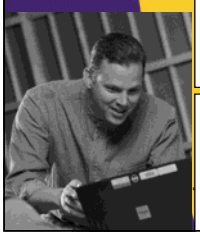
PrimeFaces: Dialogs and Other Overlays

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

For live training on JSF 2, PrimeFaces, or other Java EE topics, email hall@coreservlets.com

Marty is also available for consulting and development support



Taught by the author of *Core Servlets and JSP*, this tutorial, and JSF 2.2 version of *Core JSF*. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, Ajax, jQuery, Spring MVC, JSP, Android, general Java, Java 8 lambdas/streams, GWT, custom topic mix
  - Courses available in any location worldwide. Maryland/DC area companies can also choose afternoon/evening courses.
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Hadoop, Spring, Hibernate/JPA, RESTful Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- Theme-aware error messages
- Tooltips
- Growls: popup error messages
- Dialogs
- Dialogs with validation
- Confirmation dialogs

p:message and p:messages

p:message and p:messages: Overview

- **Appearance and behavior**
  - Theme-aware replacements for h:message and h:messages, for displaying FacesMessages
    - No styleClass attribute: picks up theme automatically
  - p:messages also has an option where it updates automatically for each Ajax request, without an explicit p:ajax “update” (or f:ajax “render”)
- **Purpose: for displaying error messages**
  - <p:messages/> displays set of error messages
    - Same as with h:messages
  - <p:message for="some-id"/> displays error message for JSF component with that ID
    - Same as with h:message

p:messages: Summary of Most Important Attributes

- **<p:messages …/>**
  - autoUpdate (true or false [default])
    - Should messages automatically be re-rendered for each Ajax request, even if “update” does not specify the messages?
  - closable (true or false [default])
    - Should an icon be added that lets user close message?
      - Note that it is “closable” (slightly more common spelling in written English), not “closeable” (the spelling used for Java interfaces).
  - escape (true [default] or false)
    - Should HTML tags in error messages be suppressed?
      - Only applies to a FacesMessage built in your own code.
  - *Same attributes as h:messages*
    - Except that style and styleClass are omitted. PrimeFaces User’s Guide specifies the name of the CSS styles that are used.

p:message: Summary of Most Important Attributes

- **<p:message for="some-id" …/>**
  - for
    - The id of JSF component whose error message should be displayed. Same usage as with h:message.
  - display (text, icon, or both [default])
    - If icon only, text is automatically available as tooltip.
  - escape (true [default] or false)
    - Should HTML tags in error messages be suppressed?
  - *Same attributes as h:message*
    - Except that style and styleClass are omitted. PrimeFaces User's Guide specifies the name of the CSS styles that are used.

Example: Entering a Message for Customer Service

- **Input page collects**
  - Message to be sent to customer service
  - Email address for replies
  - Uses p:message to show error messages
    - Alternative version uses h:message so that we can compare the look of each
- **Results page shows**
  - Confirmation of data sent
- **Note**
  - Main code is same as example in PrimeFaces String Input Elements (Part II).

Bean

```
@ManagedBean
public class MessageBean {
  private String message, emailAddress;

  public String getMessage() {
    return(message);
  }

  public void setMessage(String message) {
    this.message = message;
  }

  // Similar getter/setter for email address

  public String showMessage() {
    sendMessageToCustomerSupport();
    return("show-message");
  }
```


Input Page

```
<h:form>
<h:panelGrid columns="3" styleClass="formTable">
  Message:
  <p:editor value="#{messageBean.message}"
            required="true" id="message"
            requiredMessage="Message cannot be empty"/>
  <p:message for="message"/>
  Email address:
  <p:inputText value="#{messageBean.emailAddress}"
               required="true" id="email"
               requiredMessage="Email address required"/>
  <p:message for="email"/>
</h:panelGrid>
<p:commandButton action="#{messageBean.showMessage}"
                 value="Send to Customer Support"
                 ajax="false"/>
</h:form>
```

Alternative Input Page

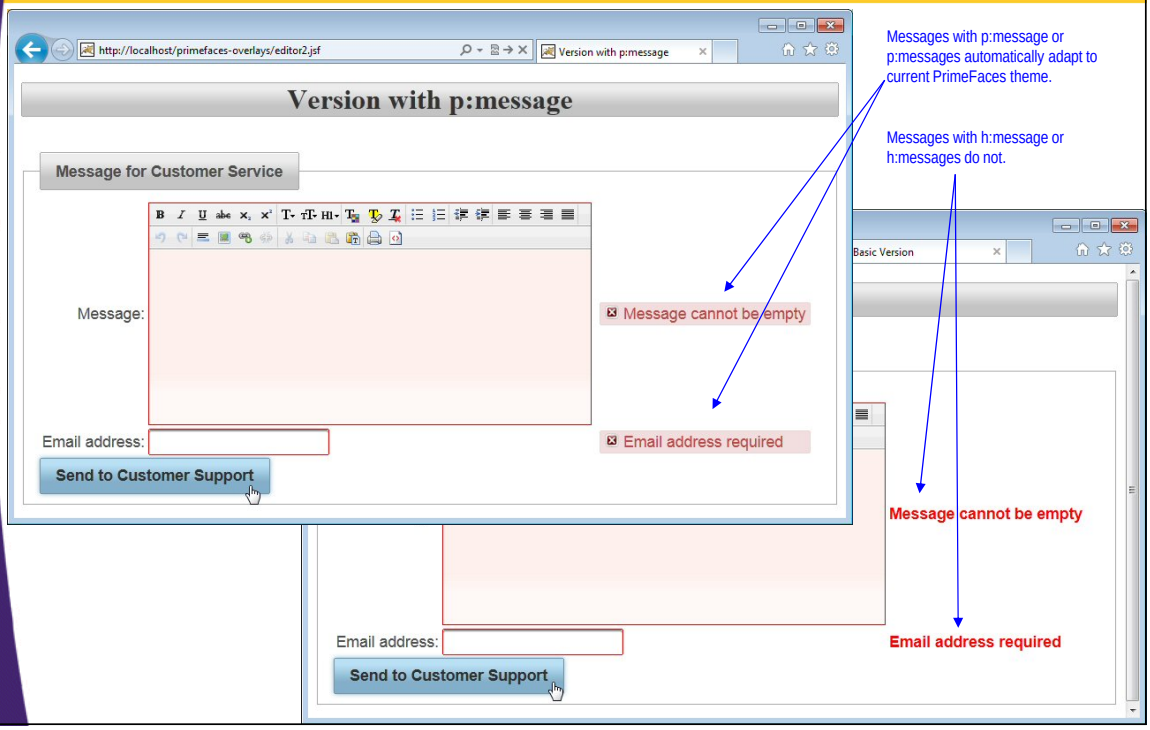
```
<h:form>
<h:panelGrid columns="3" styleClass="formTable">
  Message:
  <p:editor value="#{messageBean.message}"
            required="true" id="message"
            requiredMessage="Message cannot be empty"/>
  <h:message for="message" styleClass="error"/>
  Email address:
  <p:inputText value="#{messageBean.emailAddress}"
               required="true" id="email"
               requiredMessage="Email address required"/>
  <h:message for="email" styleClass="error"/>
</h:panelGrid>
<p:commandButton action="#{messageBean.showMessage}"
                 value="Send to Customer Support"
                 ajax="false"/>
</h:form>
```

Same as previous page except that this uses the old-style <h:message for="some-id" styleClass="some-CSS-name"/> instead of <p:message for="some-id"/>

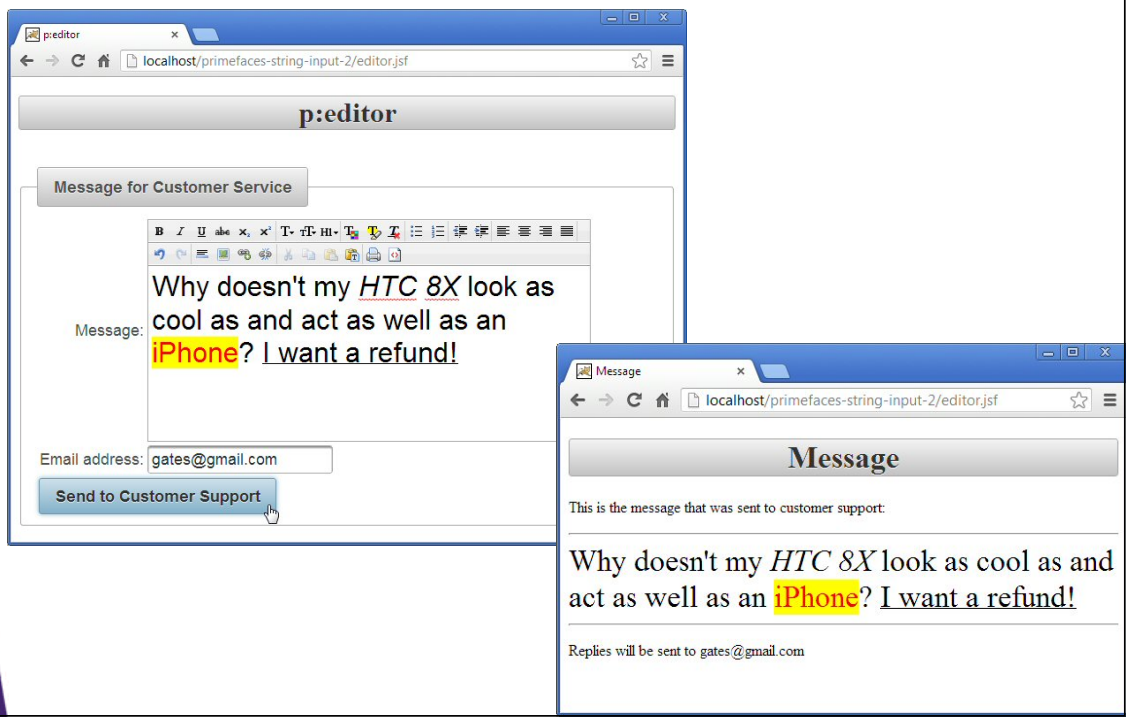
Results Page

```
...
<p>This is the message that was sent to
customer support:</p>
<hr/>
<div style="font-size: 120%">
<h:outputText value="#{messageBean.message}"
              escape="false"/>
</div>
<hr/>
<p>Replies will be sent to
#{messageBean.emailAddress}</p>
...
```

Results (Missing Data)



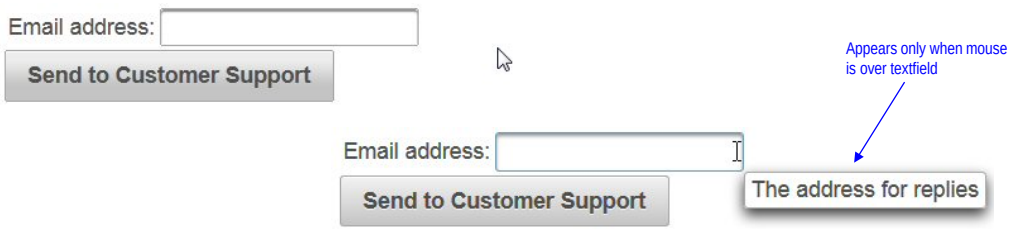
Results (Good Data)



p:tooltip

p:tooltip: Overview

- **Appearance and behavior**
  - Theme-aware tooltip that temporarily pops up when the mouse hovers over a specified element
- **Purpose: for giving extra information**
  - Information that is not important enough to put in text of page, but that is helpful to user when he or she goes to type into textfield, click on button, etc.



p:tooltip: Summary of Most Important Attributes

- **<p:tooltip for="**id**" value="**text**" …/>**
  - for
    - The id of the JSF element that triggers the tooltip. I.e., when mouse hovers over that element, the tooltip should be displayed.
  - value
    - The tooltip text. If you want images or other elements in the tooltip, use <p:tooltip…>content</p:tooltip>.
  - showEffect, hideEffect
    - jQuery UI animation effects used to show and hide the tooltip. For details of the options, see the Date Input section.
  - showEvent (default: mouseover), hideEvent (default: mouseout)
    - The events that trigger opening or closing the tooltip

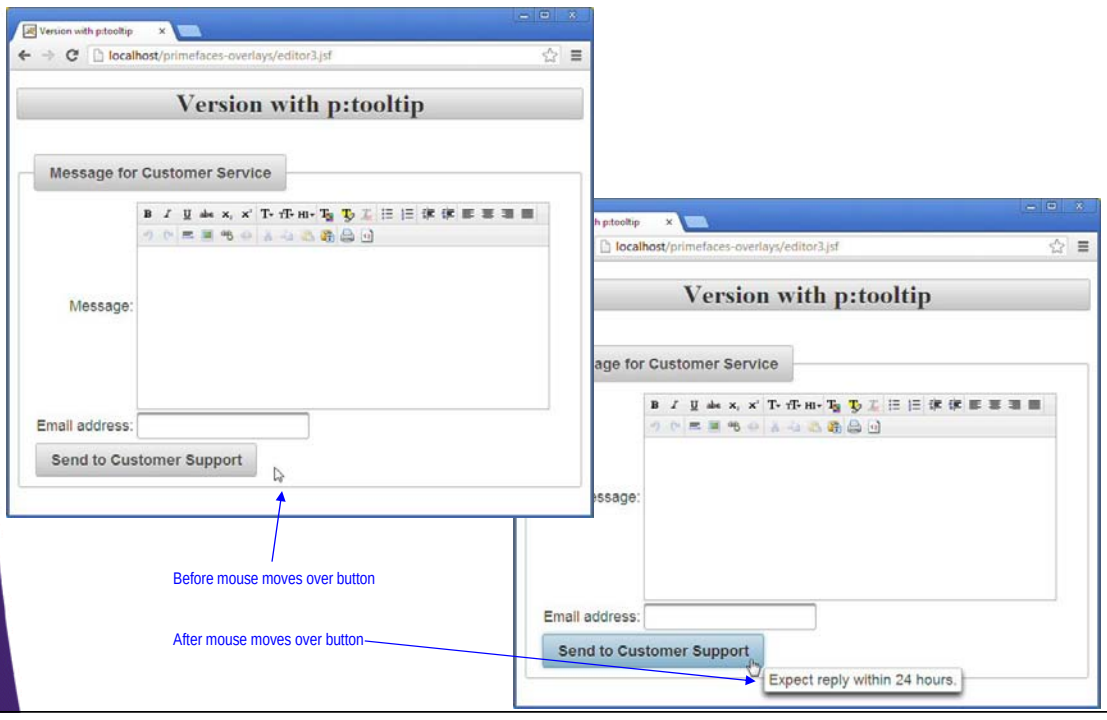
Example: Entering a Message for Customer Service

- **Input page collects**
  - Message to be sent to customer service
  - Email address for replies
  - Uses p:message to show error messages
  - Added tooltips to the textfield, editor field, and button
- **Results page shows**
  - Confirmation of data sent
- **Note**
  - Bean and results page are same as previous example, so are not repeated here

Input Page

```
<h:form>
<h:panelGrid columns="4" styleClass="formTable">
  Message:
  <p:editor value="#{messageBean.message}"
            required="true" id="message"
            requiredMessage="Message cannot be empty"/>
  <p:message for="message"/>
  <p:tooltip for="message" value="Formatted message for customer service"/>
  Email address:
  <p:inputText value="#{messageBean.emailAddress}"
               required="true" id="email"
               requiredMessage="Email address required"/>
  <p:message for="email"/>
  <p:tooltip for="email" value="The address for replies"/>
</h:panelGrid>
<p:commandButton action="#{messageBean.showMessage}" id="button"
                 value="Send to Customer Support" ajax="false"/>
<p:tooltip for="button"
           value="Expect reply within 24 hours."
           showEffect="slide" hideEffect="explode"/>
</h:form>
```


Results



p:growl

p:growl: Overview

- **Appearance and behavior**
  - Temporary translucent popup that shows error messages. Inspired by the Growl notification system for Mac OS.
- **Purpose: for displaying error messages**
  - A replacement for p:messages and h:messages for displaying FacesMessages
    - Messages are a temporary overlay rather than a permanent part of the page.

p:growl: Summary of Most Important Attributes

- **<p:growl …/>**
  - sticky (true or false [default])
    - Should messages stay up? Or fade away after *x* seconds?
  - life (*milliseconds* [default 6000])
    - How long growl stays up before it fades away. Default is 6 seconds (6000 milliseconds)
  - autoUpdate (true or false [default])
    - Should growl be automatically updated after every Ajax request? If not, use "id" and target it with the "update" attribute of p:ajax. See separate Ajax lecture.
  - showSummary, showDetail, globalOnly
    - Same as with p:messages and h:messages

Positioning the Growl Notification

- **CSS attributes**
  - By default, notification is displayed at the top right corner of the window. You can control it with the CSS selector ".ui-growl" and left or right and top or bottom values.
    - If you put values into external style sheet that is loaded with standard HTML "link" tag, you must mark them as !important, or auto-loaded PrimeFaces style sheet will override them. If you put values in <style>…</style>, or if you use h:outputStyleSheet, then !important not needed.
  - Example

```
.ui-growl {
  right: 50px !important;
  top: 100px !important;
}
```

- **Location of <p:growl/> irrelevant**
  - Since notification pops up, it does not matter where you put <p:growl/> (except of course that it must be inside the form). Conventions are the top or the bottom of the h:form.

Putting CSS Files in Resources Folder

- **Reminder from templating/components lecture**
  - If you put images, style sheets, and scripts under resources (WebContent/resources in Eclipse), then you can load them in context-independent manner
- **Style sheet**
  - WebContent/resources/css/styles.css
- **Page that uses style sheet**
  - <h:outputStylesheet name="css/styles.css"/>
    - Note no mention of “resources” in this tag. The folder under resources (css) and the file name (styles.css) are arbitrary, but the name of the main folder (resources) cannot be changed.

Example: Entering a Message for Customer Service

- **Input page collects**
  - Message to be sent to customer service
  - Email address for replies
  - Gave tooltips to the textfield, editor field, and button
  - Replaced p:messages with p:growl
- **Results page shows**
  - Confirmation of data sent
- **Note**
  - Bean and results page are same as previous example, so are not repeated here

Input Page

```
<h:form>
<h:panelGrid columns="3" styleClass="formTable">
  Message:
  <p:editor value="#{messageBean.message}"
            required="true" id="message"
            requiredMessage="Message cannot be empty"/>
  <p:tooltip for="message" value="Formatted message for customer service"/>

  Email address:
  <p:inputText value="#{messageBean.emailAddress}"
               required="true" id="email"
               requiredMessage="Email address required"/>
  <p:tooltip for="email" value="The address for replies"/>
</h:panelGrid>
<p:commandButton action="#{messageBean.showMessage}" id="button"
                 value="Send to Customer Support" ajax="false"/>
<p:tooltip for="button"
           value="Expect reply within 24 hours."/>
<p:growl/>
</h:form>
```


Style Sheet

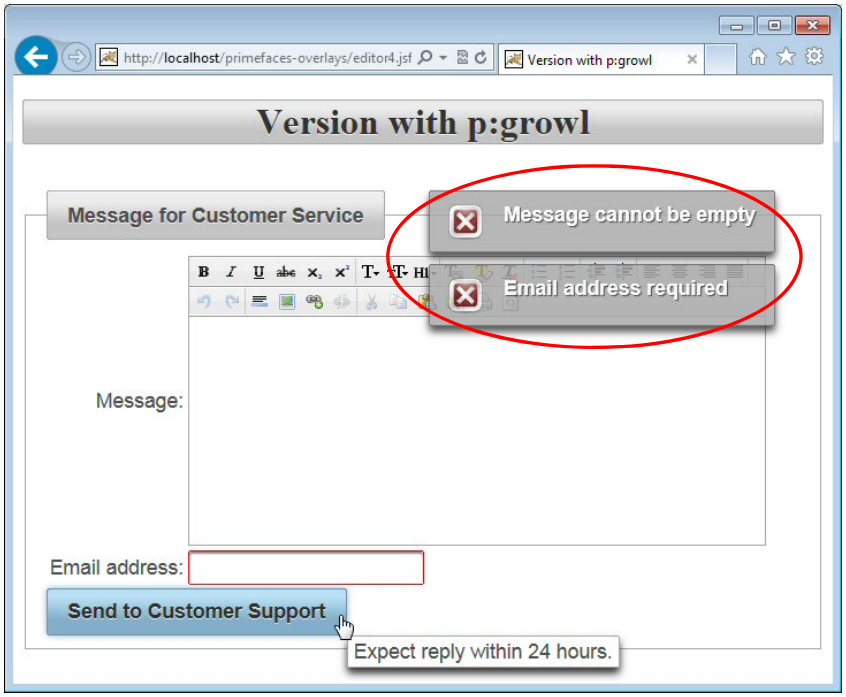
```
.formTable {
  display: table;
}
.formTable td:first-child {
  text-align: right;
}
.error {
  color: red;
  font-weight: bold;
}
.ui-growl {
  right: 50px;
  top: 100px;
}
```

Style sheet is WebContent/resources/css/styles.css.

Tag to load it is <h:outputStylesheet name="css/styles.css"/>.

Since style sheet is loaded by a JSF tag, it comes after the PrimeFaces styles, and you do not need to use !important.

Results (Missing Data)



p:dialog

p:dialog: Overview

- **Appearance and behavior**
  - Theme-aware popup window.
    - Really a styled, draggable, resizable, closeable div. So not blocked by popup blockers, and cannot be positioned outside of main browser window.
- **Purpose: for displaying data on same page**
  - Used for results, login forms, pictures, error messages, and *lots* more.
    - The availability of dialogs makes you rethink your project design
    - On large projects with separate Web page designers, it is important that designers be aware of the option of dialogs.

Dialog Basics

- **Give id to dialog with widgetVar**

```
<p:dialog header="Title" widgetVar="dlg">
   Arbitrary JSF and HTML content. Hidden at start.
</p:dialog>
```

- **Find dialog in JavaScript: PF('widgetVar')**

```
PF('dlg')
```

- **Open dialog with show()**

```
<p:commandButton … oncomplete="PF('dlg').show()"/>
```

  - Replace oncomplete with onclick if not sending Ajax data.
  - Either way, the button must be inside h:form
- **Close dialog with hide()**

```
<p:commandButton … onclick="PF('dlg').hide()"/>
```

  - This button might be inside the dialog itself
  - You can also omit any explicit button for closing, and user can just click on the “x” at top right of dialog

p:dialog: Summary of Most Important Attributes

- **<p:dialog header="*Title*" widgetVar="…">**
  Arbitrary JSF and HTML Content
  </p:dialog>
  - » Note: in general, p:dialog does not have to be inside a form. But, it often contains a bean property that is updated via Ajax, and in that situation it must be in form.

  - header
    - Title text. There is also less-used “footer” attribute.
  - widgetVar
    - ID of the element, for use by buttons or other elements that call thatId.show() and thatId.hide()
  - modal (true or false [default])
    - If modal="true", browser screen is grayed out and user cannot interact with it until dialog is closed

p:dialog: More Attributes

- showEffect, hideEffect
  - The jQuery UI animation effects used when bringing up and closing the dialog. See tutorial section on p:calendar for options and descriptions.
- position (default is centered horizontally & vertically)
  - 1 or 2 choices from center, left, right, top, and bottom
  - Or, two numbers (x, y)
    - In both cases, separated by comma
- width, height, minWidth, minHeight
  - To control size
- resizable, closable
  - Can user stretch (resizable) or close (closable) dialog?

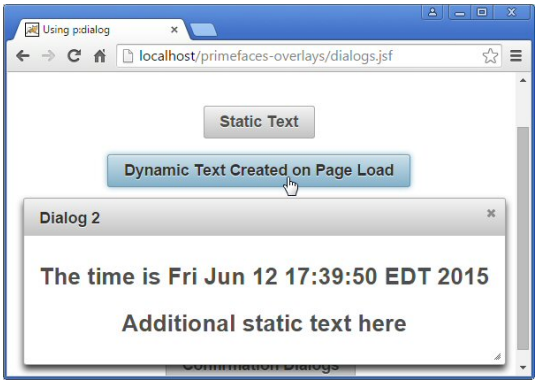
Example: Dialog with Static Text

```
<p:commandButton value="Static Text"
                 onclick="PF('dialog1').show()"/>
<p:dialog header="Dialog 1" widgetVar="dialog1">
  <h2>Some static text here</h2>
  <h2>More static text here</h2>
</p:dialog>
```

All the dialog examples are enclosed in h:form.

Example: Dialog with Dynamic Text (Created on Page Load)

```
<p:commandButton
      value="Dynamic Text Created on Page Load"
      onclick="PF('dialog2').show()"/>
<p:dialog header="Dialog 2" widgetVar="dialog2">
  <h2>The time is #{dialogBean.time}</h2>
  <h2>Additional static text here</h2>
</p:dialog>
```



If you close the dialog and then press the button to reopen it, the time does not change.

Supporting Java Code

```
@ManagedBean
public class DialogBean {
  private Date time = new Date();

  public Date getTime() {
    return(time);
  }

  public String updateTime() {
    time = new Date();
    return(null);
  }

  ...
}
```


Quick Ajax Review

- **Standard JSF: need nested f:ajax**

```
<h:commandButton action="..." value="…" ...>
    <f:ajax render="..." execute="..."/>
</h:commandButton>
```

- **PrimeFaces: attributes are in p:commandButton**

```
<p:commandButton action="..." value="…" ...
                 update="..." process="..."/>
```

- **The oncomplete attribute**
  - Triggered when the Ajax request has finished. With Ajax, pop up dialog box using oncomplete instead of onclick.
- **p:ajax**
  - If you want Ajax to be triggered by something other than a commandButton, use p:ajax in a manner similar to f:ajax, but with update/process instead of render/execute
    - See separate lecture on Ajax in PrimeFaces

Example: Dialog with Dynamic Text (Created on Button Click)

```
<p:commandButton
     value="Dynamic Text Created on Button Click"
     action="#{dialogBean.updateTime}"
     update="time-message"
     oncomplete="PF('dialog3').show()"/>
<p:dialog header="Dialog 3" widgetVar="dialog3">
  <h2>The time is
      <h:outputText value="#{dialogBean.time}"
                    id="time-message"/>
  </h2>
  <h2>Additional static text here</h2>
</p:dialog>
```

Each time you press the button to open or reopen the dialog, the current time is shown.

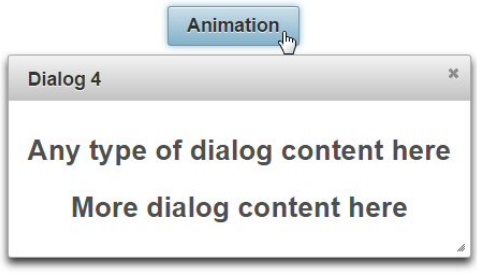
Example: Dialog with Animation Effects

```
<p:commandButton value="Animation"
                 onclick="PF('dialog4').show()"/>
<p:dialog header="Dialog 4" widgetVar="dialog4"
          showEffect="bounce" hideEffect="explode">
  <h2>Any type of dialog content here</h2>
  <h2>More dialog content here</h2>
</p:dialog>
```

When you press the command button, the dialog appears and then bounces up and down briefly.

When you click the "x" to close the dialog, it scatters into pieces as it disappears.

Use animation with great caution, and if your project is large enough to have a Web design expert, be sure to check the idea with him or her.



Example: Modal Dialogs

```
<p:commandButton value="Modal Dialogs"
                 onclick="PF('dialog5').show()"/>
<p:dialog header="Dialog 5" widgetVar="dialog5"
          modal="true">
  <h2>Any type of dialog content here</h2>
  <h2>More dialog content here</h2>
</p:dialog>
```

Ajax-Based Validation with Dialog Boxes

Dialogs and Validation

- **Problem**
  - The oncomplete event determines when Ajax has finished. No JavaScript event determines if validation passed or failed. So, dialog box pops up either way.
    - Validation fails: p:messages updated, action method not called. Validation passes: no content in p:messages, action method called. Either way, onsuccess and oncomplete triggered.
- **Solution**
  - Have content of dialog be p:messages only
  - Have "good" content be a FacesMessage, but with severity set to SEVERITY_INFO
  - Dialog pops up either way: sometimes showing validation errors, sometimes showing final results

Dialogs and Validation: Alternative Solution

- **Remove the built-in validation tags**
  - required, requiredMessage, converterMessage, validatorMessage, f:validate*Blah*, p:message, p:messages, p:growl
- **Do all validation in action controller method**
  - Validation fails: generate text (not FacesMessage) describing problem
  - Validation succeeds: generate text (not FacesMessage) showing final result
  - Use Ajax to update the h:outputText that spits out that message, then pop up the dialog
- **Major downside**
  - The built-in validation tags are *far* more convenient

Quick Preview: autoUpdate

- **Idea**
  - If you use <p:messages autoUpdate="true"/>, then the messages are automatically updated after each Ajax request
- **Convenient, but not necessary**
  - You could just give an id to the p:messages, then reference that id with the update attribute
- **Caution**
  - Only use autUpdate when there is a single p:messages in the page. Avoid if you have multiple forms in same page.
- **More details**
  - Separate lecture on Ajax with PrimeFaces

Example: Entering a Message for Customer Service

- **Input page collects**
  - Message to be sent to customer service
  - Email address for replies
  - Gave tooltips to the textfield, editor field, and button
- **Dialog instead of separate results page**
  - Behavior
    - Validation fails: dialog shows validation errors
    - Validation passes: dialog shows confirmation of data sent
  - Approach to get this behavior
    - Dialog contains only p:messages
    - Confirmation of data sent is put into a FacesMessage

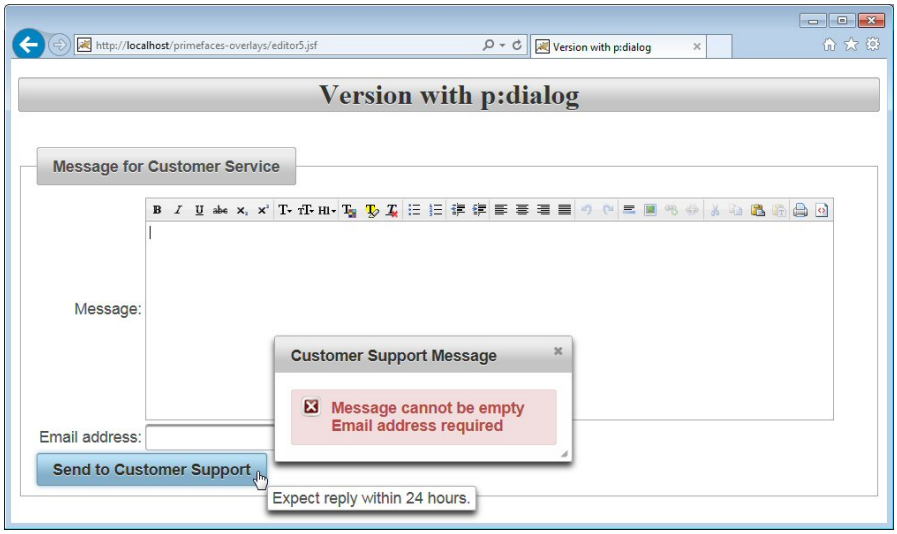
Bean (Part of MessageBean that was Shown Earlier)

```
public String showDialogMessage() {
  String messageTemplate =
    "<p>Message that was sent to customer support:</p>\n" +
    "<hr/>\n" +
    "<div style='font-size: 120%%'>\n" +
    "%s\n" +
    "</div>\n" +
    "<hr/>\n" +
    "<p>Replies will be sent to %s.</p>";
  String messageText =
    String.format(messageTemplate, message, emailAddress);
  sendMessageToCustomerSupport();
  FacesMessage message = new FacesMessage(messageText);
  message.setSeverity(FacesMessage.SEVERITY_INFO);
  FacesContext.getCurrentInstance().addMessage(null, message);
  return(null);
}
```

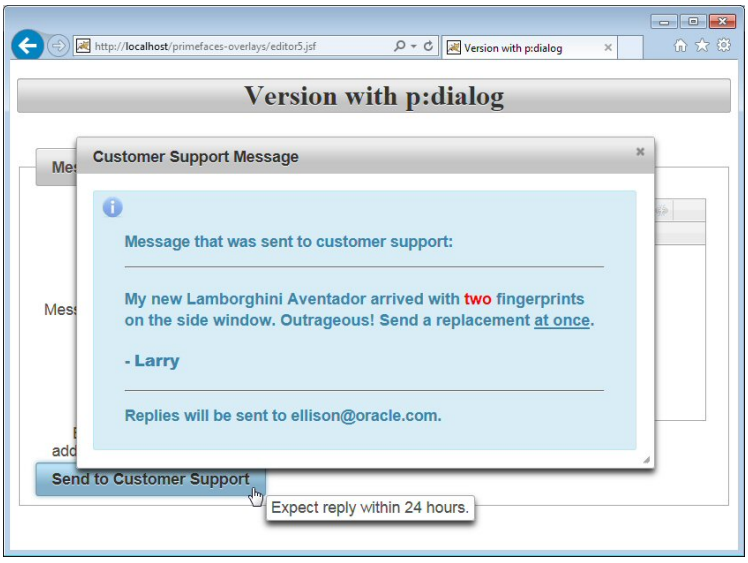

Input Page

```
<h:form>

<!-- Message and email address input as shown earlier -->

<p:commandButton action="#{messageBean.showDialogMessage}"
                 value="Send to Customer Support"
                 process="@form" id="button"
                 oncomplete="PF('dlg').show()"/>
<p:tooltip for="button"
           value="Expect reply within 24 hours."/>
<p:dialog header="Customer Support Message" widgetVar="dlg">
  <p:messages autoUpdate="true" escape="false"/>
</p:dialog>

</h:form>
```

Results (Validation Failed)



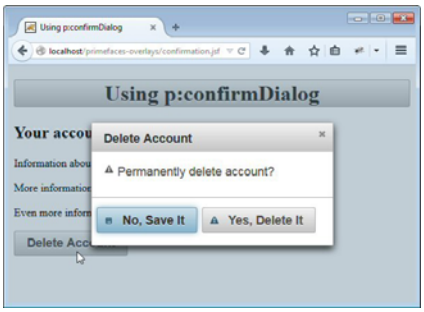
Results (Validation Passed)



p:confirmDialog

p:confirmDialog: Overview

- **Appearance and behavior**
  - Theme-aware dialog box that shows a simple message in top half and arbitrary content (usually confirm/cancel buttons) in bottom half.
    - Could be implemented with p:dialog, but more convenient when you want to ask user for confirmation.
- **Purpose: to replace JavaScript confirmation**
  - Used to ask user for agreement before showing next set of results or doing next set of actions. Modal.



p:confirmDialog: Summary of Most Important Attributes

```
<p:confirmDialog header="Title"
                 message="Text for Top Section"
                 widgetVar="...">
   Content for bottom of confirmation dialog.
   Usually buttons.
</p:confirmDialog>
```

- header, widgetVar
  - Same as with p:dialog.
- message
  - Content to go in the top of confirmation dialog (above the buttons). Alternatively, you can use f:facet inside the confirmDialog body:

    <f:facet name="message">

    Content for top half of confirmation dialog

    </f:facet>
- severity (info or alert [default])
  - Determines which icon is shown

Typical Usage

```
<p:confirmDialog ... widgetVar="confirmation">
  <p:commandButton value="Confirm" ...
                   action="#{bean.confirmAction}"
                   oncomplete="PF('confirmation').hide(); maybeMoreStuff()"/>
  <p:commandButton value="Cancel" ...
                   action="#{bean.cancelAction}"
                   oncomplete="PF('confirmation').hide(); maybeMoreStuff()"/>
</p:confirmDialog>
```

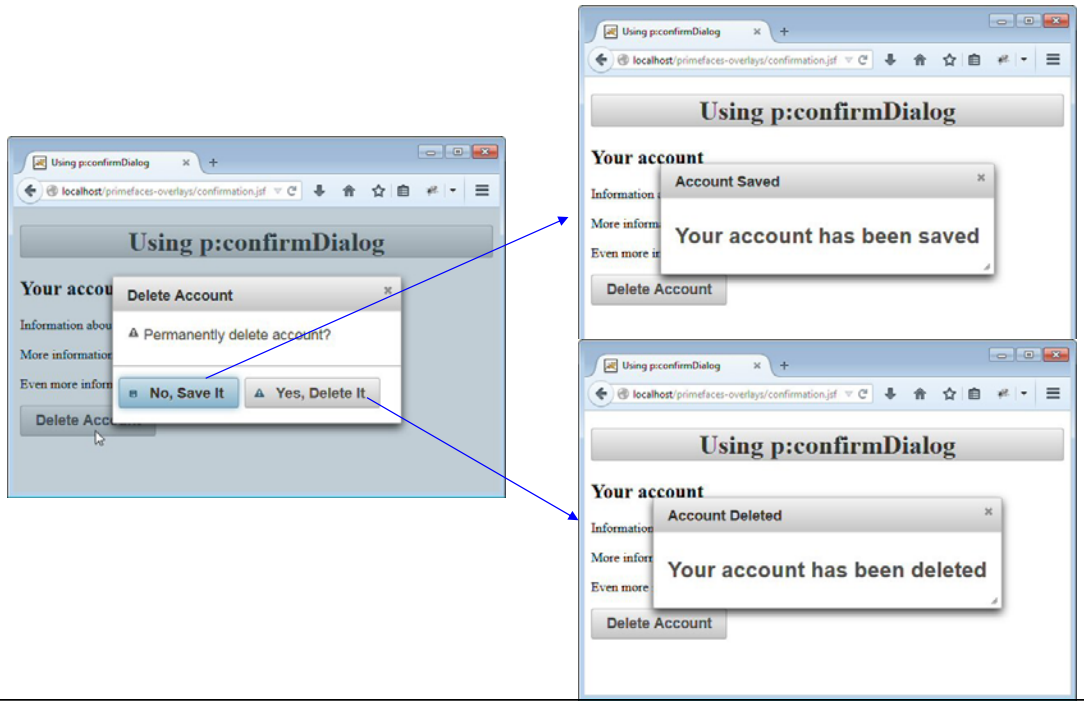
If you have no server-side code to run (common with the Cancel button), you can omit "action" and change "oncomplete" to "onclick".

The maybeMoreStuff() might involve popping up a regular p:dialog box that shows the results of confirmAction or cancelAction.

Example: Input Page

```
<p:commandButton value="Delete Account"
                 onclick="PF('confirmation').show()"/>
<p:confirmDialog header="Delete Account"
                 message="Permanently delete account?"
                 severity="alert"
                 widgetVar="confirmation">
  <p:commandButton value="No, Save It"
      icon="ui-icon-disk"
      onclick="PF('confirmation').hide(); PF('saved').show()"/>
  <p:commandButton value="Yes, Delete It"
      action="#{dialogBean.deleteAccount}"
      icon="ui-icon-alert"
      oncomplete="PF('confirmation').hide(); PF('deleted').show()"/>
</p:confirmDialog>
<p:dialog header="Account Saved" widgetVar="saved">
  <h2>Your account has been saved</h2>
</p:dialog>
<p:dialog header="Account Deleted" widgetVar="deleted">
  <h2>Your account has been deleted</h2>
</p:dialog>
```


Example: Results



Wrap-Up

Other PrimeFaces Overlays

- **p:overlayPanel**
  - Lower-level panel type used as a building block for panels that pop up.
    - http://www.primefaces.org/showcase/ui/overlay/overlayPanel.xhtml
- **p:lightBox**
  - Displays a series of content panels, one at a time, with arrows for transitions.
  - Can also run in iFrame mode, where content comes from an outside URL
    - http://www.primefaces.org/showcase/ui/overlay/lightBox.xhtml
- **p:notificationBar**
  - Fixed-position panel for notifications. Top or bottom.
    - http://www.primefaces.org/showcase/ui/panel/notificationBar.xhtml

Summary

<p:messages/> and **<p:message for="..."/>**

- Theme-aware replacements for h:messages and h:message

<p:tooltip for="id" value="text" …/>

- Attaches tooltips to other elements

<p:growl/>

- Replacement for p:messages that pops up temporarily

```
<p:dialog header="Title" widgetVar="…">
   Arbitrary JSF and HTML Content
</p:dialog>
```

- Use PF('widgetVar').show() and PF('widgetVar')hide(). If you have validation, content is often <p:messages/>.

```
<p:confirmDialog header="Title"
                 message="Stuff for Top Part"
                 widgetVar="...">
     Content for bottom. Usually buttons.
</p:confirmDialog>
```

- Outside button does show() on widgetVar. Inside buttons do hide() on widgetVar.

Questions?

More info:
http://www.coreservlets.com/JSF-Tutorial/jsf2/ – JSF 2.2 tutorial
http://www.coreservlets.com/JSF-Tutorial/primefaces/ – PrimeFaces tutorial
http://courses.coreservlets.com/jsf-training.html – Customized JSF and PrimeFaces training courses
http://coreservlets.com/ – JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training