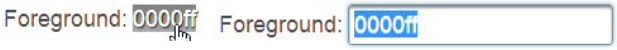
PrimeFaces: String Input Elements (Part II)

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

- **Interactive HTML color chooser**
  - p:colorPicker

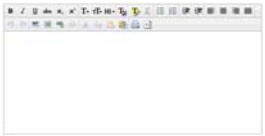
- **Text that becomes editable when clicked**
  - p:inplace



- **Element that validates text matches images**
  - p:captcha

- **Password field with feedback on strength**
  - p:password

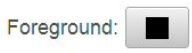
- **Editor that lets user create rich HTML text**
  - p:editor

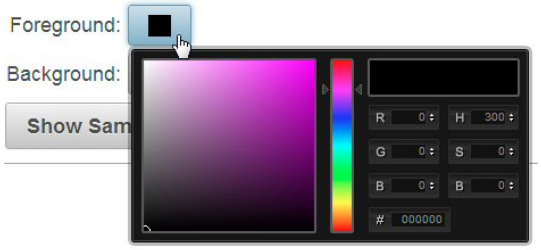


p:colorPicker

p:colorPicker: Overview

- **Appearance and behavior**
  - Small button that, when clicked, pops up interactive chooser that lets the user select an HTML color
- **Purpose: for collecting color specs**
  - Value is String of form "rrggbb" in hex.
  - Note that value does *not* start with “#”.





p:colorPicker: Problems

- **Bug in 3.5.0: pops up in wrong location**
  - Problem: with primefaces-3.5.jar (the latest version as of September 2013), the color picker pops up at bottom of the page, rather than under the button.
    - It works correctly in earlier PrimeFaces versions, and it works correctly in the PrimeFaces showcase, which uses the not-yet-public primefaces-3.5.13.jar. So, the bug is probably fixed by the time you read this, so try it the normal way before using the fix.
  - Fix: add the following between <h:head> and </h:head>

```
<h:outputScript name="jquery/jquery-plugins.js"
                library="primefaces"/>
```

- **Minor deficiency in all versions: color picker lacks a close button or “x”**
  - You must click off of the color chooser to close it

p:colorPicker: Summary of Most Important Attributes

- **<p:colorPicker …/>**
  - value
    - Should point to bean property of type String. Value sent will be of form “rrggbb” (in hex), with no leading “#”.
  - mode (popup [default] or inline)
    - Determines whether the color picker pops up when user presses the button, or if the color picker is already open and embedded in the page.
    - Works just like the mode option to p:calendar.

Example 1: Choosing Page Colors (Popup Color Picker)

- **Input page collects**
  - Foreground color
    - With popup color picker
  - Background color
    - With popup color picker
- **Results page shows**
  - Sample of text, using given colors

Bean (Main Bean Properties)

```
@ManagedBean
@SessionScoped
public class ColorPreferences implements Serializable {
  private String foreground="0000ff", background="fdf5e6";

  public String getForeground() {
    return(foreground);
  }

  public void setForeground(String foreground) {
    this.foreground = foreground;
  }

  public String getBackground() {
    return(background);
  }

  public void setBackground(String background) {
    this.background = background;
  }
```


Bean (Derived Property and Action Controller)

```
  public String getStyle() {
    String style =
      String.format("color: #%s; background-color: #%s",
                    foreground, background);
    return(style);
  }

  public String showSample() {
    return("show-colors");
  }
}
```

Since the String sent does not contain "#", you have to explicitly add it to the output.

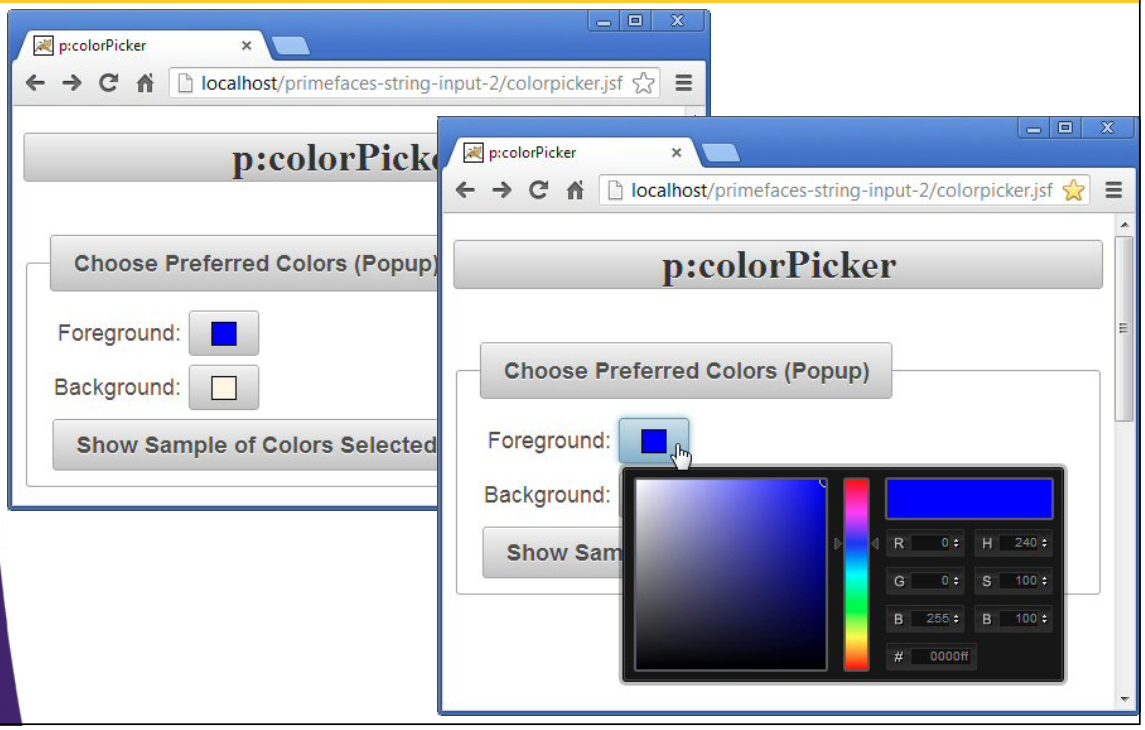
Input Page

```
<h:form>
<h:panelGrid columns="2" class="formTable">
  Foreground:
  <p:colorPicker value="#{colorPreferences.foreground}"/>

  Background:
  <p:colorPicker value="#{colorPreferences.background}"/>

</h:panelGrid>
  <p:commandButton action="#{colorPreferences.showSample}"
                   value="Show Sample of Colors Selected"
                   ajax="false"/>
</h:form>
```


Results Page

```
...
<h:body style="#{colorPreferences.style}">
...
<h2>Foreground: #{colorPreferences.foreground}</h2>
<h2>Background: #{colorPreferences.background}</h2>

<p>
Blah, blah, blah. Foo, bar, baz. Yadda, yadda, yadda.
Blah, blah, blah. Foo, bar, baz. Yadda, yadda, yadda.
...
```

Results (Input Page)



Results (Results Page)

Example 2: Choosing Page Colors (Inline Color Picker)

- **Input page collects**
  - Foreground color
    - With inline color picker
  - Background color
    - With inline color picker
- **Results page shows**
  - Sample of text, using given colors
- **Code for bean and results page**
  - Unchanged from previous example, so not repeated here

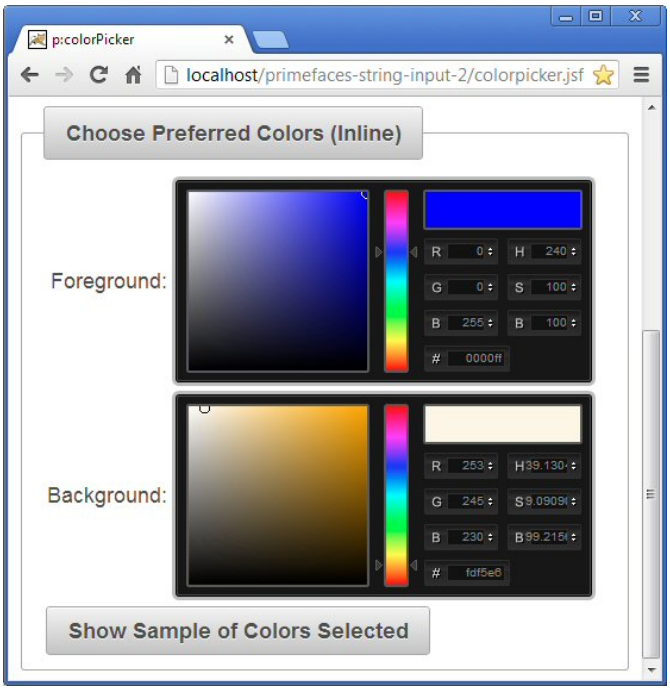
Input Page

```
<h:form>
<h:panelGrid columns="2" class="formTable">
  Foreground:
  <p:colorPicker value="#{colorPreferences.foreground}"
                 mode="inline"/>

  Background:
  <p:colorPicker value="#{colorPreferences.background}"
                 mode="inline"/>
</h:panelGrid>
  <p:commandButton action="#{colorPreferences.showSample}"
                   value="Show Sample of Colors Selected"
                   ajax="false"/>
</h:form>
```

Results Page

```
...
<h:body style="#{colorPreferences.style}">
...
<h2>Foreground: #{colorPreferences.foreground}</h2>
<h2>Background: #{colorPreferences.background}</h2>

<p>
Blah, blah, blah. Foo, bar, baz. Yadda, yadda, yadda.
Blah, blah, blah. Foo, bar, baz. Yadda, yadda, yadda.
...
```


Results (Input Page)

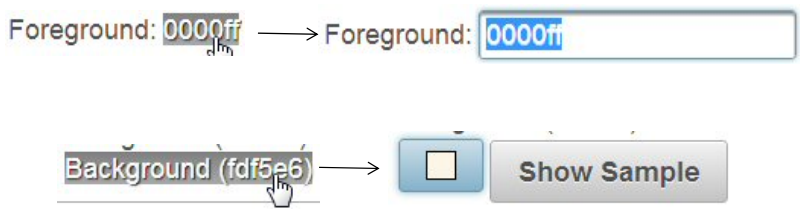


Results (Results Page)

p:inplace

p:inplace: Overview

- **Appearance and behavior**
  - Text that, when clicked, turns into something else, usually input elements
- **Purpose: for collecting arbitrary input**
  - Any type of input elements can be used



p:colorPicker: Summary of Most Important Attributes

- **<p:inplace …>***input elements***</p:inplace>**
  - *No* value attribute
    - Text is displayed, but no value is sent as part of a form. No corresponding bean property. When you click on the text, the text disappears and the elements inside are displayed instead.
    - The text that is displayed to user is the "value" attribute of the first element inside the body, unless "label" is used.
  - label
    - If you want text to be something different than "value" of first element. The label attribute may use the JSF EL.
  - editor (true or false [default])
    - If true, shows save and cancel icons. Clicking either closes the content and reverts to the text. You can also attach Ajax listeners to respond to "save" and "cancel" events, but icons also used for toggling. Elements reevaluated when clicked.

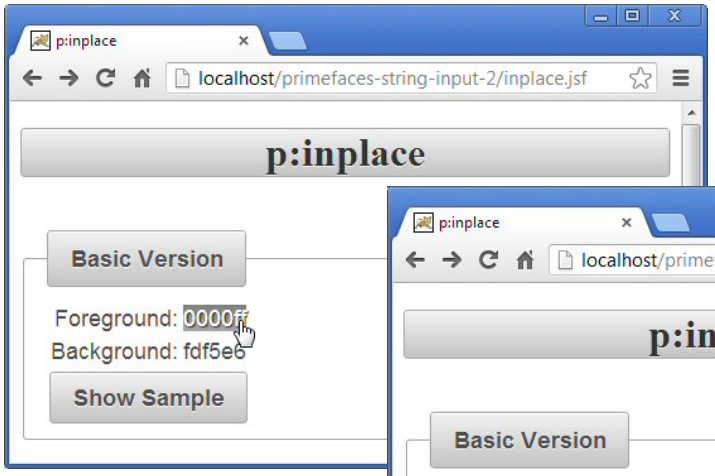
Example: Displaying Page Colors (Textfield if Clicked)

- **Input page displays**
  - Foreground color
    - Turns into textfield when clicked
  - Background color
    - Turns into textfield when clicked
- **Results page shows**
  - Sample of text, using given colors
- **Code for bean and results page**
  - Unchanged from previous example, so not repeated here

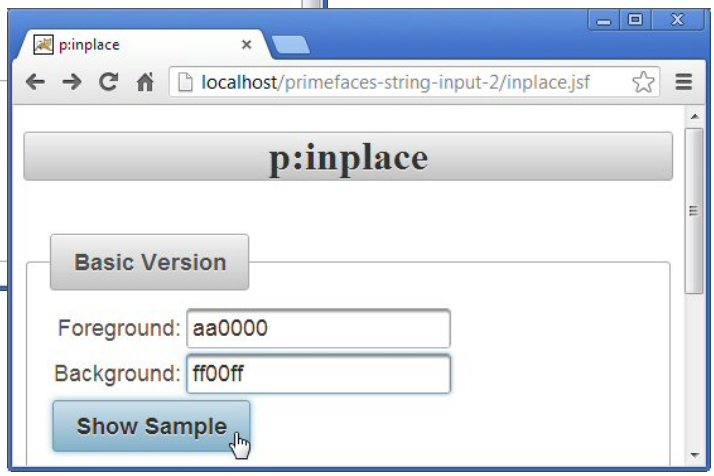
Input Page

```
<h:form>
<h:panelGrid columns="2" class="formTable">
  Foreground:
  <p:inplace>
    <p:inputText value="#{colorPreferences.foreground}"/>
  </p:inplace>
  Background:
  <p:inplace>
    <p:inputText value="#{colorPreferences.background}"/>
  </p:inplace>
</h:panelGrid>
<p:commandButton action="#{colorPreferences.showSample}"
                 value="Show Sample" ajax="false"/>
</h:form>
```

Results (Input Page)



Before clicking either of the colors



After clicking on both of the colors, then changing the values in the resultant textfields

Results (Results Page)

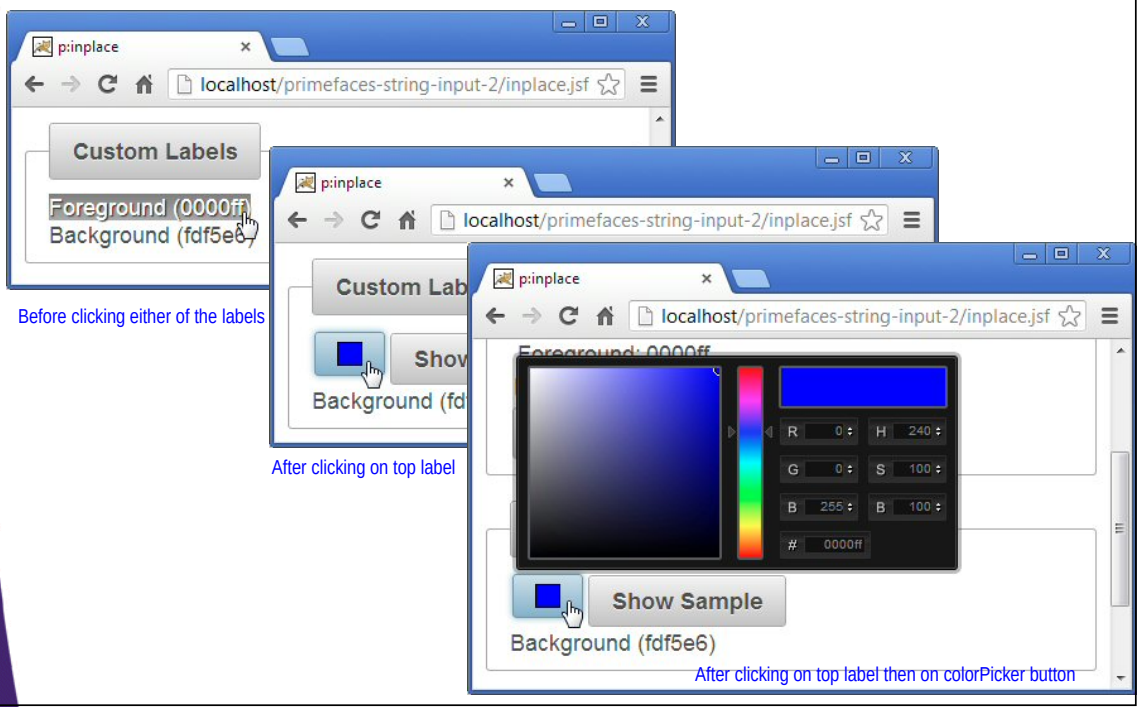
Example: Displaying Page Colors (Color Picker if Clicked)

- **Input page displays**
  - Foreground color
    - Turns into color picker when clicked. Uses custom label instead of “value” of color picker.
  - Background color
    - Turns into color picker when clicked. Uses custom label instead of “value” of color picker.
- **Results page shows**
  - Sample of text, using given colors
- **Code for bean and results page**
  - Unchanged from previous example, so not repeated here

Input Page

```
<h:form>
  <p:inplace label="Foreground (#{colorPreferences.foreground})">
    <p:colorPicker value="#{colorPreferences.foreground}"/>
    <p:commandButton action="#{colorPreferences.showSample}"
                     value="Show Sample" ajax="false"/>
  </p:inplace><br/>
  <p:inplace label="Background (#{colorPreferences.background})">
    <p:colorPicker value="#{colorPreferences.background}"/>
    <p:commandButton action="#{colorPreferences.showSample}"
                     value="Show Sample" ajax="false"/>
  </p:inplace>
</h:form>
```

Results (Input Page)



Before clicking either of the labels

After clicking on top label

After clicking on top label then on colorPicker button

Example: Displaying Random Internet Cafes (Icons to Close)

- **Input page displays**
  - Text that says "Random Internet Café"
    - Turns into image when clicked.
    - Has save and cancel icons that close the content.
    - If you click again after closing content, you get new result. Content reevaluated each time user clicks.
- **No results page**

Bean

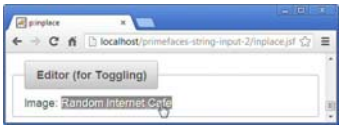
```
@ApplicationScoped
@ManagedBean
public class ImageBean {
  private int numImages = 22;
  private Random r = new Random();
  private String template =
    "/images/internet-cafes/cafe-%s.jpg";

  public String getRandomImage() {
    int num = r.nextInt(numImages) + 1;
    String path = String.format(template, num);
    return(path);
  }
}
```

The path will be used with "url" attribute of h:graphicImage. So, path is relative to WebContent in Eclipse project. If you want it relative to WebContent/resources, move "images" to resources and use <h:graphicImage name="cafe-*x*.jpg" library="images/internet-cafes"/>. h:graphicImage is discussed in the page templating section of the general JSF2 tutorial (http://www.coreservlets.com/JSF-Tutorial/jsf2/).

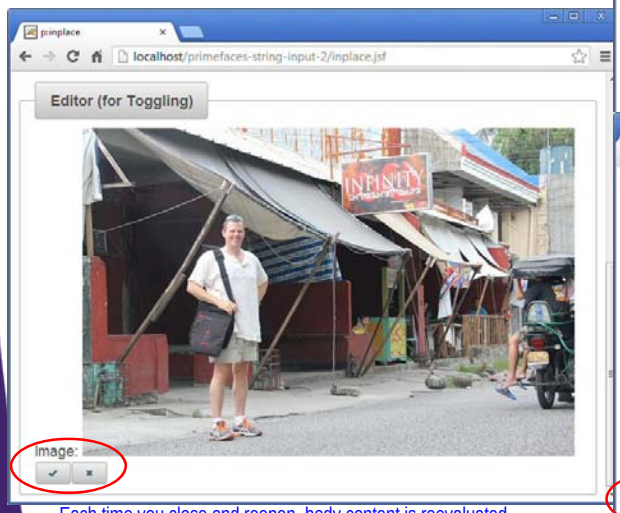
Input Page

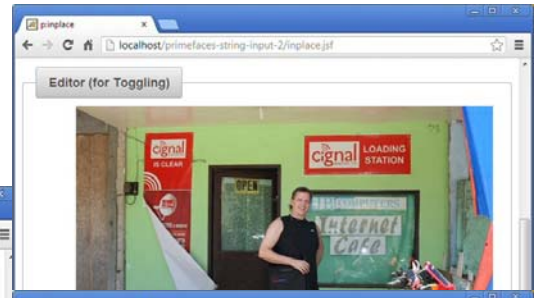
```
<h:form>
Image:
<p:inplace label="Random Internet Cafe" editor="true">
  <h:graphicImage url="#{imageBean.randomImage}"/>
</p:inplace>
</h:form>
```

Results



Before clicking the text





Each time you close and reopen, body content is reevaluated

p:captcha

p:captcha: Overview

- **Appearance and behavior**
  - Box into which user types two words to match images
  - No output or value
    - Either prevents form submission (if words do not match), or is transparent to the rest of the form (if words match)
- **Purpose: to prevent automated submissions (especially for forms that send email)**
  - Uses the Google Recaptcha API

Captcha:

p:captcha: Summary of Most Important Attributes

- **<p:captcha …/>**
  - theme (red [default], white, blackglass, clean)
    - Component is served up by Google, so does not follow the PrimeFaces themes. There are four builtin theme options.
  - language (default is en)
    - The ISO 639-1 language code to use for the prompts. Google supports a number of languages, but their documentation (http://www.google.com/recaptcha/) is unclear about which. Best option is to simply experiment.
  - validatorMessage
    - Overrides the builtin messages when user text does not match. You can also set this globally in your app via the JSF message bundle mechanism with keys primefaces.captcha.INVALID and primefaces.captcha.INVALID_detail.
      - p:captcha also supports required and requiredMessage

Captcha Installation and Setup

- **Sign up for captcha keys from Google**
  - The service is free, but it requires you to sign up for public and private keys so that the captcha process can be encrypted.
    - http://www.google.com/recaptcha/whyrecaptcha
- **Store keys as context params in web.xml**
  - Use the context-param names primefaces.PRIVATE_CAPTCHA_KEY and primefaces.PUBLIC_CAPTCHA_KEY
    - See next slide
- **Put <p:captcha …/> in your form**
  - You normally set required, requiredMessage, and validatorMessage. Theme and language are also possible.

Example Deployment Descriptor (WEB-INF/web.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
  <context-param>
    <param-name>primefaces.PRIVATE_CAPTCHA_KEY</param-name>
    <param-value>Private key from Google</param-value>
  </context-param>
  <context-param>
    <param-name>primefaces.PUBLIC_CAPTCHA_KEY</param-name>
    <param-value>Public key from Google</param-value>
  </context-param>
  ...
</web-app>
```

If you download the Eclipse project that corresponds to this lecture (see http://www.coreservlets.com/JSF-Tutorial/primefaces/), you can see a full example. My keys are stored there, but cannot be used long-term by others because I periodically cancel the old keys and request new ones.

Example: Submission with Captcha

- **Input page collects**
  - First name
    - Value corresponds to bean property
  - Last name
    - Value corresponds to bean property
  - Email address
    - Value corresponds to bean property
  - Captcha strings
    - No value; just pass or fail
- **Results page shows**
  - Confirmation of first name, last name, and email address.

Bean

```
@ManagedBean
public class RegistrationBean {
  private String firstName, lastName, emailAddress;

  public String getFirstName() {
    return(firstName);
  }
  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  // Similar getters/setters for last name & email

  public String register1() {
    return("show-registration-1");
  }
```

Bean (Derived Property, Action Controller, Helper Methods)

```
public String getObscuredPassword() {
  return(firstLetter(password) + "..."
         + lastLetter(password));
}

public String register1() {
  return("show-registration-1");
}

private String firstLetter(String s) {
  return(s.substring(0, 1));
}

private String lastLetter(String s) {
  int length = s.length();
  return(s.substring(length-1, length));
}
```


web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
  <context-param>
    <param-name>primefaces.PRIVATE_CAPTCHA_KEY</param-name>
    <param-value>6LeE...</param-value>
  </context-param>
  <context-param>
    <param-name>primefaces.PUBLIC_CAPTCHA_KEY</param-name>
    <param-value>6LeE...</param-value>
  </context-param>
  ...
</web-app>
```

Input Page (Top Half)

```
<h:form>
<h:panelGrid columns="3" class="formTable">
  First name:
  <p:inputText value="#{registrationBean.firstName}"
               required="true" id="firstName"
               requiredMessage="First name required"/>
  <p:message for="firstName"/>

  Last name:
  <p:inputText value="#{registrationBean.lastName}"
               required="true" id="lastName"
               requiredMessage="Last name required"/>
  <p:message for="lastName"/>
```


Input Page (Bottom Half)

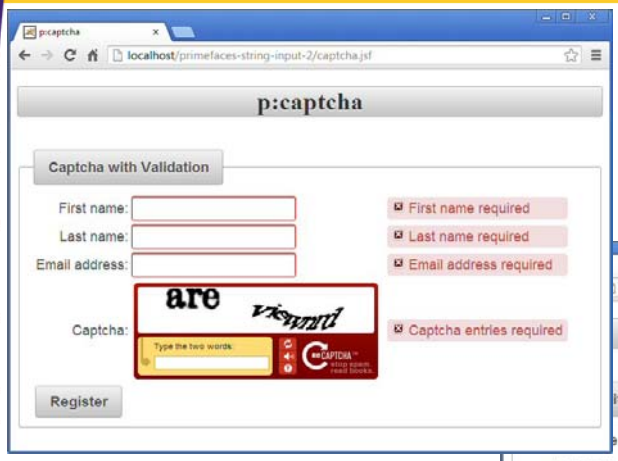
```
  Email address:
  <p:inputText value="#{registrationBean.emailAddress}"
               required="true" id="emailAddress"
               requiredMessage="Email address required"/>
  <p:message for="emailAddress"/>
  Captcha:
  <p:captcha id="captcha"
             required="true"
             requiredMessage="Captcha entries required"
             validatorMessage=
                 "Text does not match. Try again."/>
  <p:message for="captcha"/>
</h:panelGrid>
<p:commandButton action="#{registrationBean.register1}"
                 value="Register" ajax="false"/>
</h:form>
```

Results Page

```
...
Registration Confirmed</h1>
<ul>
  <li>First name: #{registrationBean.firstName}</li>
  <li>Last name: #{registrationBean.lastName}</li>
  <li>Email: #{registrationBean.emailAddress}</li>
</ul>
...
```


Results (Input Page – Blank)

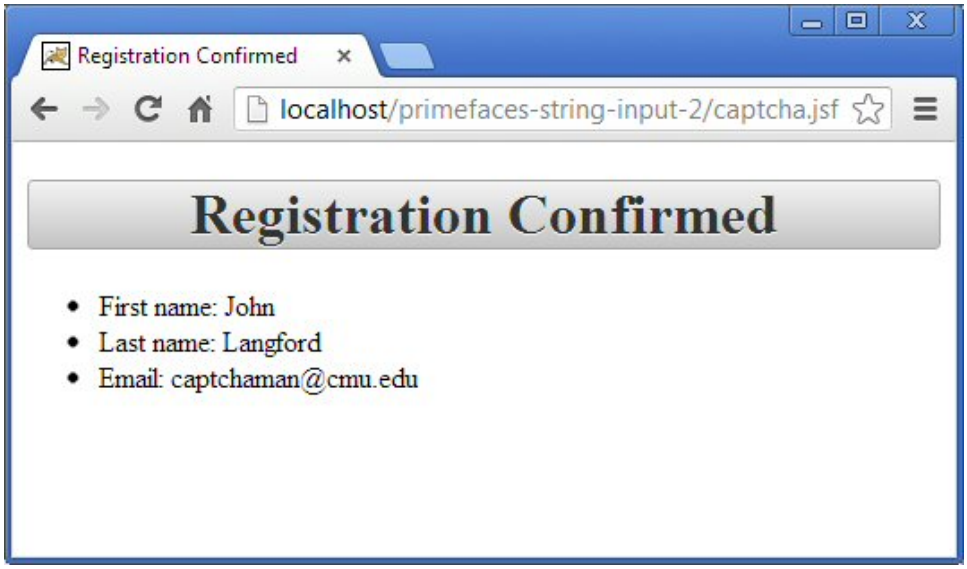
Results (Input Page – Bad Data)



Captcha entries omitted

Captcha entries did not match text

Results (Results Page)



p:password

p:password: Overview

- **Appearance and behavior**
  - Password field
  - Optionally, popup feedback on password strength
- **Purposes**
  - Basic: collect password to match against stored value
    - In this usage, p:password is just a theme-aware replacement for h:inputSecret. Note the inconsistency in the name (p:password, not p:inputSecret).
  - Alternative: collect *new* password
    - In this usage, you set feedback="true", and as user enters text, a small popup gives feedback on how strong the password is

p:password: Summary of Most Important Attributes

- **<p:password …/>**
  - value
    - Should point to bean property of type String.
  - feedback (true or false [default])
    - Determines whether there should be a popup showing feedback on password strength
  - promptLabel, weakLabel, goodLabel, strongLabel
    - Replaces the builtin prompts

Example 1: Choosing Password (Default Prompts)

- **Input page collects**
  - First name
  - Last name
  - Email address
  - Password
    - This is new password, not password to match against a stored password, so we give feedback on strength
- **Results page shows**
  - Confirmation of all values.
  - For security reasons, only part of the password is shown

Bean (Main Bean Properties)

```
@ManagedBean
public class RegistrationBean {
  private String firstName, lastName,
                 emailAddress, password;

  public String getFirstName() {
    return(firstName);
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  // Similar getters and setters for last name,
  // email address, and password
```


Bean (Derived Property, Action Controller, Helper Methods)

```
  public String getObscuredPassword() {
    return(firstLetter(password) + "..."
           + lastLetter(password));
  }

  public String register2() {
    return("show-registration-2");
  }

  private String firstLetter(String s) {
    return(s.substring(0, 1));
  }

  private String lastLetter(String s) {
    int length = s.length();
    return(s.substring(length-1, length));
  }
```

Input Page (Top)

```
<h:form>
<h:panelGrid columns="3" class="formTable">
  First name:
  <p:inputText value="#{registrationBean.firstName}"
               required="true" id="firstName"
               requiredMessage="First name required"/>
  <p:message for="firstName"/>
  Last name:
  <p:inputText value="#{registrationBean.lastName}"
               required="true" id="lastName"
               requiredMessage="Last name required"/>
  <p:message for="lastName"/>
  Email address:
  <p:inputText value="#{registrationBean.emailAddress}"
               required="true" id="emailAddress"
               requiredMessage="Email address required"/>
  <p:message for="emailAddress"/>
```

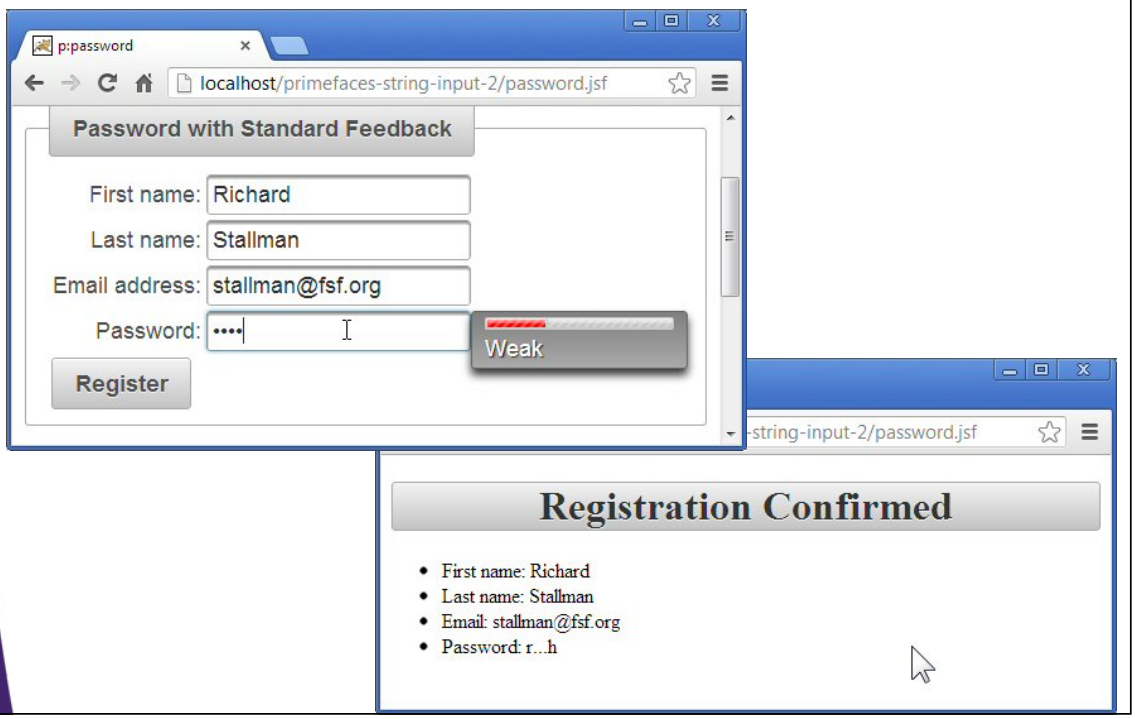

Input Page (Bottom)

```
 Password:
  <p:password value="#{registrationBean.password}"
              feedback="true"
              id="password1"
              required="true"
              requiredMessage="Password required"
              validatorMessage=
                  "Password must be 2 or more chars.">
    <f:validateLength minimum="2"/>
  </p:password>
  <p:message for="password1"/>
</h:panelGrid>
<p:commandButton action="#{registrationBean.register2}"
                 value="Register" ajax="false"/>
</h:form>
```

Results Page

```
...
Registration Confirmed</h1>
<ul>
  <li>First name: #{registrationBean.firstName}</li>
  <li>Last name: #{registrationBean.lastName}</li>
  <li>Email: #{registrationBean.emailAddress}</li>
  <li>Password: #{registrationBean.obscuredPassword}</li>
</ul>
...
```


Results



Example 2: Choosing Password (Custom Prompts)

- **Input page collects**
  - First name, last name, email address, and new password as before
    - But custom prompts are used for password feedback
- **Results page shows**
  - Confirmation of values as before
- **Code for bean and results page**
  - Unchanged from previous example, so not repeated here

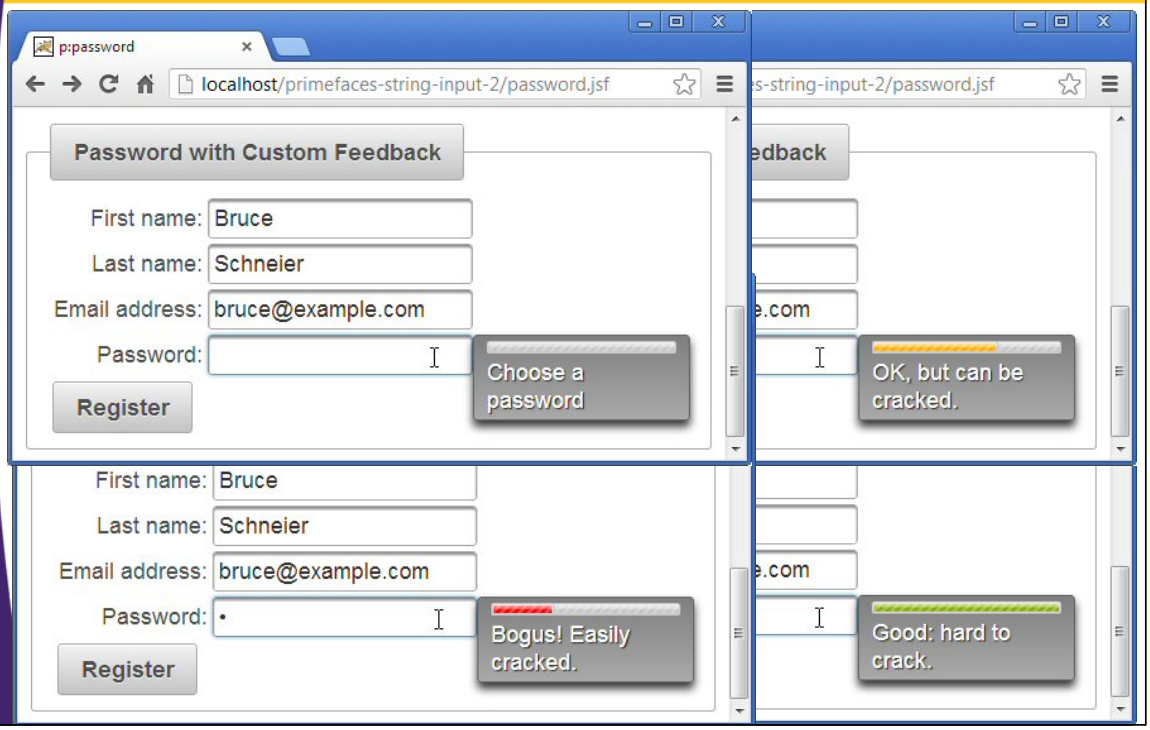
Input Page (Top – Same as Before)

```
<h:form>
<h:panelGrid columns="3" class="formTable">
  First name:
  <p:inputText value="#{registrationBean.firstName}"
               required="true" id="firstName"
               requiredMessage="First name required"/>
  <p:message for="firstName"/>
  Last name:
  <p:inputText value="#{registrationBean.lastName}"
               required="true" id="lastName"
               requiredMessage="Last name required"/>
  <p:message for="lastName"/>
  Email address:
  <p:inputText value="#{registrationBean.emailAddress}"
               required="true" id="emailAddress"
               requiredMessage="Email address required"/>
  <p:message for="emailAddress"/>
```

Input Page (Bottom)

```
  Password:
  <p:password value="#{registrationBean.password}"
              feedback="true"
              promptLabel="Choose a password"
              weakLabel="Bogus! Easily cracked."
              goodLabel="OK, but can be cracked."
              strongLabel="Good: hard to crack."
              id="password2"
              required="true"
              requiredMessage="Password required"
              validatorMessage="Password must be 2 or more chars.">
    <f:validateLength minimum="2"/>
  </p:password>
  <p:message for="password2"/>
</h:panelGrid>
<p:commandButton action="#{registrationBean.register2}"
                 value="Register" ajax="false"/>
</h:form>
```


Results



Common Questions Regarding Password Feedback

- **What algorithm is used?**
  - Not documented, but you can look in the PrimeFaces JavaScript source code for the testStrength function of the PrimeFaces.widget.Password object
- **Can I change the algorithm?**
  - Also not documented, but you can do this:

```
PrimeFaces.widget.Password.prototype.testStrength= function(password) {
    // Return number between 0 and 100
}
```

    - 0-30 means weak, 30-80: medium, 80-100: strong
- **Can I accept or reject passwords based on their strength?**
  - No, not without extensive changes to the code. End user sees the feedback, but the server does not.

p:editor

p:editor: Overview

- **Appearance and behavior**
  - Large text area where user can enter and style text
    - Uses CLEditor: http://premiumsoftware.net/cleditor/
    - For an alternative JSF plugin, see http://www.primefaces.org/showcase-ext/views/ckEditor.jsf
- **Purpose: for collecting HTML text**
  - Value is String that may contain HTML tags
  - When you output bean property, you must use escape="false" to preserve the HTML tags.

p:colorPicker: Summary of Most Important Attributes

- **<p:editor .../>**
  - value
    - Should point to bean property of type String. String may contain HTML tags.
  - controls
    - A space-separated list of the controls to show at the top of the editor. Options include bold, italic, underline, strikethrough, undo, redo, etc. See the PDF User's Guide for a complete list.
      - To prevent attacks, for pages accessible to the outside world that display one user's results to another user, you should disallow the "source" option. Failing to do so lets attackers easily enter arbitrary HTML tags, including <script>.
      - Sadly, you can only list controls you do allow, not ones you disallow. This means that disallowing "source" means you will have to list all the others one at a time.

Example: Entering a Message for Customer Service

- **Input page collects**
  - Message to be sent to customer service
  - Email address for replies
- **Results page shows**
  - Confirmation of data sent

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() {
    return("show-message");
  }
```

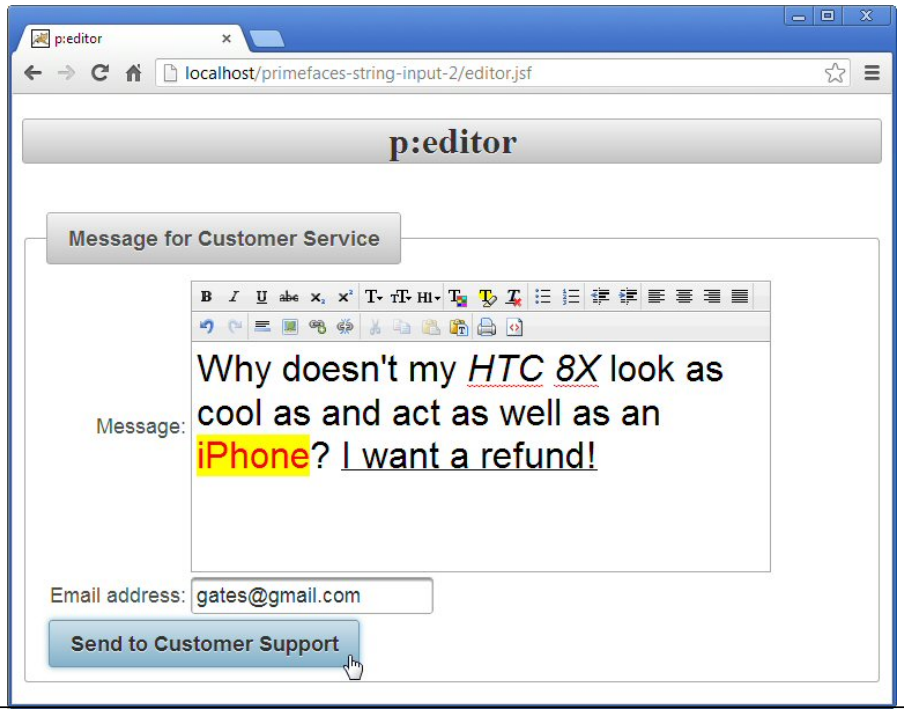
Input Page

```
<h:form>
<h:panelGrid columns="3" class="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>
```


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 (Input Page)



Results (Results Page)

p:editor Security Risks: Problem

- **Malicious end users can send code with links, images, scripts, etc., that other users see**
  - Displaying unfiltered user data containing HTML can result in badly formatted pages if user enters unbalanced tags, images, and so forth.
  - Storing the results and displaying them later to outside users lets users make malicious pages (with scripts, etc.) that are hosted on *your* site
  - The CL Editor used by PrimeFaces has a way that lets clever users directly enter arbitrary HTML tags (see my interactive example). But even if it did not, a sophisticated attacker could bypass the CL Editor and send the text directly to your server. So, if you display *user* data with escape="false", you are vulnerable.

p:editor Security Risks: Solution

- **Filter on the server**
  - Do not trust client. Filter out the bad HTML tags on server.
    - Normally all tags are automatically filtered out by JSF, but when you say escape="false" they are not, and now you have selectively filter some tags and not others. Best approach is white listing: keep only certain tags, reject all others.
    - We are ignoring these risks in this example, but you should not in real life.
- **Resources**
  - Cross site scripting references
    - https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29
  - Java tools for filtering HTML
    - http://jsoup.org/cookbook/cleaning-html/whitelist-sanitizer

Wrap-Up

Summary

- **<p:colorPicker value="…" mode="…"/>**
  - Collects color string, with *no* leading "#"
- **<p:inplace label="…" editor="…">**
  Some content, usually input elements
  </p:inplace>
- **<p:captcha/>**
  - Prevents automated submissions
- **<p:password value="…" feedback="…"/>**
  - Collects password, with optional strength feedback
- **<p:editor value="…"/>**
  - Collects formatted HTML text. Use escape="false" when outputting, and beware of displaying to other users.

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