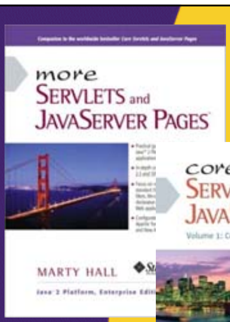
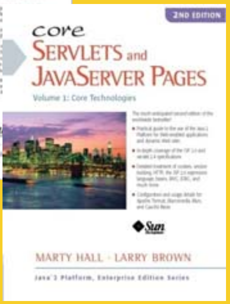
PrimeFaces: Extended Data Tables

Originals of slides and source code for examples: http://www.coreservlets.com/JSF-Tutorial/primefaces/
Also see the JSF 2 tutorial – http://www.coreservlets.com/JSF-Tutorial/jsf2/
and customized JSF2 and PrimeFaces training courses – http://courses.coreservlets.com/jsf-training.html





Topics in This Section

- Overview
- Basics
- Column headers
- Resizable columns
- Row styling
- Sortable tables
- Pagination

Overview

Motivation

- **For data tables in general**
  - When you have variable-length data to display in the results page
    - E.g., if you want to show deposits and withdrawals in last month rather than just current balance
- **For PrimeFaces extended version**
  - h:dataTable is popular and easy to use
  - But, important features are either lacking or very difficult to implement
    - Skinning/theming
    - Sorting by given column attributes
    - Paging when number of entries is long
- **Most common/important options shown**
  - *Many* more options. See User's Guide for details.

Simplified Testing

- **Real life**
  - Normally, the data is produced in the business logic that is called by the action controller
    - E.g., you collect a bank customer ID and month in a form, and the button says
      <h:commandButton … action="#{user.findChanges}"/>
      where findChanges finds the bank account changes (deposits and withdrawals) in the month and puts then into an array or List
- **For practice**
  - Here, we will hardcode the data for simplicity
    - I.e., make a managed bean with a method that returns a fixed List

Simplifying Testing: Example

Bean	Standalone Test Page

```
@ManagedBean
public class Test {
  private List<Person> people =
    ... ;

  public List<Person> getPeople() {
    return(people);
  }
}
```
	```
	<!DOCTYPE ...>
	...
	<p:dataTable var="person"
	             value="#{test.people}">
	  ...
	</p:dataTable>
	...
	```

p:dataTable Basics

Simplest Syntax

- **Attributes**
  - var, value
- **Nested element**
  - p:column
- **Example**

```
<p:dataTable var="someVar" value="#{bean.someCollection}">
   <p:column>#{someVar.property1}</p:column>
   <p:column>#{someVar.property2} </p:column>
   …
</p:dataTable>
```

- **Legal types for “value” attribute**
  - List, Array, Collection, ResultSet, Result, DataModel, LazyDataModel
    - Some features (e.g., sorting) work only with List, so routinely use List instead of array

Supporting Class: Programmer

```
public class Programmer {
  private String firstName, lastName, level;
  private double salary;
  private String[] languages;

  public Programmer(String firstName,
                    String lastName,
                    String level,
                    double salary,
                    String... languages) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.level = level;
    this.salary = salary;
    this.languages = languages;
  }

  // getFirstName, getLastName, getLevel, getSalary,
  // getLanguages
```

Programmer (Continued)

```
/** Formats the salary like "$24,567.89". */

 public String getFormattedSalary() {
   return(String.format("$%,.2f", salary));
 }

 /** Returns a String like "Java, C++, and Lisp".
  *  That is, it puts commas and the word "and" into list.
  */
 public String getLanguageList() {
   StringBuilder langList = new StringBuilder();
   for(int i=0; i<languages.length; i++) {
     if(i < (languages.length-1)) {
       langList.append(languages[i] + ", ");
     } else {
       langList.append("and " + languages[i]);
     }
   }
   return(langList.toString());
 }
```

Supporting Class: Company

```
public class Company {
  private String companyName;
  private List<Programmer> programmers;

  public Company(String companyName,
                 Programmer... programmers) {
    this.companyName = companyName;
    this.programmers = Arrays.asList(programmers);
  }

  public String getCompanyName() {
    return(companyName);
  }

  public List<Programmer> getProgrammers() {
    return(programmers);
  }
}
```

Managed Bean: Company1

In early examples, data never changes, so it could be application scoped. But, in later examples, we will sort the Programmer array and we want that to persist (and not to affect other users).

```
@ManagedBean
@SessionScoped
public class Company1 extends Company {
  public Company1() {
    super("My-Small-Company.com",
        new Programmer("Larry", "Ellison", "Junior", 34762.52,
                       "SQL", "Prolog", "OCL", "Datalog"),
        new Programmer("Larry", "Page", "Junior", 43941.86,
                       "Java", "C++", "Python", "Go"),
        new Programmer("Steve", "Ballmer", "Intermediate", 83678.29,
                       "Visual Basic", "VB.NET", "C#", "Visual C++",
                       "Assembler"),
        new Programmer("Sam", "Palmisano", "Intermediate", 96550.03,
                       "REXX", "CLIST", "Java", "PL/I", "COBOL"),
        new Programmer("Steve", "Jobs", "Intermediate", 103488.80,
                       "Objective-C", "AppleScript", "Java", "Perl",
                       "Tcl"));
  }
}
```

Facelets Page: Top

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui">
<h:head><title>p:dataTable Basics</title>
</h:head>
<h:body>
```

This first simple example doesn't use any f: tags, but many real p:dataTable examples use f:facet for the overall table heading (caption). So, plan ahead and add this namespace from the beginning.

You probably are already using this namespace for p:commandButton, p:inputText, etc. But even if not, you need it for p:dataTable and p:column.

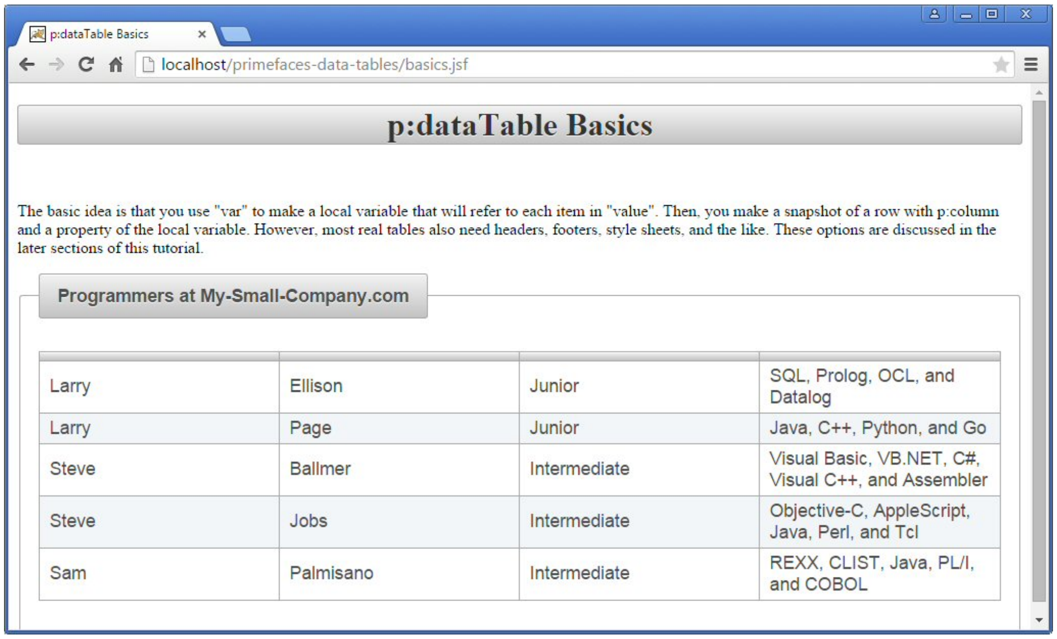
Facelets Page: Main Code

```
<p:fieldset
   legend="Programmers at #{company1.companyName}">
<br/>

<p:dataTable var="programmer"
             value="#{company1.programmers}">
  <p:column>#{programmer.firstName}</p:column>
  <p:column>#{programmer.lastName}</p:column>
  <p:column>#{programmer.level}</p:column>
  <p:column>#{programmer.languageList}</p:column>
</p:dataTable>

<br/>
</p:fieldset>
```

Results



p:dataTable Basics

The basic idea is that you use "var" to make a local variable that will refer to each item in "value". Then, you make a snapshot of a row with p:column and a property of the local variable. However, most real tables also need headers, footers, style sheets, and the like. These options are discussed in the later sections of this tutorial.

Programmers at My-Small-Company.com

Larry	Ellison	Junior	SQL, Prolog, OCL, and Datalog
Larry	Page	Junior	Java, C++, Python, and Go
Steve	Ballmer	Intermediate	Visual Basic, VB.NET, C#, Visual C++, and Assembler
Steve	Jobs	Intermediate	Objective-C, AppleScript, Java, Perl, and Tcl
Sam	Palmisano	Intermediate	REXX, CLIST, Java, PL/I, and COBOL

Column and Table Headers

Column Headers

- **Problem**
  – Regular content gives a snapshot of a row. So, most content inside p:column is repeated for every row.
- **Solution**
  – Mark headings with headerText. Shown for 1st row only.

```
<p:dataTable var="someVar" value="#{someCollection}">
    <p:column headerText="First Heading">
      #{someVar.property1}
    </p:column>
    …
</p:dataTable>
```

You can also use the standard JSF style of f:facet with name="header". But, that is a lot more cumbersome.

- **Footers**
  – Use footerText

Table Header (Caption)

- **Problems**
  – A regular heading above the table does not take on look and feel of the table.
  – p:dataTable has no support for <caption>
- **Solution**
  – Mark overall table heading with f:facet with name="header". This is *before* the first p:column.

```
<p:dataTable var="someVar" value="#{someCollection}">
    <f:facet name="header">Table Header</f:facet>
    <p:column headerText="First Heading">
      #{someVar.property1}
    </p:column>
    …
</p:dataTable>
```

This is a reserved name, not something arbitrary.
So, it will fail if you use "heading", "Header", "headers", etc.

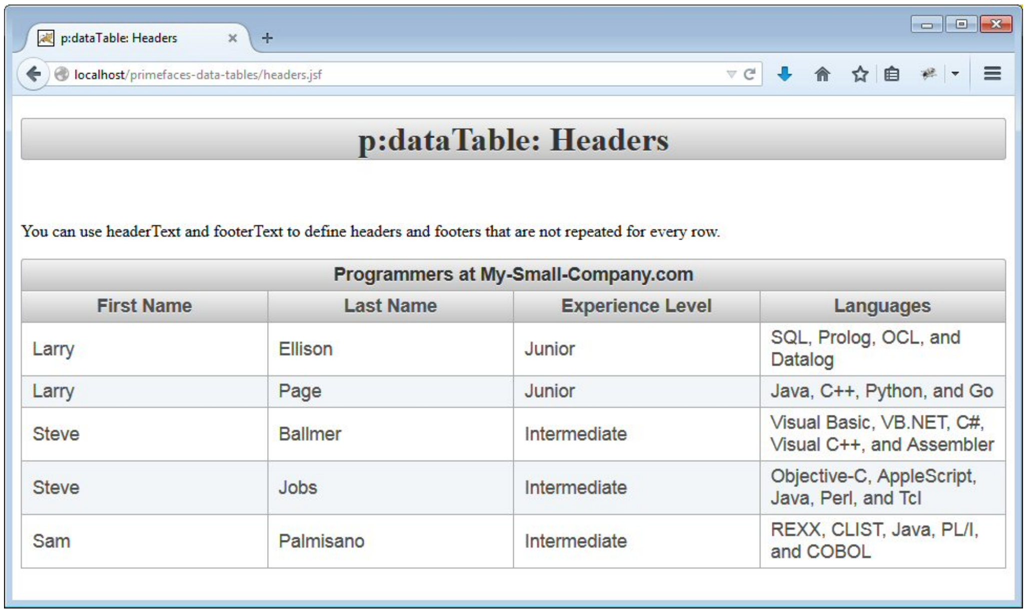
It is surprising that PrimeFaces built in a headerText shortcut for each p:column, but failed to build in a similar shortcut for p:dataTable.

Headers: Facelets

```
<p:dataTable var="programmer"
             value="#{company1.programmers}">
  <f:facet name="header">
    Programmers at #{company1.companyName}
  </f:facet>
  <p:column headerText="First Name">
    #{programmer.firstName}
  </p:column>
  <p:column headerText="Last Name">
    #{programmer.lastName}
  </p:column>
  <p:column headerText="Experience Level">
    #{programmer.level}
  </p:column>
  <p:column headerText="Languages">
    #{programmer.languageList}
  </p:column>
</p:dataTable>
```

Note: same managed bean (Company1) and supporting class (Programmer) as previous example.

Results


p:dataTable: Headers

localhost/primefaces-data-tables/headers.jsf

p:dataTable: Headers

You can use headerText and footerText to define headers and footers that are not repeated for every row.

Programmers at My-Small-Company.com			
First Name	**Last Name**	**Experience Level**	**Languages**
Larry	Ellison	Junior	SQL, Prolog, OCL, and Datalog
Larry	Page	Junior	Java, C++, Python, and Go
Steve	Ballmer	Intermediate	Visual Basic, VB.NET, C#, Visual C++, and Assembler
Steve	Jobs	Intermediate	Objective-C, AppleScript, Java, Perl, and Tcl
Sam	Palmisano	Intermediate	REXX, CLIST, Java, PL/I, and COBOL

Row Styling

Styling Table as a Whole

- **tableStyleClass**
  - styleClass applies to the entire component, whose outer element is a div
  - tableStyleClass applies to the table itself
- **Caution**
  - Large-scale changes to look of the table can make it hard to adapt to a new PrimeFaces theme

Styling Rows

- **Styling rows consistently**
  - Problem: no rowClasses
    - Standard JSF supports rowClasses, which lets you specify sets of styles that apply to n rows and then repeat
  - Solution: use CSS3 positional selectors
    - For every-other row, use ".tableClass tr:nth-child(odd) td" ".tableClass tr:nth-child(even) td"
- **Styling rows dynamically**
  - Use rowStyleClass
    - It is JSF EL expression that is re-evaluated for *each* row
    - Example:

```
<p:dataTable … rowStyleClass="#{bean.test ? 'style1' : 'style2'}">
```

Consistent Row Styling: Example

- **Big idea: zebra striping**
  - Make every-other row have a different background color
- **Approach**
  - Use tableStyleClass to give overall table a style of "zebra"
  - Style odd-numbered rows with positional selector

```
table.zebra tr:nth-child(odd) td {
    background-color: #dddddd;
}
```

Consistent Row Styling: CSS

- **CSS file**
  - WebContent/resources/css/table-row-styles.css

```
table.zebra tr:nth-child(odd) td {
  background-color: #dddddd;
}
.green { color: green }
.red { color: red }
```

- **Facelets page**

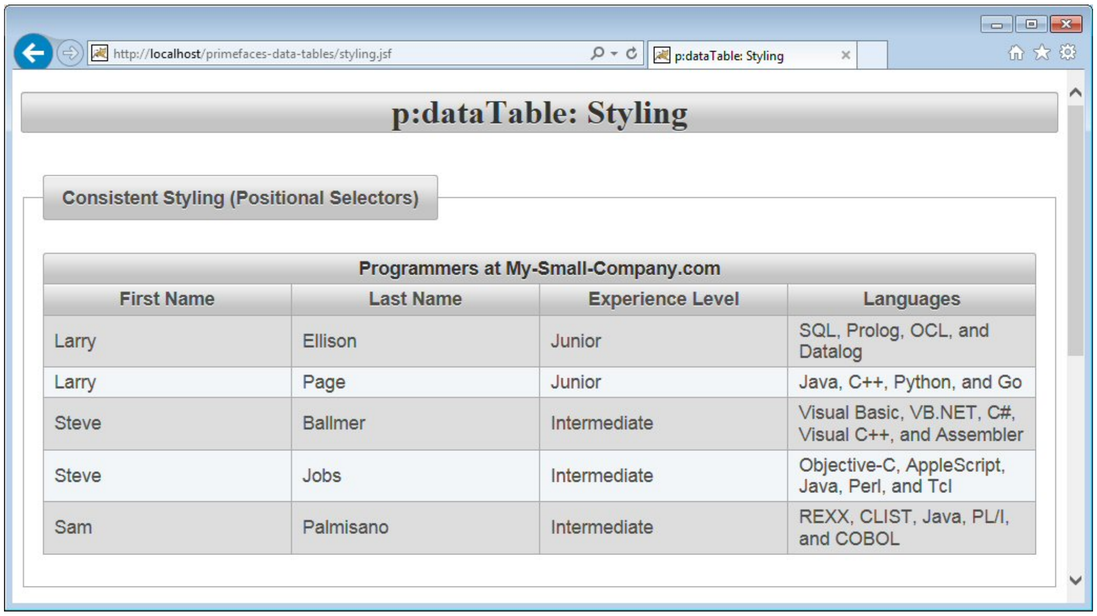
```
<h:head><title>p:dataTable: Styling</title>
<h:outputStylesheet name="table-row-styles.css"
                              library="css"/>
</h:head>
```

Consistent Row Styling: Facelets

```
<p:dataTable var="programmer" value="#{company1.programmers}"
             tableStyleClass="zebra">
  <f:facet name="header">Programmers at #{company1.companyName}</f:facet>
  <p:column headerText="First Name">#{programmer.firstName}</p:column>
  <p:column headerText="Last Name">#{programmer.lastName}</p:column>
  <p:column headerText="Experience Level">#{programmer.level}</p:column>
  <p:column headerText="Languages">#{programmer.languageList}</p:column>
</p:dataTable>
```

Note: same managed bean (Company1) and supporting class (Programmer) as earlier examples.

Consistent Row Styling: Results



p:dataTable: Styling

Consistent Styling (Positional Selectors)

Programmers at My-Small-Company.com

First Name	Last Name	Experience Level	Languages
Larry	Ellison	Junior	SQL, Prolog, OCL, and Datalog
Larry	Page	Junior	Java, C++, Python, and Go
Steve	Ballmer	Intermediate	Visual Basic, VB.NET, C#, Visual C++, and Assembler
Steve	Jobs	Intermediate	Objective-C, AppleScript, Java, Perl, and Tcl
Sam	Palmisano	Intermediate	REXX, CLIST, Java, PL/I, and COBOL

Dynamic Row Styling: Example

- **Big idea: color based on bean property**
  - Use green text for Java developers
  - Use red text for non-Java developers
- **Approach**
  - Give Programmer an isJavaDeveloper() method
  - Create two styles, “red” and “green”
  - Use rowStyleClass to dynamically assign the appropriate style

```
<p:dataTable …
      rowStyleClass="#{programmer.javaDeveloper ? 'green' : 'red'}">
```

Consistent Row Styling: CSS

- **CSS file**
  - WebContent/resources/css/table-row-styles.css

```
table.zebra tr:nth-child(odd) td {
  background-color: #dddddd;
}
.green { color: green }
.red { color: red }
```

- **Facelets page**

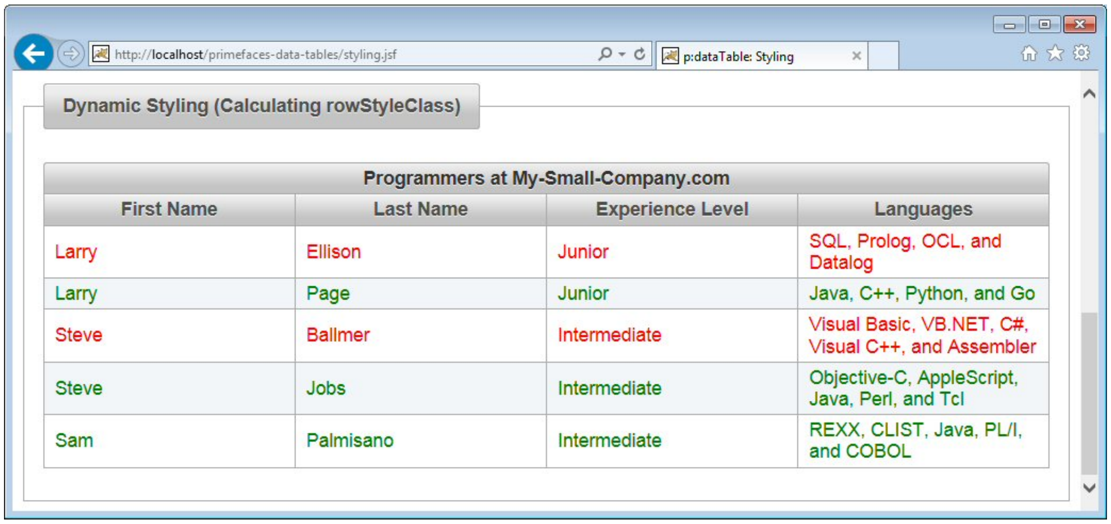
```
<h:head><title>p:dataTable: Styling</title>
<h:outputStylesheet name="table-row-styles.css"
                    library="css"/>
</h:head>
```

Dynamic Row Styling: Facelets

```
<p:dataTable var="programmer" value="#{company1.programmers}"
           rowStyleClass="#{programmer.javaDeveloper ? 'green' : 'red'}">
  <f:facet name="header">Programmers at #{company1.companyName}</f:facet>
  <p:column headerText="First Name">#{programmer.firstName}</p:column>
  <p:column headerText="Last Name">#{programmer.lastName}</p:column>
  <p:column headerText="Experience Level">#{programmer.level}</p:column>
  <p:column headerText="Languages">#{programmer.languageList}</p:column>
</p:dataTable>
```

Note: same managed bean (Company1) and supporting class (Programmer) as earlier examples.

Consistent Row Styling: Results



Dynamic Styling (Calculating rowStyleClass)

Programmers at My-Small-Company.com

First Name	Last Name	Experience Level	Languages
Larry	Ellison	Junior	SQL, Prolog, OCL, and Datalog
Larry	Page	Junior	Java, C++, Python, and Go
Steve	Ballmer	Intermediate	Visual Basic, VB.NET, C#, Visual C++, and Assembler
Steve	Jobs	Intermediate	Objective-C, AppleScript, Java, Perl, and Tcl
Sam	Palmisano	Intermediate	REXX, CLIST, Java, PL/I, and COBOL

Resizable Columns

Big Idea

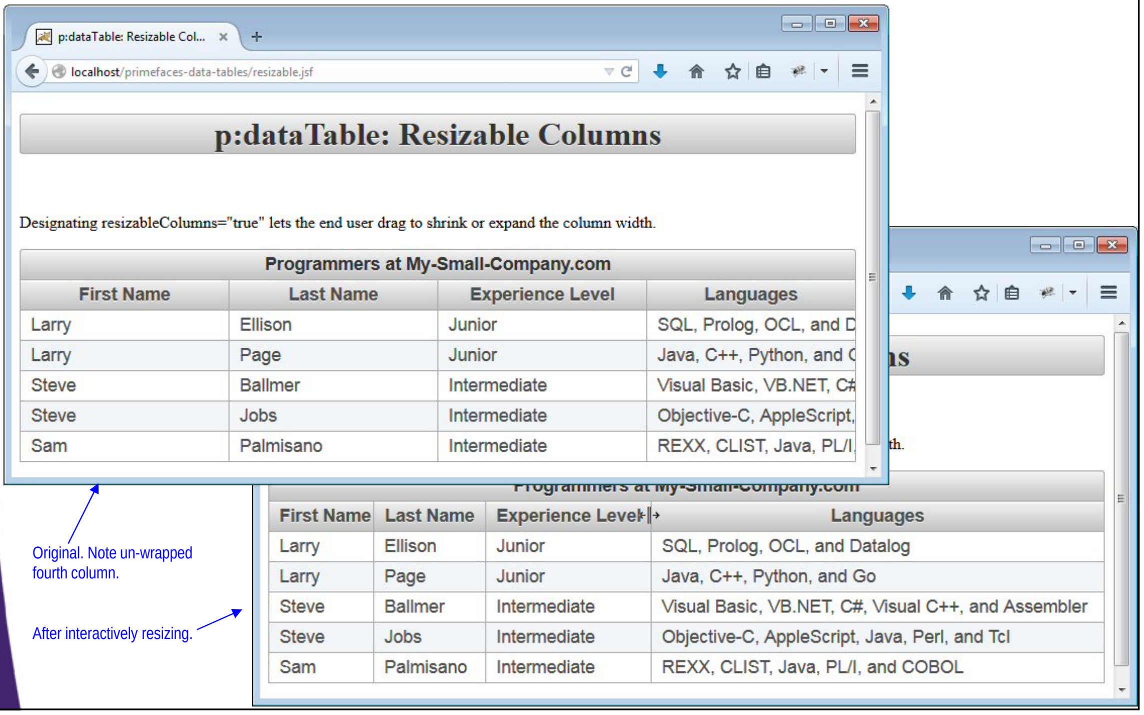
- **Default behavior**
  - Each column gets an equal width
- **Can let user drag columns wider or more narrow**
  - Use <p:dataTable … resizableColumns="true">
- **Major deficiency**
  - Long table cells do not wrap when resize is turned on

Facelets

```
<p:dataTable var="programmer" value="#{company1.programmers}"
             resizableColumns="true">
  <f:facet name="header">Programmers at #{company1.companyName}</f:facet>
  <p:column headerText="First Name">#{programmer.firstName}</p:column>
  <p:column headerText="Last Name">#{programmer.lastName}</p:column>
  <p:column headerText="Experience Level">#{programmer.level}</p:column>
  <p:column headerText="Languages">#{programmer.languageList}</p:column>
</p:dataTable>
```

Note: same managed bean (Company1) and supporting class (Programmer) as earlier examples.

Results


p:dataTable: Resizable Col...
localhost/primefaces-data-tables/resizable.jsf

p:dataTable: Resizable Columns

Designating resizableColumns="true" lets the end user drag to shrink or expand the column width.

Programmers at My-Small-Company.com

First Name	Last Name	Experience Level	Languages
Larry	Ellison	Junior	SQL, Prolog, OCL, and D
Larry	Page	Junior	Java, C++, Python, and C
Steve	Ballmer	Intermediate	Visual Basic, VB.NET, C#
Steve	Jobs	Intermediate	Objective-C, AppleScript,
Sam	Palmisano	Intermediate	REXX, CLIST, Java, PL/I,

First Name	Last Name	Experience Level	Languages
Larry	Ellison	Junior	SQL, Prolog, OCL, and Datalog
Larry	Page	Junior	Java, C++, Python, and Go
Steve	Ballmer	Intermediate	Visual Basic, VB.NET, C#, Visual C++, and Assembler
Steve	Jobs	Intermediate	Objective-C, AppleScript, Java, Perl, and Tcl
Sam	Palmisano	Intermediate	REXX, CLIST, Java, PL/I, and COBOL

Original. Note un-wrapped fourth column.

After interactively resizing.

Sortable Tables

Big Idea

- **Default behavior**
  – Rows are shown in the order they appear in the data
- **Can let user click to sort based on column**
  – Use <p:column … sortBy="#{bean.property}">
    - Uses Ajax: sorts List on the server
- **Sorting uses standard Java comparisons**
  – Alphabetical for String properties
  – By number value for numeric properties
  – Can override comparison with sortFunction
    - As with normal Java compare function, returns -1, 0, or 1 depending if first arg should be earlier, tied, or later
- **Must use List (or ListModel)**
  – Main table data source must be List, not array

Undocumented Requirements

- **Must use List (or ListModel)**
  - The type for “value” must be List, not array
- **You must enclose table in h:form**

```
<h:form>
<p:dataTable …>…</p:dataTable>
</h:form>
```

  - This is ostensibly because clicks result in Ajax calls, but pagination also uses Ajax and does not require h:form
- **Sorting and pagination do not mix**
  - Tables with sorting cannot use pagination (next topic)
- **None of these requirement are documented**
  - Not mentioned in 5.1 User’s Guide
  - Not explicitly stated in Showcase

Example

- **When table first comes up**
  - In order that programmers appear in original data
- **When user clicks on first name, last name, or experience level**
  - Sorts alphabetically by that property
    - Uses sortBy
  - Clicking a second time reverses the order
- **When user clicks on languages**
  - Sorts by length
    - Uses sortBy and sortFunction
  - Clicking a second time reverses the order

Facelets

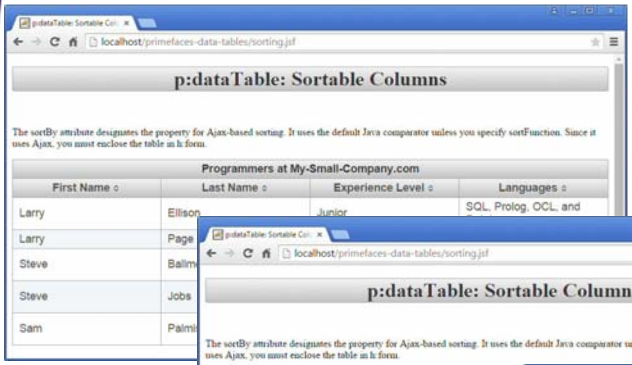
```
<h:form>
<p:dataTable var="programmer" value="#{company1.programmers}">
  <f:facet name="header">Programmers at #{company1.companyName}</f:facet>
  <p:column headerText="First Name" sortBy="#{programmer.firstName}">
    #{programmer.firstName}
  </p:column>
  <p:column headerText="Last Name" sortBy="#{programmer.lastName}">
    #{programmer.lastName}
  </p:column>
  <p:column headerText="Experience Level" sortBy="#{programmer.level}">
    #{programmer.level}
  </p:column>
  <p:column headerText="Languages" sortBy="#{programmer.languageList}"
            sortFunction="#{sorter.compareLength}">
    #{programmer.languageList}
  </p:column>
</p:dataTable>
</h:form>
```

Note: same managed bean (Company1) and supporting class (Programmer) as earlier examples.

Java: Comparison Function For Languages Column

```
@ManagedBean
@ApplicationScoped
public class Sorter {
  public int compareLength(String s1, String s2) {;
    return(s2.length() - s1.length());
  }
}
```

Results

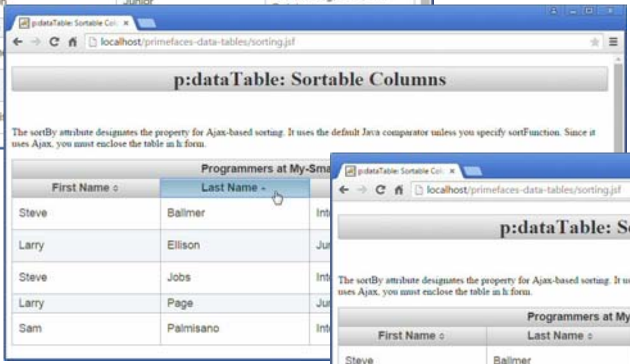

localhost/primefaces-data-tables/sorting.jsf

p:dataTable: Sortable Columns

The sortBy attribute designates the property for Ajax-based sorting. It uses the default Java comparator unless you specify sortFunction. Since it uses Ajax, you must enclose the table in h:form.

Programmers at My-Small-Company.com			
First Name ⇕	Last Name ⇕	Experience Level ⇕	Languages ⇕
Larry	Ellison	Junior	SQL, Prolog, OCL, and
Larry	Page		
Steve	Ballm		
Steve	Jobs		
Sam	Palmi		

Original

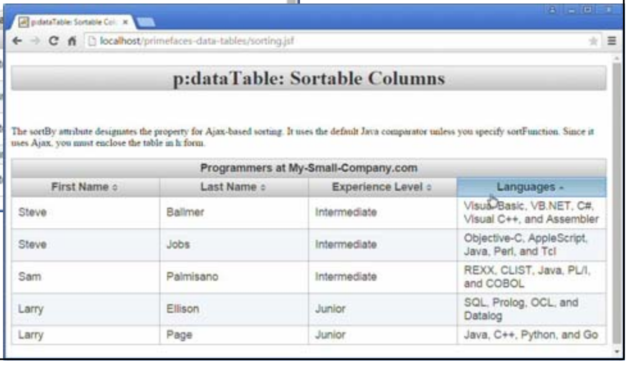

localhost/primefaces-data-tables/sorting.jsf

p:dataTable: Sortable Columns

The sortBy attribute designates the property for Ajax-based sorting. It uses the default Java comparator unless you specify sortFunction. Since it uses Ajax, you must enclose the table in h:form.

Programmers at My-Sma		
First Name ⇕	Last Name ⌄	
Steve	Ballmer	Int
Larry	Ellison	Ju
Steve	Jobs	Int
Larry	Page	Ju
Sam	Palmisano	Int

Sorted alphabetically by last name


localhost/primefaces-data-tables/sorting.jsf

p:dataTable: Sortable Columns

The sortBy attribute designates the property for Ajax-based sorting. It uses the default Java comparator unless you specify sortFunction. Since it uses Ajax, you must enclose the table in h:form.

Programmers at My-Small-Company.com			
First Name ⇕	Last Name ⇕	Experience Level ⇕	Languages ⌄
Steve	Ballmer	Intermediate	Visual Basic, VB.NET, C#, Visual C++, and Assembler
Steve	Jobs	Intermediate	Objective-C, AppleScript, Java, Perl, and Tcl
Sam	Palmisano	Intermediate	REXX, CLIST, Java, PL/I, and COBOL
Larry	Ellison	Junior	SQL, Prolog, OCL, and Datalog
Larry	Page	Junior	Java, C++, Python, and Go

Sorted by length of language list

Pagination

Big Idea

- **Default behavior**
  - All rows are shown
  - Can limit the rows with rows="*n*"
- **Can show limited number and let user page**
  - Use <p:dataTable … rows="*n*" paginator="true">
    - Uses Ajax to fetch new data from server
- **Still loads entire data into server memory**
  - But lazy loading also supported
    - Facelets use p:dataTable with rows="n", paginator="true", and lazy="true“
    - Java data source then must implement LazyDataModel
- **Warning: Do not paginate sortable tables**
    - Resets to first page when you sort, but then sorting is lost on following pages (at least as of PrimeFaces 5.1)

Pagination Options

- **Basic**
  - rows="*n*" and paginator="true"
- **Position of paginator controls**
  - paginatorPosition (top, bottom, both [default])
- **Controls**
  - paginatorTemplate
    - Can specify which controls appear, and in which order. For details, see pagination section of DataGrid (not DataTable) in User’s Manual.
  - pageLinks
    - Max number of links to display [default 10]
  - paginatorAlwaysVisible [default true]
    - Determines if paginator controls should be hidden if total number of rows is less than number of rows per page

Example

- **Total data length**
  - 50
- **Number of rows shown at a time**
  - 10
- **Default paginator position**
  - Both top and bottom

Facelets

```
<p:dataTable var="programmer" value="#{company2.programmers}"
             rows="10" paginator="true">
  <f:facet name="header">Programmers at #{company2.companyName}</f:facet>
  <p:column headerText="First Name">#{programmer.firstName}</p:column>
  <p:column headerText="Last Name">#{programmer.lastName}</p:column>
  <p:column headerText="Experience Level">#{programmer.level}</p:column>
  <p:column headerText="Languages">#{programmer.languageList}</p:column>
</p:dataTable>
```

Managed Bean

```
@ManagedBean
@SessionScoped
public class Company2 extends Company
       implements Serializable {
  public Company2() {
    super("Some Random Company",
          Programmers.makeProgrammers(50));
  }
}
```

Helper Class to Make Programmers (Part 1)

```
public class Programmers {
  private static String[] levels =
    { "Beginner", "Junior", "Intermediate",
      "Senior", "Super Hacker" };

  public static Programmer makeProgrammer(int i) {
    String firstName = "Firstname" + i;
    String lastName = "Lastname" + i;
    String level = RandomUtils.randomElement(levels);
    double salary = Math.random() * 1_000_000;
    String[] languages =
      { "LanguageA" + i, "LanguageB" + i,
        "LanguageC" + i };
    return(new Programmer(firstName, lastName,
                          level, salary, languages));
  }
```

Helper Class to Make Programmers (Part 2)

```
public static Programmer[] makeProgrammers(int count) {
  Programmer[] programmers = new Programmer[count];
  for(int i=0; i<programmers.length; i++) {
    programmers[i] = makeProgrammer(i);
  }
  return(programmers);
}

...
```

Helper Class to Choose from Array at Random

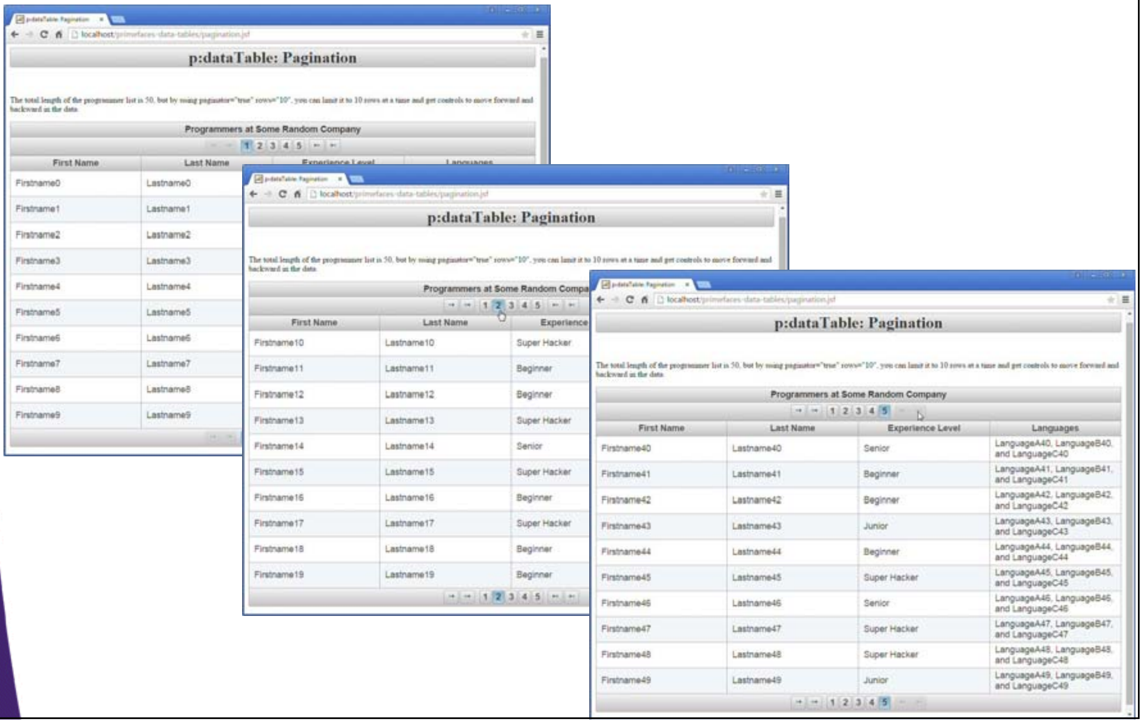
```
public class RandomUtils {
  private static Random r = new Random();

  public static int randomInt(int range) {
    return(r.nextInt(range));
  }

  public static int randomIndex(Object[] array) {
    return(randomInt(array.length));
  }

  public static <T> T randomElement(T[] array) {
    return(array[randomIndex(array)]);
  }
}
```

Results


localhost/primefaces-data-tables/pagination.jsf

p:dataTable: Pagination

The total length of the programmer list is 50, but by using paginator="true" rows="10", you can limit it to 10 rows at a time and get controls to move forward and backward in the data.

Programmers at Some Random Company

1 2 3 4 5

First Name	Last Name
Firstname0	Lastname0
Firstname1	Lastname1
Firstname2	Lastname2
Firstname3	Lastname3
Firstname4	Lastname4
Firstname5	Lastname5
Firstname6	Lastname6
Firstname7	Lastname7
Firstname8	Lastname8
Firstname9	Lastname9

localhost/primefaces-data-tables/pagination.jsf

p:dataTable: Pagination

The total length of the programmer list is 50, but by using paginator="true" rows="10", you can limit it to 10 rows at a time and get controls to move forward and backward in the data.

Programmers at Some Random Compa

1 2 3 4 5

First Name	Last Name	Experience
Firstname10	Lastname10	Super Hacker
Firstname11	Lastname11	Beginner
Firstname12	Lastname12	Beginner
Firstname13	Lastname13	Super Hacker
Firstname14	Lastname14	Senior
Firstname15	Lastname15	Super Hacker
Firstname16	Lastname16	Beginner
Firstname17	Lastname17	Super Hacker
Firstname18	Lastname18	Beginner
Firstname19	Lastname19	Beginner

1 2 3 4 5

localhost/primefaces-data-tables/pagination.jsf

p:dataTable: Pagination

The total length of the programmer list is 50, but by using paginator="true" rows="10", you can limit it to 10 rows at a time and get controls to move forward and backward in the data.

Programmers at Some Random Company

1 2 3 4 5

First Name	Last Name	Experience Level	Languages
Firstname40	Lastname40	Senior	LanguageA40, LanguageB40, and LanguageC40
Firstname41	Lastname41	Beginner	LanguageA41, LanguageB41, and LanguageC41
Firstname42	Lastname42	Beginner	LanguageA42, LanguageB42, and LanguageC42
Firstname43	Lastname43	Junior	LanguageA43, LanguageB43, and LanguageC43
Firstname44	Lastname44	Beginner	LanguageA44, LanguageB44, and LanguageC44
Firstname45	Lastname45	Super Hacker	LanguageA45, LanguageB45, and LanguageC45
Firstname46	Lastname46	Senior	LanguageA46, LanguageB46, and LanguageC46
Firstname47	Lastname47	Super Hacker	LanguageA47, LanguageB47, and LanguageC47
Firstname48	Lastname48	Super Hacker	LanguageA48, LanguageB48, and LanguageC48
Firstname49	Lastname49	Junior	LanguageA49, LanguageB49, and LanguageC49

1 2 3 4 5

Wrap-Up

Summary

- **General format**

```
<p:dataTable var="someName" value="#{someBean.someList}">
  <f:facet name="header">Table Title</f:facet>
  <p:column headerText="Col1 Name">#{someName.prop1}</p:column>
  <p:column headerText="Col2 Name">#{someName.prop2}</p:column>
  ...
</p:dataTable>
```

- **Options**
  - p:dataTable
    - tableStyleClass, rowStyleClass, resizableColumns, rows, paginator
  - p:column
    - sortBy, sortFunction
      - For sortable tables, use List not array and enclose table in h:form

Questions?