What this book will do for you...

- Understand basic concepts such as AJAX processing and partial view submit so that you can work with PrimeFaces
- Go beyond the basics and discover more interesting features of PrimeFaces: PrimeFaces selectors, RequestContext, Dialog Framework, and more
- Delve deep into over 100 rich UI components with all the required details
- Get practical solutions to complex problems that arise in a JSF ecosystem
- Use best practices, avoid pitfalls, and get performance tips when working with the component suite

Inside the Cookbook...

- A straightforward and easy-to-follow format
- A selection of the most important tasks and problems
- Carefully organized instructions for solving the problem efficiently
- Clear explanations of what you did
- Apply the solution to other situations

Over 100 practical recipes to learn PrimeFaces 5.x – the most popular JSF component library on the planet

Forewords by: Çağatay Çivici - Founder and Lead Developer of PrimeFaces
Ed Burns - JavaServer Faces Specification Co-lead

In this package, you will find:

- The authors biography
- A preview chapter from the book, Chapter 8 'Drag Me, Drop Me'
- A synopsis of the book’s content
- More information on PrimeFaces Cookbook Second Edition

About the Authors

Mert Çalışkan is a software stylist living in Ankara, Turkey. He has more than 10 years of expertise in software development with the architectural design of Enterprise Java web applications. He is an advocate of open source for software projects such as PrimeFaces and has also been a committer and founder of various others.

He is the coauthor of the first edition of this book. He is also the coauthor of Beginning Spring by Wiley Publications. He is the founder of AnkaraJUG, which is the most active JUG in Turkey that has been having monthly meetups for 3 years now.

In 2014, he was entitled a Java Champion for his achievements. He is also a Sun Certified Java professional since 2007. He does part-time lecturing at Hacettepe University on enterprise web application architecture and web services. He shares his knowledge at national and international conferences, such as JDays 2015, JavaOne 2013, JDC2010, and JSFDays'08. You can reach Mert via twitter at @mertcal.
Oleg Varaksin is a senior software engineer living in the Black Forest, Germany. He is a graduate computer scientist who studied informatics at Russian and German universities. His main occupation and "daily bread" in the last 10 years has consisted of building various web applications based on JSP, JSF, CDI, Spring, web services, REST, jQuery, AngularJS, and HTML5. He has a deep understanding of web usability and accessibility.

Oleg is an experienced JSF expert and has been working with the PrimeFaces library since its beginning in 2009. He is also a well-known member of the PrimeFaces community and a cocreator of the PrimeFaces Extensions project on additional JSF components for PrimeFaces.

Besides the aforementioned technologies, he has worked as a frontend developer with many other web and JavaScript frameworks—Struts, GWT, Prototype, YUI library, and so on. He also implemented an AJAX framework before all the hype about AJAX began.

Oleg normally shares the knowledge he has acquired on his blog at http://ovaraksin.blogspot.de.
PrimeFaces Cookbook
Second Edition

PrimeFaces Cookbook, Second Edition, is the most comprehensive book about PrimeFaces—the rapidly evolving and leading JSF component suite. The book provides a head start to its readers by covering all the knowledge needed to work with the PrimeFaces framework and components in the real world. It is a quick, practical guide to learn PrimeFaces, written in a clear, comprehensible style. PrimeFaces Cookbook, Second Edition, addresses a wide audience interested in modern, trendsetting Java EE web development.

What This Book Covers

Chapter 1, Getting Started with PrimeFaces, provides details on the setup and configuration of PrimeFaces, along with the core concepts for every web application powered by PrimeFaces. The chapter gives a sneak preview of the basic features of PrimeFaces, such as AJAX processing and updating, component referencing by keywords and selectors, partial submitting, handling with Internationalization and Localization, along with the right-to-left language support and resource ordering.

Chapter 2, Theming Concepts, introduces PrimeFaces themes and the concept involved. Readers will learn about the theming of PrimeFaces components. The difference between structural and skinning CSS, installing and customizing PrimeFaces themes, along with creating new themes, will be detailed. Readers will also see how to adjust the font family and the font size throughout the PrimeFaces components to provide a consistent look and feel. Discussions of two variants of theme switchers and integrating additional icons finish this chapter.

Chapter 3, Enhanced Inputs and Selects, explains how to work with the input and select components available in PrimeFaces. Such components are the main parts of every web application. PrimeFaces provides nearly 25 components for data input that extend the standard JSF component suite with user-friendly interfaces, skinning capabilities, AJAX interactions, Client-side Validation, and many other useful features.

Chapter 4, Grouping Content with Panels, covers various container components, such as panel, accordion, scrollPanel, tabView, and dashboard, which allow grouping of JSF components. Various settings to configure panel components are detailed in this chapter. Furthermore, the chapter explains how to create complex layouts with the layout component and also responsive layouts for mobile devices and desktops with Grid CSS.
Chapter 5, *Data Iteration Components*, covers basic and advanced features to visualize data with data iteration components provided by PrimeFaces, including dataTable, dataList, pickList, orderList, tree, and treeTable. The discussed features include sorting, pagination, filtering, lazy loading, and single and multiple selections. Advanced data visualization with the schedule and dataScroller components will be demonstrated as well.

Chapter 6, *Endless Menu Variations*, explains several menu variations. PrimeFaces' menus fulfill all major requirements. They come with various facets—static, dynamic, tiered, hybrid, iPod-styled, and so on—and leave nothing to be desired. Readers will face a lot of recipes that discuss the menu's structure, configuration options, customizations, and integration with other components. At the end of this chapter, readers will know what kind of menu to choose and how to put it on a page for a particular use case.

Chapter 7, *Working with Files, Images, and Multimedia*, provides ways of managing operations on files such as uploading and downloading, image operations such as capturing, cropping, and displaying images with galleria, imageSwitch, and contentFlow. Readers will learn basic as well as advanced configuration of components and use cases.

Chapter 8, *Drag Me, Drop Me*, explains how the drag and drop utilities in PrimeFaces allow you to create draggable and droppable user interfaces efficiently. They abstract developers from dealing with implementation details on the browser level. In this chapter, readers will learn about PrimeFaces' drag and drop utilities—Draggable and Droppable. AJAX-enhanced drag and drop and a special integration with data iteration components will be explained as well.

Chapter 9, *Creating Charts and Maps*, covers the ways to create visual charts with PrimeFaces' extensive charting features and create maps based on Google Maps. PrimeFaces offers basic and advanced charting with its easy-to-use and user-friendly charting infrastructure. Throughout the chapter, mapping abilities such as drawing polylines and polygons and handling markers and events are covered as well.

Chapter 10, *Client-side Validation*, gives advice on how to implement Client-side Validation (CSV) with PrimeFaces. PrimeFaces' Client Side Validation Framework is the most complete and advanced CSV solution for JSF. Readers will learn all CSV tricks—configuration, standard validation, instant validation, and integration with Bean Validation. They will also meet custom client-side validators and find out how to extend CSV with JSF validators and Bean Validation.
Chapter 11, Miscellaneous Advanced Use Cases, introduces more interesting features of the PrimeFaces library. You will learn about RequestContext—a helpful utility that allows marking components as updatable targets at runtime, adding AJAX callback parameters, opening external pages in dynamically generated dialog (Dialog Framework), and more. In this chapter, a number of real-world samples will be also developed—blocking UI during AJAX calls, periodic polling, focus handling, controlling from submission, sticking components, content caching, and targetable messages, to name a few. Furthermore, after reading this chapter, readers will be aware of the pitfalls of menus within layout units and nested panels as well as possibilities for exception handling.
Drag Me, Drop Me

In this chapter, we will cover the following topics:

- Making a component draggable
- Restricting dragging by axis, grid, and containment
- Snapping to the edges of nearest elements
- Defining droppable targets
- Restricting dropping by tolerance and acceptance
- AJAX-enhanced drag and drop
- Integrating drag and drop with data iteration components

Introduction

Drag and drop is an action, which means *grabbing* an object and *dragging* it to a different location. The components capable of being dragged and dropped enrich the Web and make a solid base for modern UI patterns. The drag and drop utilities in PrimeFaces allow us to create draggable and droppable user interfaces efficiently. They make it abstract for the developers to deal with the implementation details at the browser level.

In this chapter, we will learn about PrimeFaces' drag and drop utilities—*draggable* and *droppable*. AJAX-enhanced drag and drop, and a special integration with data iteration components, will be explained as well.
Making a component draggable

Any component can be enhanced with the draggable behavior. To enable the draggable functionality on any PrimeFaces component, we always need a component called draggable.

In this recipe, we will see how to make a component draggable and learn some basic features of draggable. To demonstrate these features, we will make several p:panel components draggable.

How to do it...

A component can be made draggable by using p:draggable. The value of the for attribute specifies a search expression for the draggable target. In our case, it matches the ID of p:panel.

Chapter 1, Getting Started with PrimeFaces, provides more details on search expressions.

If the for attribute is omitted, the parent component will be selected as a draggable target. Let us make some panel components draggable and apply some basic features:

```html
<p:panel id="pnl" header="Draggable panel with default settings">
    Drag me around
</p:panel>
<p:draggable for="pnl"/>

<p:panel id="hpnl" header="Draggable panel by handle">
    I can be only dragged by my header
</p:panel>
<p:draggable for="hpnl" handle=".ui-panel-titlebar"/>

<p:panel id="cpnl" header="Draggable panel with clone">
    I display a clone as helper while being dragged
</p:panel>
<p:draggable for="cpnl" helper="clone"/>

<p:panel id="rpnl" header="Draggable panel with revert">
    I will be returned to my start position when dragging stops
</p:panel>
<p:draggable for="rpnl" revert="true"/>

<p:panel id="opnl" header="Draggable panel with opacity">
```
```
The following screenshot shows the five panels. The last panel is being dragged. Its opacity has been changed to 0.5 after the dragging starts.

### How it works...

By default, any point in a dragged component can be used as a handle. To restrict the drag-start click to a specified element(s), we can use the handle option, which is a jQuery selector. The second panel is dragged by using its header only.

By default, the actual component is used as a drag indicator. The helper option allows keeping the component at its original location during dragging. This can be achieved with helper set to clone for the third panel.

If the revert option is set to true, the component will return to its starting position when the dragging stops, and the draggable component is not dropped onto a matching droppable component. The fourth panel features this behavior.

Opacity for helper, while it is being dragged, is another useful option to give the user a visual feedback. The opacity of the fifth panel is reduced when dragging.
There's more...

Other basic features are related to the attributes cursor and stack. cursor is a CSS cursor that is to be displayed when dragging. It is handy to set its value to move. stack is a jQuery selector. It controls the z-index of the set of draggable elements that match the selector and always brings them to the front. That means the draggable component always overlays the other draggables.

See also

Refer to the Restricting dragging by axis, grid, and containment and Snapping to the edges of nearest elements recipes discussed later in this chapter to learn the advanced features of Draggable.

PrimeFaces Cookbook Showcase application

This recipe is available in the demo web application on GitHub (https://github.com/ova2/primefaces-cookbook/tree/second-edition). Clone the project if you have not done it yet, explore the project structure, and build and deploy the WAR file on every Servlet 3.x compatible application server, such as JBoss WildFly or Apache TomEE.

The showcase for the recipe is available at http://localhost:8080/pf-cookbook/views/chapter8/draggable.jsf.

Restricting dragging by axis, grid, and containment

The dragging behavior can be limited with some configurable constraints.

In this recipe, we will see how to drag an element, either horizontally or vertically, on a grid or inside a certain section of the page.

How to do it...

The next example demonstrates three draggable panels and one draggable image. The first panel can be dragged only horizontally, the second one only vertically, and the third panel is dragged on a grid. Dragging on a grid means the dragging helper snaps to a grid—every specific x and y pixel. The image is placed within an h:panelGroup tag, which acts as a container for dragging. The image cannot go outside this container.

```html
<p:panel id="hpnl" header="Only horizontal draggable panel">
    I can be only dragged horizontally.
</p:panel>
```
The following screenshot shows the result achieved with the preceding code snippet. Especially, we can see that the image has stayed in its boundaries although the cursor has gone outside.
**How it works...**

Horizontal or vertical dragging is possible by setting the `axis` attribute as `axis="x"` or `axis="y"`, which means that the draggable element can be dragged only horizontally or only vertically, respectively.

Dragging on a grid is defined by the `grid` attribute. The value for dragging on a grid takes comma-separated dimensions. For instance, `grid="40,50"` means that the draggable element can be dragged in only 40 pixel steps horizontally and 50 vertically.

The `containment` attribute constraints dragging within the boundaries of the containment element. Possible string values are `parent`, `document`, `window`, and `[x1, y1, x2, y2]`. The setting `containment="parent"` in the preceding example means that the draggable element cannot go outside its parent.

**See also**

Refer to the Snapping to the edges of nearest elements recipe to learn about the more advanced features of Draggable.

**PrimeFaces Cookbook Showcase application**

This recipe is available in the demo web application on GitHub (https://github.com/ova2/primefaces-cookbook/tree/second-edition). Clone the project if you have not done it yet, explore the project structure, and build and deploy the WAR file on every Servlet 3.x compatible application server, such as JBoss WildFly or Apache TomEE.

The showcase for the recipe is available at http://localhost:8080/pf-cookbook/views/chapter8/advancedDraggable.jsf.

**Snapping to the edges of nearest elements**

With PrimeFaces, we can snap the dragged component to the inner or outer boundaries of another component (a component’s DOM element).

In this recipe, we will discuss snapping and its options in detail. As an example, we will develop a big `h:panelGroup` component as a snap target and three other small `h:panelGroup` components as draggable components, with various snapping options.
Chapter 8

How to do it...

Generally, the snapping behavior is activated by setting the attribute `snap` to `true`. The snapping behavior is configurable with two options—`snapMode` and `snapTolerance`. The first option, `snapMode`, determines which edges of snap elements the draggable component will snap to. The second option, `snapTolerance`, determines a distance in pixels the draggable component must be from the element when snapping is invoked.

```html
<h:panelGroup id="snaptarget" layout="block"
    styleClass="ui-widget-content"
    style="height:150px; width:450px;">
    <p class="ui-widget-header" style="margin:0; padding:5px;">
        I'm a snap target to play with me
    </p>
    <p:draggable/>
</h:panelGroup>

<h:panelGroup id="defsnap" layout="block"
    styleClass="dragSnap ui-widget-content">
    <p>I'm with default snap and snap to all edges of other draggable elements</p>
</h:panelGroup>
<p:draggable for="defsnap" snap="true"/>

<h:panelGroup id="outersnap" layout="block"
    styleClass="dragSnap ui-widget-content">
    <p>I only snap to the outer edges - try with the big box</p>
</h:panelGroup>
<p:draggable for="outersnap" snap="true" snapMode="outer"/>

<h:panelGroup id="innersnap" layout="block"
    styleClass="dragSnap ui-widget-content">
    <p>I only snap to the inner edges - try with the big box</p>
</h:panelGroup>
<p:draggable for="innersnap" snap="true"
    snapMode="inner" snapTolerance="15"/>
```
The following screenshot shows the snapping for the last h:panelGroup tag. The component can be snapped only to the inner edges of the snap target when it is being dragged.

---

**How it works...**

The snapping is enabled by setting `snap` to `true`. If the `snap` attribute is set to `false` (default), no snapping occurs. The first small h:panelGroup has no snapping options. It snaps to the inner as well as outer boundaries of other draggable components. The second h:panelGroup sets `snapMode` and can only snap to the outer boundaries. Possible values of `snapMode` are `inner`, `outer`, and `both`. The third h:panelGroup also has a custom `snapTolerance` parameter in addition to `snapMode`, set to `inner`. This is the distance in pixels from the snap element's edges at which the snapping should occur. The default value is 20 pixels, but we have set it to 15.

In the current PrimeFaces implementation, a draggable component with `snap` set to `true` snaps to all other draggable components. This is a little bit different from jQuery's Draggable ([http://jqueryui.com/draggable](http://jqueryui.com/draggable)), where we can also specify the elements that the draggable component will snap to when it is close to the edge of such an element.

---

**PrimeFaces Cookbook Showcase application**

This recipe is available in the demo web application on GitHub ([https://github.com/ova2/primefaces-cookbook/tree/second-edition](https://github.com/ova2/primefaces-cookbook/tree/second-edition)). Clone the project if you have not done it yet, explore the project structure, and build and deploy the WAR file on every Servlet 3.x compatible application server, such as JBoss WildFly or Apache TomEE.

Defining droppable targets

Any component can be enhanced with the droppable behavior. Droppable components are targets for draggable ones. To enable the droppable functionality on any PrimeFaces component, we always need a component called droppable.

In this recipe, we will see how to define droppable targets and will learn a client-side callback onDrop.

How to do it...

A component can be made droppable by using p:droppable. The component ID must match the for attribute of p:droppable. If the for attribute is omitted, the parent component will be selected as a droppable target. We will take two h:panelGroup components and make them droppable and draggable, respectively. In addition, we will define a client-side callback that gets invoked when a draggable component is dropped. This can be accomplished by the onDrop attribute, which points to a JavaScript function.

```html
<h:panelGroup id="drop" layout="block" styleClass="ui-widget-content"
    style="height:150px; width:300px;">
    <p class="ui-widget-header" style="margin:0; padding:5px;">
        Drop here
    </p>
    <p:droppable onDrop="handleDrop"/>
</h:panelGroup>
<br/>

<h:panelGroup id="drag" layout="block"
    styleClass="dragDiv ui-widget-content">
    <p>Drag me to my target</p>
</h:panelGroup>
<p:draggable for="drag"/>
```

The client-side callback highlights the droppable h:panelGroup component and adds the text Dropped! to the paragraph tag p, when invoked.

```javascript
function handleDrop(event, ui) {
    $(event.target).addClass("ui-state-highlight").
        find("p").html("Dropped!");
}
```
The following screenshot shows the result after dropping the draggable `<h:panelGroup>` component onto the droppable one:

![Dragged!](image)

**How it works...**

The `onDrop` callback gets two parameters: `event` and `ui`, which are objects holding information about the drag and drop event. The droppable target is accessible by `event.target`. We use this fact to add the style class `ui-state-highlight` to the target. This class is defined by jQuery ThemeRoller.

The `event` parameter is the original browser event, and `ui` is a prepared object with the following properties:

- `ui.draggable`: This is the current draggable element, a jQuery object
- `ui.helper`: This is the current draggable helper, a jQuery object
- `ui.position`: This is the current position of the draggable helper as the `{top, left}` object
- `ui.offset`: This is the current absolute position of the draggable helper as the `{top, left}` object

**See also**

- The most important style classes defined by jQuery ThemeRoller are described in the *Understanding structural and skinning CSS* recipe in Chapter 2, *Theming Concepts*
- Advanced configuration and use cases of `droppable` will be discussed in the remaining three recipes of this chapter, that is, *Restricting dropping by tolerance and acceptance, AJAX-enhanced drag and drop*, and *Integrating drag and drop with data iteration components*
PrimeFaces Cookbook Showcase application

This recipe is available in the demo web application on GitHub (https://github.com/ova2/primefaces-cookbook/tree/second-edition). Clone the project if you have not done it yet, explore the project structure, and build and deploy the WAR file on every Servlet 3.x compatible application server, such as JBoss WildFly or Apache TomEE.

The showcase for the recipe is available at http://localhost:8080/pf-cookbook/views/chapter8/droppable.jsf.

Restricting dropping by tolerance and acceptance

Droppable behavior is highly configurable. There are a lot of options to restrict dropping. They are useful in matching the draggable and droppable components more precisely.

In this chapter, we will meet options for tolerance and acceptance. We will take several h:panelGroup components and make them droppable with different tolerance and acceptance values.

How to do it...

Tolerance specifies which mode to use for testing if a draggable component is over a droppable target. There are four different tolerance modes. They can be chosen by the tolerance attribute of p:droppable. The following code snippet shows four h:panelGroup components with settings for tolerance:

```html
<h:panelGrid columns="4">
  <h:panelGroup id="dropFit" layout="block"
    styleClass="dropTarget ui-widget-content">
    <p class="ui-widget-header">Drop here (tolerance = fit)</p>
    <p:droppable onDrop="handleDrop" tolerance="fit"/>
  </h:panelGroup>

  <h:panelGroup id="dropIntersect" layout="block"
    styleClass="dropTarget ui-widget-content">
    <p class="ui-widget-header">Drop here (tolerance = intersect)</p>
    <p:droppable onDrop="handleDrop" tolerance="intersect"/>
  </h:panelGroup>

  <h:panelGroup id="dropPointer" layout="block"
    styleClass="dropTarget ui-widget-content">
    <p class="ui-widget-header">Drop here (tolerance = pointer)</p>
    <p:droppable onDrop="handleDrop" tolerance="pointer"/>
  </h:panelGroup>

  <h:panelGroup id="dropBox" layout="block"
    styleClass="dropTarget ui-widget-content">
    <p class="ui-widget-header">Drop here (tolerance = box)</p>
    <p:droppable onDrop="handleDrop" tolerance="box"/>
  </h:panelGroup>
</h:panelGrid>
```
The `scope` attribute is used for acceptance. Its aim is to group sets of the draggable and droppable components. Only a draggable component with the same scope value as a droppable one will be accepted during drag and drop. The following code snippet shows two draggable `h:panelGroup` components with different scope values. Only one can be dropped onto the droppable `h:panelGroup` component with the ID `dropTarget2`.

```html
<h:panelGroup id="dropTouch" layout="block"
    styleClass="dropTarget ui-widget-content">
    <p class="ui-widget-header">Drop here (tolerance = touch)</p>
    <p:droppable onDrop="handleDrop" tolerance="touch"/>
</h:panelGroup>

<br/>

<h:panelGroup id="drag" layout="block"
    styleClass="dragDiv ui-widget-content">
    <p>Drag me to my target</p>
    <p:draggable scope="dnd"/>
</h:panelGroup>
```

```
<h:panelGroup id="dropTarget2" layout="block"
    styleClass="ui-widget-content"
    style="height:120px; width:300px;">
    <p class="ui-widget-header" style="margin:0; padding:5px;">
        Drop here
    </p>
    <p:droppable onDrop="handleDrop" scope="dnd"/>
</h:panelGroup>
```
Chapter 8

The following screenshot demonstrates that the `handleDrop` callback is not invoked when the `h:panelGroup` with `scope` set to `dummy` gets dropped onto the `h:panelGroup` with `scope` set to `dnd`:

<table>
<thead>
<tr>
<th>Acceptance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drop here</td>
</tr>
</tbody>
</table>

Drag me to my target

How it works...

The following table lists four tolerance modes that define the way to accept a draggable:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fit</td>
<td>Draggable component should overlap the droppable component entirely</td>
</tr>
<tr>
<td>intersect</td>
<td>Draggable component should overlap the droppable component by at least 50 percent</td>
</tr>
<tr>
<td>pointer</td>
<td>Mouse pointer should overlap the droppable</td>
</tr>
<tr>
<td>touch</td>
<td>Draggable should overlap the droppable by any amount</td>
</tr>
</tbody>
</table>
There's more...

In addition to `scope`, there is also the `accept` attribute. This is the jQuery selector that defines the accepted components. Only the draggable components matching the selector will be accepted by the droppable component.

**PrimeFaces Cookbook Showcase application**

This recipe is available in the demo web application on GitHub (https://github.com/ova2/primefaces-cookbook/tree/second-edition). Clone the project if you have not done it yet, explore the project structure, and build and deploy the WAR file on every Servlet 3.x compatible application server, such as JBoss WildFly or Apache TomEE.

The showcase for the recipe is available at http://localhost:8080/pf-cookbook/views/chapter8/advancedDroppable.jsf.

**AJAX-enhanced drag and drop**

The user's client-side drag and drop interactions can be posted to the server. Drag and drop has only one (default) AJAX behavior event provided by the droppable component, which is processed when a valid draggable component is dropped. That is the `drop` event. If we define a listener, it will be invoked by passing an event instance of the type `org.primefaces.event.DragDrop` as parameter. This parameter holds information about the dragged and dropped components. Through this information, the server-side state of the draggable/droppable items can be updated.

In this recipe, we will develop a workflow simulating the process of pizza ordering. The pizza ordering should occur by drag and drop. Users should be able to select any available Turkish pizza and drag and drop it onto the order list. The remove functionality, capable of drag and drop, should be included as well. For this purpose, we will implement a trash for the items removed from the pizza items in the order list.

**How to do it...**

The following screenshots demonstrate the entire workflow:
The first screenshot shows the dragging process from the list of available pizzas to the order list.
The second screenshot shows what happens when the dragged pizza image is dropped into the order list. A growl component is displayed with the currently selected pizza name.

The last screenshot demonstrates the removal process. One pizza has been dragged from the order list and dropped into the trash list.

We will make the five pizza image tags \texttt{h:graphicImage draggable}.

```xml
<p:growl id="growl" escape="false"/>

<h:panelGrid id="selectPizza" columns="1">
  <h:outputText value="Kiymali Pide" styleClass="text"/>
  <h:graphicImage id="pizza1" styleClass="pizzaimage"
                  library="images" name="dragdrop/pizza1.png"
                  title="Kiymali Pide"/>

  <h:outputText value="Kusbasi Pide" styleClass="text"/>
  <h:graphicImage id="pizza2" styleClass="pizzaimage"
                  library="images" name="dragdrop/pizza2.png"
                  title="Kusbasi Pide"/>

  <h:outputText value="Sucuklu Ve Yumurtali Pide" styleClass="text"/>
  <h:graphicImage id="pizza3" styleClass="pizzaimage"
                  library="images" name="dragdrop/pizza3.png"
                  title="Sucuklu Ve Yumurtali Pide"/>

  <h:outputText value="Peynirli Pide" styleClass="text"/>
```
Two `<h:panelGroup>` tags will be made droppable. One `<h:panelGroup>` tag is intended to be used for the order list and one is for items removed from the order list. Droppable `p:droppable` tags will get AJAX behaviors `p:ajax` attached with corresponding listeners in each case. One listener should be invoked on pizza ordering and another on pizza removal.

```xml
<h:panelGroup id="order" layout="block" styleClass="ui-widget-content"
    style="width:350px; padding:1px;">
    <p class="ui-widget-header" style="margin:0; padding:5px;">
        Order
    </p>
    <p:draggable for="pizza1" helper="clone" revert="true"
cursor="move"/>
    <p:draggable for="pizza2" helper="clone" revert="true"
cursor="move"/>
    <p:draggable for="pizza3" helper="clone" revert="true"
cursor="move"/>
    <p:draggable for="pizza4" helper="clone" revert="true"
cursor="move"/>
    <p:draggable for="pizza5" helper="clone" revert="true"
cursor="move"/>
</h:panelGroup>
<p:draggable for="pizza1" helper="clone" revert="true"
cursor="move"/>
<p:draggable for="pizza2" helper="clone" revert="true"
cursor="move"/>
<p:draggable for="pizza3" helper="clone" revert="true"
cursor="move"/>
<p:draggable for="pizza4" helper="clone" revert="true"
cursor="move"/>
<p:draggable for="pizza5" helper="clone" revert="true"
cursor="move"/>

Two `<h:panelGroup>` tags will be made droppable. One `<h:panelGroup>` tag is intended to be used for the order list and one is for items removed from the order list. Droppable `p:droppable` tags will get AJAX behaviors `p:ajax` attached with corresponding listeners in each case. One listener should be invoked on pizza ordering and another on pizza removal.

```xml
<p:draggable for="pizza1" helper="clone" revert="true"
cursor="move"/>
<p:draggable for="pizza2" helper="clone" revert="true"
cursor="move"/>
<p:draggable for="pizza3" helper="clone" revert="true"
cursor="move"/>
<p:draggable for="pizza4" helper="clone" revert="true"
cursor="move"/>
<p:draggable for="pizza5" helper="clone" revert="true"
cursor="move"/>
```
<h:outputText value="#{op}"/>
</h:panelGroup>

<p:draggable for="op" revert="true" cursor="move"
scope="trash"/>
</p:dataList>

<p:droppable id="drop1" for="order" accept=".pizzaimage"
tolerance="touch" activeStyleClass="ui-state-default"
hoverStyleClass="ui-state-hover">
<p:ajax listener="#{ajaxDragDrop.onPizzaOrder}"
update="order growl"/>
</p:droppable>

<p:commandButton value="Send order" action="#
ajaxDragDrop.sendOrder"
update="growl" style="margin:10px 0 20px 0;"/>

<h:panelGroup id="trash" layout="block" styleClass="ui-widget-content"
style="width:350px; padding:1px;">
<p class="ui-widget-header" style="margin:0;
padding:5px;">Trash</p>

<h:panelGroup layout="block" style="padding:10px;"
rendered="#{empty ajaxDragDrop.removedPizza}">
Drag and drop a pizza from the ordered list to remove it
</h:panelGroup>

<p:dataList value="#{ajaxDragDrop.removedPizza}" var="rp"
rendered="#{not empty ajaxDragDrop.removedPizza}"
<h:panelGroup styleClass="text" layout="block">
<h:outputText value="#{rp}"/>
</h:panelGroup>
</p:dataList>

<p:droppable id="drop2" for="trash" scope="trash"
tolerance="touch"
activeStyleClass="ui-state-default"
hoverStyleClass="ui-state-hover">
<p:ajax listener="#{ajaxDragDrop.onPizzaRemove}"
update="order trash growl"/>
</p:droppable>
</h:panelGroup>
The corresponding CDI bean `AjaxDragDrop` adds an ordered pizza to the `orderedPizza` list, and moves the pizza to the `removedPizza` list when it gets removed. This happens in the listeners `onPizzaOrder` and `onPizzaRemove`, respectively.

```java
@Named
@ViewScoped
public class AjaxDragDrop implements Serializable {

    private List<String> orderedPizza = new ArrayList<String>();
    private List<String> removedPizza = new ArrayList<String>();

    public List<String> getOrderedPizza() {
        return orderedPizza;
    }

    public List<String> getRemovedPizza() {
        return removedPizza;
    }

    public void onPizzaOrder(DragDropEvent event) {
        HtmlGraphicImage image = (HtmlGraphicImage) event.getComponent().findComponent(event.getDragId());
        String pizza = image != null ? image.getTitle() : "";
        orderedPizza.add(pizza);
        FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_INFO, "Selected pizza: " + pizza, null);
        FacesContext.getCurrentInstance().addMessage(null, msg);
    }

    public void onPizzaRemove(DragDropEvent event) {
        DataList dataList = (DataList) event.getComponent().findComponent("orderedPizza");
        FacesContext fc = FacesContext.getCurrentInstance();
        dataList.invokeOnComponent(fc, event.getDragId(), new ContextCallback() {
            public void invokeContextCallback(FacesContext fc, UIComponent comp) {
                HtmlPanelGroup pGroup = (HtmlPanelGroup)comp;
                
```
Drag Me, Drop Me

String pizza = pGroup != null ?
    (String) pGroup.getAttributes().get("pizza") : "

    orderedPizza.remove(pizza);
    removedPizza.add(pizza);

    FacesMessage msg = new FacesMessage(
        FacesMessage.SEVERITY_INFO,
        "Removed pizza: " + pizza, null);
    fc.addMessage(null, msg);
})}
}

How it works...

To make h:graphicImage draggable, we use p:draggable with proper options:
    helper="clone", revert="true", and cursor="move". The draggable images have
the title attributes set to the pizza names. This is important for getting the dropped
pizza's name in the onPizzaOrder listener by means of the findComponent() call.
The draggable h:panelGroup tag in the order list has, in contrast to h:graphicImage,
f:attribute with the pizza name as the value. This allows us to get the dropped pizza's
name from the component's attribute map in the onPizzaRemove listener by means of
the invokeOnComponent() call. Client IDs of draggable/droppable components can be
accessed by getDragId() or getDropId() on a DragDropEvent instance.
Refer to the JSF 2 API documentation (http://javaserverfaces.java.net/nonav/docs/2.2/javadocs/javax/faces/component/UIComponent.html) to read more about findComponent() and invokeOnComponent().

Last but not least, we use different ways to accept draggable. In the case of images, we set accept to .pizzaimage. The accept attribute defines a jQuery selector for the accepted draggable components. In the case of items in the order list, we set scope to trash. The scope attribute is an alternative way to match the droppable and accepted draggable components. What is easier to use in each particular case depends on the code.

There's more...

We used two style classes with p:droppable:

- activeStyleClass set to ui-state-default
- hoverStyleClass set to ui-state-hover

They are used for better visual effects when dragging/dropping. If activeStyleClass is specified, the class will be added to the droppable component while an acceptable draggable component is being dragged. If hoverStyleClass is specified, the class will be added to the droppable component while an acceptable draggable component is being dragged over it.

PrimeFaces Cookbook Showcase application

This recipe is available in the demo web application on GitHub (https://github.com/ova2/primefaces-cookbook/tree/second-edition). Clone the project if you have not done it yet, explore the project structure, and build and deploy the WAR file on every Servlet 3.x compatible application server, such as JBoss WildFly or Apache TomEE.

The showcase for the recipe is available at http://localhost:8080/pf-cookbook/views/chapter8/ajaxDragDrop.jsf.

Integrating drag and drop with data iteration components

The droppable component has a special integration with the data iteration components extending javax.faces.component.UIData. Such PrimeFaces components are dataTable, dataGrid, dataList, dataScroller, carousel, and ring. The component tag p:droppable defines a data source option as an ID of the data iteration component that needs to be connected with droppable.
In this recipe, we will introduce a dataGrid component containing some imaginary documents and make these documents draggable in order to drop them onto a recycle bin. The dataGrid component will act as a data source for the droppable Recycle Bin.

How to do it...

For the purpose of better understanding the developed code, pictures come first. The first screenshot shows what happens when we start to drag a document. The Recycle Bin area gets highlighted as follows:

What it looks like after dropping three documents onto the Recycle Bin is reproduced in the following screenshot:
Available documents are represented as images within p:dataGrid. They are placed in the panel components, which are made draggable. The dragging occurs via the panel's titlebar. The titlebar contains the document's title (name). The recycle bin is represented by a p:fieldset tag with the ID deletedDocs. Fieldset is made droppable. It also contains a p:dataTable with the currently deleted document items. Whenever a document is being dragged and dropped into the Recycle Bin, an AJAX listener is invoked. In the listener, the dropped document is removed from the list of all available documents and added to the list of deleted documents. Data iteration components will be updated after that in order to display the correct data. The code snippet, in XHTML, looks as follows:

```xml
<p:fieldset legend="Available Documents">
    <p:dataGrid id="availableDocs" columns="3" var="doc" value="#{integrationDragDrop.availableDocs}">
        <p:column>
            <p:panel id="pnl" header="#{doc.title}" style="text-align:center">
                <h:graphicImage library="images"
                    name="dragdrop/#{doc.extension}.png"/>
            </p:panel>
        </p:column>
        <p:draggable for="pnl" revert="true"
            handle=".ui-panel-titlebar"
    </p:fieldset>
```

Drag Me, Drop Me

stack=".ui-panel" cursor="move"/>
</p:column>
</p:dataGrid>
</p:fieldset>

<p:fieldset id="deletedDocs" legend="Recycle Bin" style="margin-top:20px">
<p:outputPanel id="dropArea">
<h:outputText value="Drop documents into the recycle bin to delete them"
rendered="#{empty integrationDragDrop.deletedDocs}"
style="font-size:20px;"/>

<p:dataTable var="doc"
value="#{integrationDragDrop.deletedDocs}"
rendered="#{not empty integrationDragDrop.deletedDocs}">

<p:column headerText="Title">
<h:outputText value="#{doc.title}"/>
</p:column>

<p:column headerText="Size (bytes)"
value="#{doc.size}"/>
</p:column>

<p:column headerText="Creator">
<h:outputText value="#{doc.creator}"/>
</p:column>

<p:column headerText="Creation Date">
<h:outputText value="#{doc.creationDate}"
<:convertDateTime pattern="dd.MM.yyyy"/>
</h:outputText>
</p:column>
</p:dataTable>
</p:outputPanel>
</p:fieldset>

<p:droppable id="droppable" for="deletedDocs" tolerance="touch"
activeStyleClass="ui-state-highlight"
datasource="availableDocs">
<p:ajax listener="#{integrationDragDrop.onDocumentDrop}"
update="dropArea availableDocs"/>
</p:droppable>
The model class Document contains the document properties.

```java
class Document implements Serializable {
    private String title;
    private int size;
    private String creator;
    private Date creationDate;
    private String extension;

    public Document(String title, int size, String creator,
                     Date creationDate, String extension) {
        this.title = title;
        this.size = size;
        this.creator = creator;
        this.creationDate = creationDate;
        this.extension = extension;
    }

    // getters / setters
    ...
}
```

The bean IntegrationDragDrop creates available documents (they can be loaded from a document management system, database, or filesystem), holds two lists for the data iteration components, and provides the AJAX listener onDocumentDrop.

```java
@Named
@ViewScoped
public class IntegrationDragDrop implements Serializable {
    private List<Document> availableDocs =
        new ArrayList<Document>();
    private List<Document> deletedDocs =
        new ArrayList<Document>();

    @PostConstruct
    public void initialize() {
        availableDocs.add(new Document("Perl script", 120,
                                        "Sara Schmidt", getCreationDate(), "perl"));
        ...
    }

    public List<Document> getAvailableDocs() {
        return availableDocs;
    }
```
Drag Me, Drop Me

public List<Document> getDeletedDocs() {
    return deletedDocs;
}

public void onDocumentDrop(DragDropEvent ddEvent) {
    Document doc = (Document) ddEvent.getData();
    deletedDocs.add(doc);
    availableDocs.remove(doc);
}

private Date getCreationDate() {
    Random random = new Random();
    int day = random.nextInt(30);
    int month = random.nextInt(Calendar.DECEMBER + 1);
    int year = 2014;
    GregorianCalendar calendar =
        new GregorianCalendar(year, month, day);
    return calendar.getTime();
}

How it works...

We make the second p:fieldset tag droppable, and connect it to the p:dataList tag with the ID availableDocs. This is done by setting datasource to availableDocs on p:droppable. The AJAX listener onDocumentDrop, attached by the p:ajax tag, is invoked on the drop event. Thanks to datasource, we can now access the dropped document instance in the listener: Document doc = (Document) ddEvent.getData().

PrimeFaces Cookbook Showcase application

This recipe is available in the demo web application on GitHub (https://github.com/ova2/primefaces-cookbook/tree/second-edition). Clone the project if you have not done it yet, explore the project structure, and build and deploy the WAR file on every Servlet 3.x compatible application server, such as JBoss WildFly or Apache TomEE.

The showcase for the recipe is available at http://localhost:8080/pf-cookbook/views/chapter8/dragDropIntegration.jsf.
Where to buy this book

You can buy PrimeFaces Cookbook Second Edition from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.