spring/docs/2.5.x/api/org/springframework/jdbc/core/namedparam/NamedParameterJdbcT
http://static.springsource.org/spring/docs/2.5.x/api/org/springframework/jdbc/core/SqlReturnResultSet.html

3.4 Create or Import Style

In order to incorporate styles into the workbook, style beans must be created in order to set style properties and create styles to be called by the dataset that you are generating. Each style should be named and work in a parent-child fashion meaning that base styles should be created and then referenced in new styles as they are created so the values in the defined base style are re-used and not re-written. Here is an example:

```
        <bean abstract="true" name="baseStyle" class="org.javautil.document.style.StyleDefinition">
                <property name="fontFace" value="Arial" />
                <property name="fontHeight" value="10" />
        </bean>
        <bean abstract="true" name="headerBaseStyle" class="org.javautil.document.style.StyleDefinition'
        parent="baseStyle">
                <property name="name" value="header" />
                <property name="backgroundColor" value="#ccc" />
                <property name="border" value="1 #000" />
        </bean>
        <bean abstract="true" name="dataBaseStyle" class="org.javautil.document.style.StyleDefinition"
        parent="baseStyle">
                <property name="name" value="header" />
                <property name="backgroundColor" value="#fff" />
                <property name="border" value="1 #ccc" />
        </bean>
```

As you can see above, there are two styles that have been defined: the headerBaseStyle and the dataBaseStyle. Each of these styles share the same base style which is defined at the top to use the Arial font and a value of 10 for the size of that font. Because they both share this baseStyle they are designated as children of that style that fork off of the baseStyle because they have different defined properties.

```
        <util:list id="styles">
                <bean class="org.javautil.document.style.StyleDefinition">
                        <property name="name" value="default" />
                        <property name="border" value="1 #000" />
                </bean>
                <bean class="org.javautil.document.style.StyleDefinition" parent="headerBaseStyle">
                        <property name="name" value="headerRight" />
                        <property name="horizontalAlignment" value="right" />
                </bean>
                <bean class="org.javautil.document.style.StyleDefinition" parent="headerBaseStyle">
                        <property name="name" value="headerLeft" />
                        <property name="horizontalAlignment" value="left" />
                </bean>
                <bean class="org.javautil.document.style.StyleDefinition" parent="dataBaseStyle">
                        <property name="name" value="dataDollars" />
                        <property name="formatMask" value="#,###,###,##0.00" />
                        <property name="horizontalAlignment" value="right" />
                </bean>
                <bean class="org.javautil.document.style.StyleDefinition" parent="dataBaseStyle">
                        <property name="name" value="dataInteger" />
                        <property name="horizontalAlignment" value="right" />
                </bean>
                <bean class="org.javautil.document.style.StyleDefinition" parent="dataBaseStyle">
                        <property name="name" value="dataString" />
                        <property name="horizontalAlignment" value="left" />
                </bean>
        </util:list>
        <bean id="documentStyles" class="org.javautil.document.style.StyleUtil"
                factory-method="parseStyles">
                <constructor-arg>
                        <bean class="org.javautil.document.style.DefaultStyleParser" />
                </constructor-arg>
                <constructor-arg ref="styles" />
        </bean>
```

The different style types that are defined continue to fork off as different properties are needed and new styles are creates for use with the header and use with the data itself. In this case, the colors for the background and the text are defined as well as the alignment and font format. The styles are further separated out into two header styles and 3 data styles for use when the styles for the sheet are being set in the next step. The parser after the styles is there to have the ability to parse any of the styles into java objects in order to take advantage of the multipurpose output format available with this class.

3.5 Style Interface

This is a listing of all of the potential properties that a style can have.

3.5.1 private String name;

- The name of the style

3.5.2 private String fontColor;

- The color of the font, typically defined in a CSS-like color format.
- Examples: ffffff: white, fff: short hand for white, ff0000 red, f00: short hand for red, 00ff00 green, 0f0: short hand for green, 0000ff blue, 00f: short hand for blue, 000000: black, 000: short hand for black

3.5.3 private String fontFace;

- The name of the font that will be used for text items.
- Examples: "Arial", "Garamond", "Times New Roman"

3.5.4 private String fontHeight;

- The font height in points of the font.
- Examples: 8, 9, 10, 12, 14, 18, 22

3.5.5 private String fontStyle;

- The style of the font.
- Examples: italic, normal

3.5.6 private String fontWeight;

- The weight of the font.
- Examples: bold, normal

3.5.7 private String fontUnderlineStyle;

- The underline of the font.
- Examples, none, single, double

3.5.8 private String wordWrap;

- Should word wrap be applied?
- Examples: true, false

3.5.9 private String verticalAlignment;

- Examples: top, bottom, middle

3.5.10 private String horizontalAlignment;

- Examples: left, center, right

3.5.11 private String backgroundColor;

- The color of the background, typically defined in a CSS-like color format.
- Examples: ffffff: white, fff: short hand for white, ff0000 red, f00: short hand for red, 00ff00 green, 0f0: short hand for green, 0000ff blue, 00f: short hand for blue, 000000: black, 000: short hand for black

3.5.12 private String formatMask;

- The format mask for the text.
- In some documents (like HTML), this will be applied to the text directly to alter the value that is printed. In other documents (like Excel), this will be used as a true format mask.
- Examples: "MM/dd/yyyy", "000,000.00", "MMM, yyyy", "0.00"

3.5.13 private String border;

- The border, typically defined in the CSS-like border format.
- Examples: "1 0000" : thinnest black border, "2 ff0000" : thicker red border

3.5.14 private String borderTop;

- Like border, except only for the top edge; overrides the border setting.

3.5.15 private String borderRight;

- Like border, except only for the right edge; overrides the border setting.

3.5.16 private String borderBottom;

- Like border, except only for the bottom edge; overrides the border setting.

3.5.17 private String borderLeft;

- Like border, except only for the left edge; overrides the border setting.

3.6 Layout/Bean Shell Definition

The layout area of this XML file is the area in which the page layout is set as well as the data is defined and given a style and a location. This is done by using content settings, renderer settings, and dataset settings. The layout section should first be defined with a template that is defined at the end of the spring class which is detailed in the 'Bringing It All Together and Defining the Document' section of this document.

```
<bean id="salesRepTemplate" class="org.javautil.document.renderer.BshRenderTemplate">
        <property name="bshScript">
                <value><![CDATA[
import java.util.Date;
```

The content, renderer, and dataset information should then be listed after the above bean and then closed off with a

```
]]></value>
        </property>
</bean>
```

The content, renderer, and dataset settings are expanded on below:

3.6.1 Content

The content portion of the layout are where the definitons for your general page dimensions exist. Margins, print widths, page numbers, and sheet names are all defined in this section. Here is an example:

```
content.setSheetName("Sales Detail");
content.setPrintWidthPages(1);
content.setPageMargins(new double[] { 0.8, 0.25, 0.8, 0.25 });
content.setPageOrientationLandscape();
content.setPageHeaderCenterText("Sales Detail Worksheet");
content.setPageFooterLeftText("http://www.kelloggvend.com");
content.setPageFooterCenterText("Kellogg");
content.setPageFooterPageNumbers("right");
content.setColumnWidths(new int[] { 14, 12, 12, 10, 10, 2 });
```

These are just examples of possible content calls to use. The ColumnWidth is by excel character. All the content possibilties are listed further below.

3.6.2 Renderer

The renderer layout type is the go-between step between the content and the dataset. The renderer portion provides the implementation of the styles that were previously set for the data that is being populated. The first argument in parentheses is the data that will go into the cell whereas the second argument is the style to be applied to that data. First a header example and then a data example:
Header

```
renderer.addData("Distributor Id", "headerLeft");
renderer.addData("Customer Id", "headerRight");
renderer.addData("Product Id", "headerRight");
renderer.addData("Dollars", "headerRight");
renderer.addData("Cases", "headerRight");
renderer.nextLine();
```

NOTE: The nextLine signifies that the data will always go after these headers in sequential order on the worksheet. This gives the data the ability to be printed on its own line. This works similarly to a linebreak.
Data

```
renderer.addData(distributorId, "dataString");
renderer.addData(customerId, "dataString");
renderer.addData(productId, "dataString");
renderer.addData(dollars, "dataDollars");
renderer.addData(cases, "dataInteger");
renderer.nextLine();
```

These are just examples of possible renderer calls to use. All of them are listed below.

3.6.3 Dataset

The dataset is the portion of the worksheet where the SQL query is being referenced to get data from the columns in the table(s) to populate data for the worksheet. The column names are case sensitive. The term 'integer' in this case can be any java type available in the dataset depending on the size of the data being generated. Here is an example:

```
iterator = dataset.getDatasetIterator();
while (iterator.next()) {
        Integer distributorId = iterator.getInteger("VP_DST_NBR");
        Integer customerId = iterator.getInteger("VP_CST_NBR");
        Integer productId = iterator.getInteger("VP_ITEM_NBR");
        Integer cases = iterator.getInteger("POSTED_CASES");
        Double dollars = iterator.getDouble("EXT_NET_AMT");
        Date shipDt = iterator.getDate("SHIP_DT");
```

3.6.4 Content Interface (General)

```
public Map<String, Style> getStylesByName();
public void setStylesByName(Map<String, Style> stylesByName);
public R getRowAt(int rowIndex, boolean createRow);
public C getCellAt(int rowIndex, int columnIndex, boolean createRow, boolean createCell);
public void setBlankCellAt(int rowIndex, int columnIndex, Style style);
public void setCellAt(int rowIndex, int columnIndex, Object data, Style style);
public void setFormulaCellAt(int rowIndex, int columnIndex, String formula, Style style);
public TypewriterRendererFactory getRendererFactory();
public Dimension getDimension();
public Rectangle getBounds();
```

3.6.5 Content Interface (Workbook Specific)

```
public void setFreezePane(int rowIndex, int columnIndex);
public void setVerticalFreezePane(int columnIndex);
public void setHorizontalFreezePane(int rowIndex);
public void setRowHeights(int startRow, int endRow, int rowHeight);
public void setRowHeight(int rowIndex, int rowHeight);
public void setColumnWidth(int columnIndex, int columnWidth);
public void setColumnWidths(int startColumnIndex, int[] columnWidths);
public void setColumnWidths(int[] columnWidths);
public void mergeCellRange(int rowIndexStart, int rowIndexEnd,
                int columnIndexStart, int columnIndexEnd);
public void setColumnFormulaCellAt(int rowIndex, int columnIndex,
                int firstFormulaRowIndex, int lastFormulaRowIndex,
                int formulaColumnIndex, Style style, String function,
                List<String> argsBefore, List<String> argsAfter);
public void setColumnFormulaCellAt(int rowIndex, int columnIndex,
                int firstFormulaRowIndex, int lastFormulaRowIndex,
                int formulaColumnIndex, Style style, String function);
public void setRowFormulaCellAt(int rowIndex, int columnIndex,
                int formulaRowIndex, int firstFormulaColumnIndex,
                int lastFormulaColumnIndex, Style style, String function);
public void setPageOrientationLandscape();
public void setPageOrientationPortrait();
public boolean isPageOrientationLandscape();
public boolean isPageOrientationPortrait();
public void setPrintWidthPages(int numberOfPages);
public void setPrintHeightPages(int numberOfPages);
public void setPageHeaderLeftText(String header);
public void setPageHeaderCenterText(String header);
public void setPageHeaderRightText(String header);
public void setPageFooterLeftText(String header);
public void setPageFooterCenterText(String header);
public void setPageFooterRightText(String header);
public void setPageHeaderPageNumbers(String alignment);
public void setPageFooterPageNumbers(String alignment);
public void setPageMargins(double[] margins);
public void setPageMargins(double margin);
public void setSheetName(String sheetName);
```

3.6.6 Renderer Interface (General)

```
public TypewriterBehavior getBehavior();
public void setBehavior(TypewriterBehavior behavior);
public void nextLine();
public int getRowIndex();
public void setRowIndex(int rowIndex);
public int getColumnIndex();
public void setColumnIndex(int columnIndex);
public void skip(int numberOfCells);
public void addBlank(String styleName);
public void addData(Object data, String styleName);
public void addFormula(String formula, String styleName);
public void addStyles(Map<String, Style> styles);
public Dimension getDimension();
public Rectangle getBounds();
```

3.6.7 Renderer Interface (Workbook Specific)

```
public void setFreezePane();
public void setVerticalFreezePane();
public void setHorizontalFreezePane();
public void setRowHeight(int rowHeight);
public void setColumnWidth(int columnWidth);
public void setColumnWidths(int[] columnWidths);
public void mergeCellRange(int rowIndexEnd, int columnIndexEnd);
public void addColumnFormulaCell(int firstFormulaRowIndex,
                int lastFormulaRowIndex, int formulaColumnIndex, Style style,
                String function, List<String> argsBefore, List<String> argsAfter);
public void addColumnFormulaCell(int firstFormulaRowIndex,
                int lastFormulaRowIndex, int formulaColumnIndex, Style style,
                String function);
public void addRowFormulaCell(int formulaRowIndex,
                int firstFormulaColumnIndex, int lastFormulaColumnIndex,
                Style style, String function);
```

3.6.8 Dataset Interface

```
public DatasetIterator<T> getDatasetIterator();
public DatasetMetadata getMetadata();
public String getName();
```

3.6.9 Dataset Iterator Interface

```
public DatasetMetadata getDatasetMetadata();
public Date getDate(int column) throws DatasetException;
public Date getDate(String columnName) throws DatasetException;
public Double getDouble(int column) throws DatasetException;
public Double getDouble(String column) throws DatasetException;
public Integer getInteger(int column) throws DatasetException;
public Integer getInteger(String column) throws DatasetException;
public Number getNumber(int columnIndex) throws DatasetException;
```

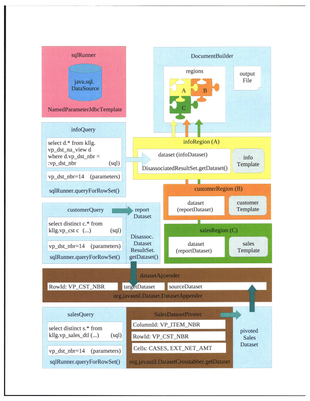
```
        public Number getNumber(String column) throws DatasetException;
        public T getObject(int columnIndex) throws DatasetException;
        public T getObject(String column) throws DatasetException;
        public List<T> getRowAsList() throws DatasetException;
        public Map<String, Object> getRowAsMap() throws DatasetException;
        public String getString(int column) throws DatasetException;
        public String getString(String column) throws DatasetException;
        public Date getTimestamp(int column) throws DatasetException;
        public Date getTimestamp(String column) throws DatasetException;
        public boolean hasNext() throws DatasetException;
        public boolean next() throws DatasetException;
        public int getRowCount() throws DatasetException;
```

3.7 Bringing It All Together and Defining the Document

The final step in creating the workbook is to define the document. At this point a directory for the file to be put in needs to be defined as well as the output file name. Other things that need to be defined here are the format of the document you're creating as well as the regions for the document that are referenced in the previous beans. Here is an example of this:

```
 <bean class="org.javautil.document.DocumentBuilder">
        <property name="outputFile" value="${outputFile}" />
        <property name="deleteExistingOutputFile" value="${deleteExistingOutputFile}" />
        <property name="format" value="xls" />
        <property name="regions">
                <list>
                <bean class="org.javautil.document.SimpleRegion">
                        <property name="name" value="Sales" />
                        <property name="dataset" ref="salesDataset" />
                        <property name="parameters" ref="salesParameters" />
                        <property name="documentStyles" ref="documentStyles" />
                        <property name="renderTemplate" ref="salesTemplate" />
                        <property name="layoutConstraints">
                                <bean class="org.javautil.document.layout.AbsoluteLayout">
                                        <property name="row" value="0" />
                                        <property name="column" value="0" />
                                </bean>
                        </property>
                </bean>
                </list>
        </property>
</bean>
```

4 Spring Workbook Configuration Flow Diagram



5 Pivoting Datasets

To allow for crosstabbing worksheets to be made available to be created and incorporated into the dataset portion of generating a workbook there are a few steps that need to be taken:

- A datasetAppender needs to be defined:

```
        <bean id="datasetAppender" class="org.javautil.dataset.DatasetAppender" />
        <bean id="sqlRunner"
                class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate">
                <constructor-arg ref="datasource" />
        </bean>
```

- The salesDatasetPivoter should be identified based on the row identifiers, column identifiers, and the cells through the SQL query:

```
<bean id="salesDatasetPivoter" class="org.javautil.dataset.DatasetCrosstabber">
        <property name="dataset" ref="salesDataset" />
        <property name="crosstabColumns">
                <bean class="org.javautil.document.crosstab.CrossTabColumnsImpl">
                        <!-- row identifiers -->
                        <constructor-arg ref="rowIdentifiers" />
                        <!-- column identifier -->
                        <constructor-arg>
                                <value>VP_ITEM_NBR</value>
                        </constructor-arg>
                        <!-- cells -->
                        <constructor-arg>
                                <list>
                                        <value>CASES</value>
                                        <value>EXT_NET_AMT</value>
                                </list>
                        </constructor-arg>
                </bean>
        </property>
</bean>
```

- Then the pivotedSalesDataset needs to be called to get the data from the dataset to be connected:

```
         <bean id="pivotedSalesDataset" factory-bean="salesDatasetPivoter"
        factory-method="getDataSet" />
```

- And finally the datasets and row identifers need to be brought together:

```
<bean class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
        <property name="targetObject" ref="datasetAppender" />
        <property name="targetMethod" value="appendRight" />
        <property name="arguments">
                <list>
                        <!-- target dataset, already contains customer information at this point --
                        <ref bean="reportDataset" />
                        <!-- source dataset, contains only the sales information at this point -->
```

```
                            <ref bean="pivotedSalesDataset" />
                            <!-- row identifiers -->
                            <ref bean="rowIdentifiers" />
                    </list>
            </property>
    </bean>
```

5.1 Crosstab Document Test

```
 <?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:util="http://www.springframework.org/schema/util"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
           http://www.springframework.org/schema/util
           http://www.springframework.org/schema/util/spring-util-2.5.xsd">

        <import resource="common-styles.xml" />

<bean id="datasource" class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
        <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
        <property name="url" value="jdbc:oracle:thin:@db1:1521:prod" />
        <property name="username" value="roprod" />
        <property name="password" value="*****" />
        <property name="accessToUnderlyingConnectionAllowed" value="true" />
</bean>
<bean id="datasetAppender" class="org.javautil.dataset.DatasetAppender" />
<bean id="sqlRunner"
        class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate">
        <constructor-arg ref="datasource" />
</bean>
<bean id="infoQuery" factory-bean="sqlRunner" factory-method="queryForRowSet">
        <constructor-arg>
        <value><![CDATA[
        select d.* from kllg.vp_dst_na_view d where vp_dst_nbr = :vp_dst_nbr
        ]]></value>
        </constructor-arg>
        <constructor-arg ref="queryParameters" />
</bean>
<bean id="infoDataset" class="org.javautil.dataset.DisassociatedResultSetDataset"
        factory-method="getDataset">
        <constructor-arg ref="infoQuery" />
</bean>

<bean id="customerQuery" factory-bean="sqlRunner" factory-method="queryForRowSet">
        <constructor-arg>
                <value><![CDATA[
        select distinct c.* from kllg.vp_cst c
        where exists (select * from kllg.vp_sales_dtl s where s.vp_cst_nbr = c.vp_cst_nbr and
        s.vp_dst_nbr = :vp_dst_nbr and s.mth = :month and s.yr = :year) order by c.name
        ]]></value>
        </constructor-arg>
        <constructor-arg ref="queryParameters" />
</bean>
<bean id="reportDataset" class="org.javautil.dataset.DisassociatedResultSetDataset"
        factory-method="getDataset">
        <constructor-arg ref="customerQuery" />
</bean>

<bean id="salesQuery" factory-bean="sqlRunner" factory-method="queryForRowSet">
        <constructor-arg>
                <value><![CDATA[
        WITH csts AS (
        SELECT  DISTINCT vp_cst_nbr
        FROM    kllg.vp_sales_dtl
        WHERE   vp_dst_nbr              =       :vp_dst_nbr
        AND     mth                             =       :month
        AND     yr                              =       :year ),
        items AS(
        SELECT  vp_item_nbr
        FROM    kllg.vp_item
        WHERE   vp_item_nbr IN (98016, 31732, 12261, 13065, 31132))
        SELECT  csts.vp_cst_nbr,
        items.vp_item_nbr,
        sum(s.posted_cases) cases,
        sum(s.ext_net_amt) ext_net_amt
        FROM
        kllg.vp_sales_dtl               s,
        csts,
        items
        WHERE
                s.vp_dst_nbr            =       :vp_dst_nbr
        AND     s.mth                           =       :month
        AND     s.yr                            =       :year
        AND s.vp_cst_nbr        =   csts.vp_Cst_nbr
        AND s.vp_item_nbr       =   items.vp_item_nbr
        GROUP BY csts.vp_cst_nbr,
                 items.vp_item_nbr
        union all
        SELECT  csts.vp_cst_nbr,
        items.vp_item_nbr,
        0, 0
        FROM
        csts,
        items
        WHERE
        not exists (
        select  'x'
        from    kllg.vp_sales_dtl    s2
        where   s2.vp_dst_nbr           =       :vp_dst_nbr
        AND     s2.mth                          =       :month
        AND     s2.yr                           =       :year
        AND s2.vp_cst_nbr       =   csts.vp_Cst_nbr
        AND s2.vp_item_nbr      =   items.vp_item_nbr )
        ORDER BY 1, 2
        ]]></value>
        </constructor-arg>
        <constructor-arg ref="queryParameters" />
</bean>
<bean id="salesDataset" class="org.javautil.dataset.DisassociatedResultSetDataset"
        factory-method="getDataset">
        <constructor-arg ref="salesQuery" />
</bean>
<util:list id="rowIdentifiers">
        <value>VP_CST_NBR</value>
</util:list>
<bean id="salesDatasetPivoter" class="org.javautil.dataset.DatasetCrosstabber">
        <property name="dataset" ref="salesDataset" />
        <property name="crosstabColumns">
                <bean class="org.javautil.document.crosstab.CrossTabColumnsImpl">
                        <!-- row identifiers -->
                        <constructor-arg ref="rowIdentifiers" />
                        <!-- column identifier -->
                        <constructor-arg>
                                <value>VP_ITEM_NBR</value>
                        </constructor-arg>
                        <!-- cells -->
                        <constructor-arg>
                                <list>
                                        <value>CASES</value>
                                        <value>EXT_NET_AMT</value>
                                </list>
                        </constructor-arg>
                </bean>
        </property>
</bean>
<bean id="pivotedSalesDataset" factory-bean="salesDatasetPivoter"
        factory-method="getDataSet" />

<bean class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
        <property name="targetObject" ref="datasetAppender" />
        <property name="targetMethod" value="appendRight" />
        <property name="arguments">
                <list>
                        <!-- target dataset, already contains customer information at this point -->
                        <ref bean="reportDataset" />
                        <!-- source dataset, contains only the sales information at this point -->
                        <ref bean="pivotedSalesDataset" />
                        <!-- row identifiers -->
                        <ref bean="rowIdentifiers" />
                </list>
        </property>
</bean>

<util:map id="queryParameters" map-class="java.util.LinkedHashMap">
        <entry key="month">
                <value>1</value>
        </entry>
        <entry key="year">
```

```
                <value>2008</value>
        </entry>
        <entry key="vp_dst_nbr">
                <value>14</value>
        </entry>
</util:map>

<bean id="infoTemplate" class="org.javautil.document.renderer.BshRenderTemplate">
        <property name="bshScript">
                <value><![CDATA[
import java.util.Date;

        // name the sheet, setup print settings
        content.setSheetName("Sales Detail");
        content.setPrintWidthPages(1);
        content.setPageMargins(new double[] { 0.8, 0.25, 0.8, 0.25 });
        content.setPageOrientationLandscape();
        content.setPageHeaderCenterText("Sales Summary");
        content.setPageFooterLeftText("http://www.kelloggvend.com");
        content.setPageFooterCenterText("Kellogg");
        content.setPageFooterPageNumbers("right");

        iterator = dataset.getDatasetIterator();
        iterator.next();
        renderer.addData("Distributor:", "dataString");
        renderer.addData(iterator.getString("NAME"), "dataString");
        renderer.nextLine();
        renderer.addData("Sales Dates:", "dataString");
        Date date = new java.text.SimpleDateFormat("MM/dd/yyyy").parse("${month}/01/${year}");
        String dateText = new java.text.SimpleDateFormat("MMM, yyyy").format(date);
        renderer.addData(dateText, "dataString");

        // write the cells
        if (iterator.next()) {
                throw new IllegalStateException("too many rows from info query");
        }
                ]]></value>
                </property>
</bean>

<bean id="customerTemplate" class="org.javautil.document.renderer.BshRenderTemplate">
        <property name="bshScript">
                <value><![CDATA[
import java.util.Date;

        // set column widths
        content.setColumnWidths(new int[] { 10, 25, 30, 18, 6, 9, 20 });

        // skip a row for the item numbers row
        for (int i = 0; i < 7; i++) {
                renderer.addBlank("headerLeft");
        }
        renderer.nextLine();
        // writer headers
        renderer.addData("Customer Id", "headerLeft");
        renderer.addData("Name", "headerLeft");
        renderer.addData("Address", "headerLeft");
        renderer.addData("City", "headerLeft");
        renderer.addData("State", "headerLeft");
        renderer.addData("Zip", "headerLeft");
        renderer.addData("Contact", "headerLeft");
        renderer.nextLine();

        renderer.setHorizontalFreezePane();

        // write data
        iterator = dataset.getDatasetIterator();
        while (iterator.next()) {
                renderer.addData(iterator.getInteger("VP_CST_NBR"), "dataString");
                renderer.addData(iterator.getString("NAME"), "dataString");
                String address = "";
                String addr1 = iterator.getString("ADDR1");
                if (addr1 != null) {
                address += addr1;
                }
                String addr2 = iterator.getString("ADDR2");
                if (addr2 != null) {
                if (address.length() > 0) {
                        address + ", ";
                }
                address += addr2;
                }
                renderer.addData(address, "dataString");
                renderer.addData(iterator.getString("CITY"), "dataString");
                renderer.addData(iterator.getString("ST"), "dataString");
                renderer.addData(iterator.getString("ZIP_CD"), "dataString");
                renderer.addData(iterator.getString("CONTACT_NAME"), "dataString");
                renderer.nextLine();
        }
                ]]></value>
        </property>
</bean>

<bean id="salesTemplate" class="org.javautil.document.renderer.BshRenderTemplate">
        <property name="bshScript">
                <value><![CDATA[
        columnCount = dataset.getMetadata().getColumnCount();

        // find the first crosstabbed column
        firstCrosstabbedColumnIndex = -1;
        for (int i = 0; i < columnCount; i++) {
        metadata = dataset.getMetadata().getColumnMetaData(i);
        if (metadata.getGroupName() != null) {
                firstCrosstabbedColumnIndex = i;
                break;
                }
        }
        if (firstCrosstabbedColumnIndex == -1) {
                throw new IllegalStateException("no crosstabbed columns were found");
        }
        numberOfDataColumns = 2;
        iterator = dataset.getDatasetIterator();

        // write first row "group" headers
        for (int i = firstCrosstabbedColumnIndex; i < columnCount; i++) {
                int type = i % numberOfDataColumns;
                if (type == 1) {
                metadata = dataset.getMetadata().getColumnMetaData(i);
                String groupName = metadata.getGroupName();
                skipWidth = numberOfDataColumns - 1;
                renderer.mergeCellRange(renderer.getRowIndex(), renderer.getColumnIndex() + skipWidth);
                renderer.addData("Item " + groupName, "headerCenter");
                renderer.skip(skipWidth);
                }
        }
        renderer.nextLine();

        // write second row "detail" headers
        for (int i = firstCrosstabbedColumnIndex; i < columnCount; i++) {
                int type = i % numberOfDataColumns;
                if (type == 0) {
                content.setColumnWidth(renderer.getColumnIndex(), 6);
                renderer.addData("Cases", "headerRight");
                } else if (type == 1) {
                content.setColumnWidth(renderer.getColumnIndex(), 8);
                renderer.addData("Dollars" , "headerRight");
                } else {
                throw new IllegalStateException("unexpected data case, type " + type);
                }
        }
        renderer.nextLine();

        // write data
        while (iterator.next()) {
                for (int i = firstCrosstabbedColumnIndex; i < columnCount; i++) {
                int type = i % numberOfDataColumns;
                if (type == 0) {
                        renderer.addData(iterator.getInteger(i), "dataInteger");
                } else if (type == 1) {
                        renderer.addData(iterator.getDouble(i), "dataDollars");
                } else {
                        throw new IllegalStateException("unexpected data case, type " + type);
                }
        }
                renderer.nextLine();
        }
                ]]></value>
        </property>
</bean>

<bean id="infoRegion" class="org.javautil.document.SimpleRegion">
```

```
        <property name="name" value="info" />
        <property name="dataset" ref="infoDataset" />
        <property name="parameters" ref="queryParameters" />
        <property name="documentStyles" ref="documentStyles" />
        <property name="renderTemplate" ref="infoTemplate" />
        <property name="layoutConstraints">
                <bean class="org.javautil.document.layout.AbsoluteLayout">
                        <property name="row" value="0" />
                        <property name="column" value="0" />
                </bean>
        </property>
</bean>
<bean id="customerRegion" class="org.javautil.document.SimpleRegion">
        <property name="name" value="customer" />
        <property name="dataset" ref="reportDataset" />
        <property name="parameters" ref="queryParameters" />
        <property name="documentStyles" ref="documentStyles" />
        <property name="renderTemplate" ref="customerTemplate" />
        <property name="layoutConstraints">
                <bean class="org.javautil.document.layout.AbsoluteLayout">
                        <property name="row" value="2" />
                        <property name="column" value="0" />
                </bean>
        </property>
</bean>
<bean id="salesRegion" class="org.javautil.document.SimpleRegion">
        <property name="name" value="sales" />
        <property name="dataset" ref="reportDataset" />
        <property name="parameters" ref="queryParameters" />
        <property name="documentStyles" ref="documentStyles" />
        <property name="renderTemplate" ref="salesTemplate" />
        <property name="layoutConstraints">
                <bean class="org.javautil.document.layout.RelativeLayout">
                        <property name="rightOfRegion" ref="customerRegion" />
                </bean>
        </property>
</bean>
<bean class="org.javautil.document.DocumentBuilder">
        <property name="outputFile" value="${outputFile}" />
        <property name="deleteExistingOutputFile" value="${deleteExistingOutputFile}" />
        <property name="format" value="xls" />
        <property name="regions">
                <list>
                        <ref bean="infoRegion" />
                        <ref bean="customerRegion" />
                        <ref bean="salesRegion" />
                </list>
        </property>
</bean>
</beans>
```