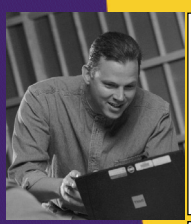
JSF: Data Tables

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/JSF-Tutorial/

This somewhat old tutorial covers JSF 1, and is left online for those maintaining existing projects. All new projects should use JSF 2, which is both simpler and more powerful. See http://www.coreservlets.com/JSF-Tutorial/jsf2/.

© 2012 Marty Hall

For live training on JSF 1.x or 2.0, please see courses at http://courses.coreservlets.com/.



Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - Java 5, Java 6, intermediate/beginning servlets/JSP, advanced servlets/JSP, Struts, JSF, Ajax, GWT, custom courses.
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, Ruby/Rails

Contact hall@coreservlets.com for details

Topics in This Section

- **Motivation**
- **Basic syntax**
- **Defining table headings**
- **Formatting tables with style sheets**
- **Displaying database tables**

Motivation: Dealing with Variable Length Data

- **Issue**
  - What if the business/data-access logic creates something with an indeterminate number of elements?
  - How do you output it in the final JSP page without breaking the MVC model?
- **Alternatives (without data tables)**
  - Non-looping
    - <h:outputText value="#{bankCustomer.depositTable}" escape="true"/>
    - <mytags:showDeposits custID="..." styleClasses="..."/>
  - Looping
    - JSTL
    - <% for(...) { ... %> HTML Code <% } %>

JSF Constructs for Handling Variable-Length Data

Simplest for Page Author

Most Control for Page Author

- Bean
- Custom tag
- h:dataTable
- t:dataList
- JSTL loop
- JSP scripting loop

Basic Syntax: h:dataTable and h:column

Building HTML Tables

- **Assume you need HTML exposed in JSP page, so you cannot use the non-looping alternatives**
  - Using JSP scripting elements to loop is unwieldy
  - Using JSTL to loop is an option, but is still complex
- **JSF provides h:dataTable**
  - You give *one* row definition, and JSF repeats it for you

```
<h:dataTable value="#{someBean.someCollection}"
             var="rowVar"
             border="1">
  <h:column>
    <h:outputText value="#{rowVar.col1Data}"/>
  </h:column>
  <h:column>
    <h:outputText value="#{rowVar.col2Data}"/>
  </h:column>
  ...
</h:dataTable>
```


Details: h:dataTable

- **value: a collection of data (list of beans, usually). Legal collection types:**
  - Array
  - List (e.g., ArrayList, LinkedList)
  - ResultSet (must be scroll-insensitive: see later section)
  - Result (wrapped ResultSet from JSTL)
  - DataModel (in javax.faces.model)
- **var: bound to each collection entry in turn**
  - This entry should be something the JSF EL can output
    - Bean, array, List, Map
- **Other attributes**
  - Standard TABLE attributes
    - border, bgcolor, width, cellpadding, cellspacing, frame, ...
  - Style sheet designators
    - rowClasses, headerClass, footerClass

Details: h:column

- **Usually encloses h:outputText elements**
  - Which reference the variable from the enclosing h:dataTable

```
<h:column>
  <h:outputText value="#{rowVar.colData}"/>
</h:column>
```

- **Can enclose other h:Xxxx elements**
  - h:outputText, h:inputText, etc.
- **Regular HTML must be enclosed in f:verbatim**

```
<h:column>
  <f:verbatim>First Name: </f:verbatim>
  <h:outputText value="#{rowVar.firstName}"/>
</h:column>
```

- **Table headings and footers specified with f:facet**
  - See later example

Example: Printing Table of Sales Based on Array

- **SalesBean class represents sales of apples and oranges (in dollars)**
- **Array of SalesBean objects represents quarterly sales in year**
- **All values to be presented in HTML table**

SalesBean: Basic Code

```
public class SalesBean {
  private double apples = 0.0, oranges = 0.0;

  public SalesBean() {}

  public SalesBean(double apples, double oranges) {
    setApples(apples);
    setOranges(oranges);
  }

  public double getApples() { return(apples); }

  public void setApples(double apples) {
    this.apples = apples;
  }

  public double getOranges() { return(oranges); }

  public void setOranges(double oranges) {
    this.oranges = oranges;
  }
```


SalesBean: Code to Create Sample Sales Array

```
public SalesBean[] getYearlySales() {
  SalesBean[] yearlySales =
    { new SalesBean(100.22, 200.32),
      new SalesBean(300.44, 400.55),
      new SalesBean(500.66, 600.77),
      new SalesBean(700.88, 800.99) };
  return(yearlySales);
}
```

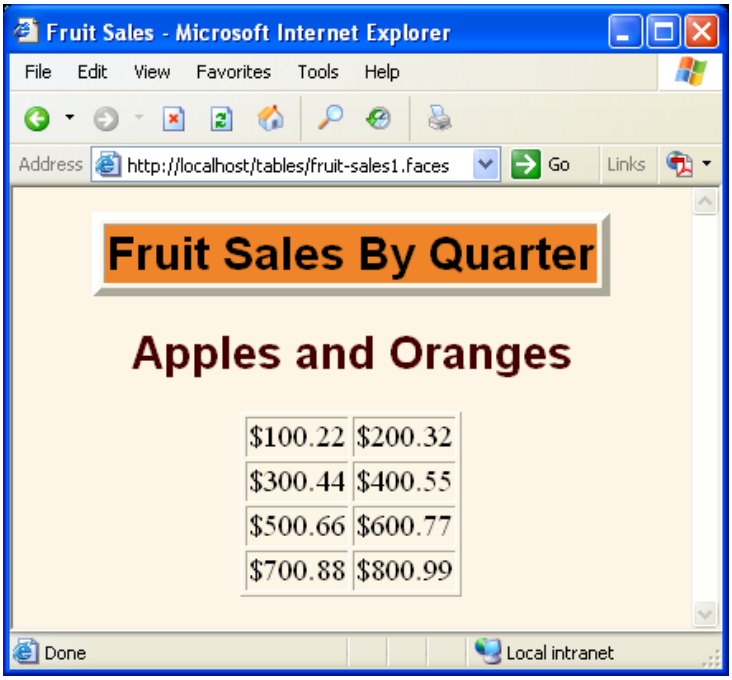
faces-config.xml

```
...
<faces-config>
  <managed-bean>
    <managed-bean-name>salesBean</managed-bean-name>
    <managed-bean-class>
      coreservlets.SalesBean
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
...
</faces-config>
```


fruit-sales1.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
...
<H2>Apples and Oranges</H2>
<h:dataTable value="#{salesBean.yearlySales}"
             var="quarterlySales"
             border="1">
  <h:column>
    <f:verbatim>$</f:verbatim>
    <h:outputText value="#{quarterlySales.apples}"/>
  </h:column>
  <h:column>
    <f:verbatim>$</f:verbatim>
    <h:outputText value="#{quarterlySales.oranges}"/>
  </h:column>
</h:dataTable>
...
</f:view>
```

fruit-sales1.faces



Defining Table Headings

Headers and Footers

- **Headers**
  - Use f:facet with name="header"
  - Value can be f:verbatim or h:outputText
  - Still need h:outputText for non-heading value
- **Footers**
  - Use f:facet with name="footer"
  - Value can be f:verbatim or h:outputText
- **Example Code**

```
<h:column>
  <f:facet name="header">
    <f:verbatim>...</f:verbatim>
  </f:facet>
  <h:outputText value="#{rowVar.colVal}"/>
</h:column>
```


Example: Displaying List of Monthly Sales

- **MonthlySalesBean class**
  - Represents a month name and a SalesBean for the apples/oranges sales in that month
- **ArrayList represents sales for January through December**
- **Want headings to be "Month", "Apples", and "Oranges"**
- **Also want to use textfields to display sales numbers, so that they can be edited**
  - Point is to illustrate that tables can contain input elements
  - Backend code to process results omitted in this example

MonthlySalesBean: Basic Code

```
public class MonthlySalesBean {
  private String month;
  private SalesBean salesBean;

  public MonthlySalesBean() {}

  public MonthlySalesBean(String month,
                          double apples,
                          double oranges) {
    setMonth(month);
    setSalesBean(new SalesBean(apples, oranges));
  }

  public String getMonth() { return(month); }

  public void setMonth(String month) {
    this.month = month; }

  public SalesBean getSalesBean() {
    return(salesBean);
  }
  public void setSalesBean(SalesBean salesBean) {
    this.salesBean = salesBean; }
```


MonthlySalesBean: Code to Give List of Sales Figures

```
public List<MonthlySalesBean> getYearlySales() {
   List<MonthlySalesBean> yearlySales =
     new ArrayList<MonthlySalesBean>();
   yearlySales.add
     (new MonthlySalesBean("January", 101.11, 501.11));
   yearlySales.add
     (new MonthlySalesBean("February", 102.22, 502.22));
   yearlySales.add
     (new MonthlySalesBean("March", 103.33, 503.33));
   ...
   yearlySales.add
     (new MonthlySalesBean("November", 111.11, 511.11));
   yearlySales.add
     (new MonthlySalesBean("December", 112.22, 512.22));
   return(yearlySales);
 }
```

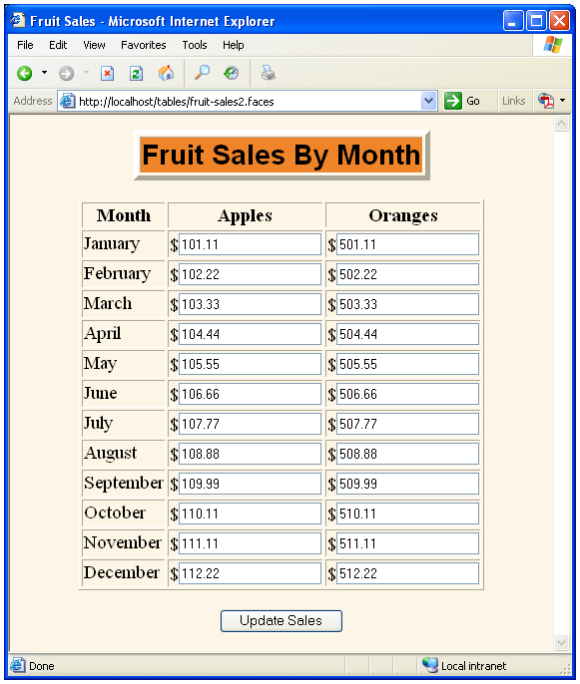
faces-config.xml

```
...
<faces-config>
  <managed-bean>
    <managed-bean-name>
      monthlySalesBean
    </managed-bean-name>
    <managed-bean-class>
      coreservlets.MonthlySalesBean
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
...
</faces-config>
```


fruit-sales2.jsp

```
<h:form>
<h:dataTable value="#{monthlySalesBean.yearlySales}"
             var="monthlySales"
             border="1">
  <h:column>
    <f:facet name="header">
      <f:verbatim>Month</f:verbatim>
    </f:facet>
    <h:outputText value="#{monthlySales.month}"/>
  </h:column>
  <h:column>
    <f:facet name="header">
      <f:verbatim>Apples</f:verbatim>
    </f:facet>
    <f:verbatim>$</f:verbatim>
    <h:inputText
        value="#{monthlySales.salesBean.apples}"/>
  </h:column>
  ... Oranges ...
</h:dataTable>
</h:form>
```

fruit-sales2.faces



Formatting Tables with Style Sheets

Options

- **Use explicit formatting in JSP**
  - Requires f:verbatim before and after each entry
  - Long and clumsy
- **Embed formatting in bean**
  - Not accessible to Web developer
  - Risks inconsistencies with style sheet
  - Beans are for model, not view
- **Use rowClasses, headerClass, footerClass**
  - **rowClasses**: comma-separated list of CSS styles. Applied to each row until list ends, then repeats
  - **headerClass**: CSS style for heading
  - **footerClass**: CSS style for footer

Example: Formatted Table of Monthly Sales

- **Same MonthlySalesBean**
- **Same faces-config.xml entry**
- **Style applied to heading**
- **Two styles applied to rows**
  - Results in alternating styles

Style Sheet (styles.css)

```
.HEADING {
  font-family: Arial, Helvetica, sans-serif;
  font-weight: bold;
  font-size: 20px;
  color: black;
  background-color: silver;
  text-align: center;
}
```

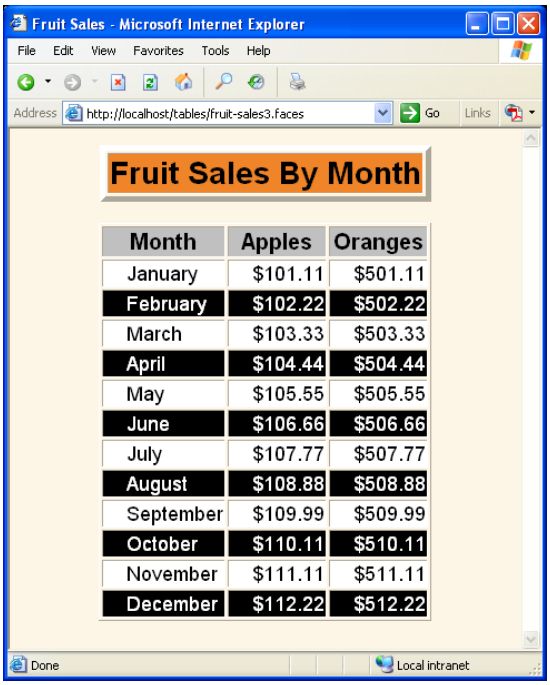

Style Sheet (styles.css)

```
.ROW1 {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 18px;
  color: black;
  background-color: white;
  text-indent: 20px;
}
.ROW2 {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 18px;
  color: white;
  background-color: black;
  text-indent: 20px;
}
```

fruit-sales.jsp

```
<h:dataTable value="#{monthlySalesBean.yearlySales}"
             var="monthlySales"
             border="1"
             headerClass="HEADING"
             rowClasses="ROW1,ROW2">
  <h:column>
    <f:facet name="header">
      <f:verbatim>Month</f:verbatim>
    </f:facet>
    <h:outputText value="#{monthlySales.month}"/>
  </h:column>
  <h:column>
    ... Apples ...
  </h:column>
  <h:column>
    ... Oranges ...
  </h:column>
</h:dataTable>
```


fruit-sales3.faces



Database-Driven Tables

Options for Displaying Database Results

- **Extract data from result set and place in array or List**
  - Tedious
  - Requires a bean to represent a row of data
- **Use ResultSet as arg to value of h:dataTable**
  - Should work, according to spec
    - In practice, you must use scroll-insensitive or it crashes
  - ResultSet is connected, so a big pain to go back and close connections (or return them to pool) later
- **Use JSTL ResultSupport class to turn ResultSet into Result**
  - Only one extra line of code
  - Robust and reliable
  - Result is disconnected, so you can close connection or return it to pool before using the Result

Example: Display Customers from Northwind Database

- Store connection info in server.xml for Tomcat
- Retrieve Connection from JNDI
- Wrap ResultSet in a Result
- Pass Result to h:dataTable
- Use column names instead of bean property names

NorthwindTableBean: Basic Code

```
package coreservlets;
import java.sql.*;
import javax.sql.*;
import javax.naming.*;
import javax.servlet.jsp.jstl.sql.*;

public class NorthwindTableBean {
  private String tableName = "customers";

  public String getTableName() {
    return(tableName);
  }

  public void setTableName(String tableName) {
    this.tableName = tableName;
  }
```

NorthwindTableBean: Code to Perform Query and Return Result

```
public Result getTable() {
  try {
    Context context = new InitialContext();
    DataSource dataSource =
      (DataSource)context.lookup
         ("java:comp/env/jdbc/Northwind");
    Connection connection =
      dataSource.getConnection();
    Statement statement =
      connection.createStatement();
    String query = "SELECT * from " + getTableName();
    ResultSet resultSet =
      statement.executeQuery(query);
    return(ResultSupport.toResult(resultSet));
  } catch(Exception e) { return(null); }
}
```

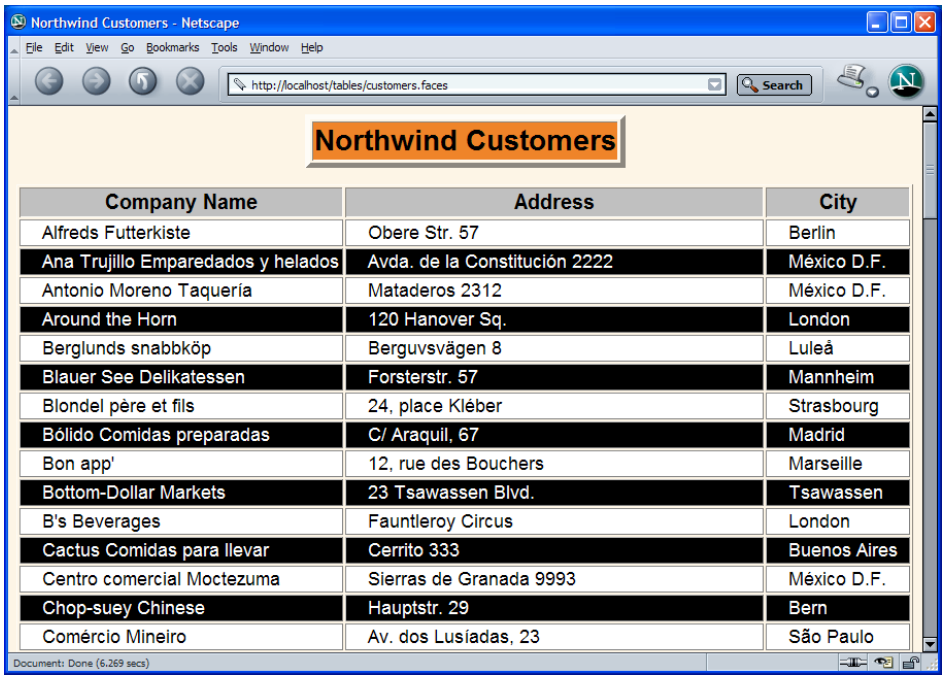

faces-config.xml

```
...
<faces-config>
  <managed-bean>
    <managed-bean-name>
      northwindBean
    </managed-bean-name>
    <managed-bean-class>
      coreservlets.NorthwindTableBean
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>...
</faces-config>
```

customers.jsp

```
<h:dataTable value="#{northwindBean.table}"
             var="dbRow"
             border="1"
             headerClass="HEADING"
             rowClasses="ROW1,ROW2">
  <h:column>
    <f:facet name="header">
      <f:verbatim>Company Name</f:verbatim>
    </f:facet>
    <h:outputText value="#{dbRow.CompanyName}"/>
  </h:column>
  <h:column>
    <f:facet name="header">
      <f:verbatim>Address</f:verbatim>
    </f:facet>
    <h:outputText
        value="#{dbRow.Address}"/>
  </h:column>
  ... City ...
</h:dataTable>
```


customers.faces



Summary

- **<h:dataTable value="*collection*" var="*variable*" ...>**
  - Collection usually array, List, or Result
  - HTML-oriented attributes:
    - border, bgcolor, width, etc.
  - Style sheet designators
    - rowClasses, headerClass, footerClass
- **<h:column>...</h:column>**
  - Encloses code that refers to variable
    - Usually h:outputText, but can be any JSF component
  - f:verbatim for literal text
  - f:facet for headings and footer

Questions?