



View Params, GET Requests, & Bookmarking

Originals of Slides and Source Code for Examples:
<http://www.coreservlets.com/JSF-Tutorial/jsf2/>

Customized Java EE Training: <http://courses.coreservlets.com/>
Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



For live training on JSF 2.x, please see courses at <http://courses.coreservlets.com/>.



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

- Courses developed and taught by Marty Hall
 - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
 - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
 - Courses developed and taught by coreservlets.com experts (edited by Marty)
 - Spring, Hibernate/JPA, EJB3, GWT, Hadoop, SOAP-based and RESTful Web Services
- Contact hall@coreservlets.com for details**

Topics in This Section

- **Motivation**
 - Why support GET?
- **Using f:viewParam**
 - To capture incoming request parameters
- **Using h:link and h:button**
 - To send outgoing request parameters
- **Using non-JSF forms**
 - To send data to JSF pages that use f:viewParam
- **Using POST-redirect-GET**
 - To make JSF results pages bookmarkable

5

© 2012 Marty Hall



Overview

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Idea

- **f:viewParam lets you associate bean properties with request parameters**
 - This introduces several new capabilities
 - New tags that navigate via GET instead of POST, and send parameters along with the address
 - Sending data from non-JSF forms to JSF pages
 - Make results pages bookmarkable
 - This is a new feature in JSF 2.0
- **Mini example**

```
<f:viewParam name="fg"
              value="#{colorPreferences.foreground}"/>
```

 - If the “fg” parameter is non-null, it is passed to setForeground before the page is rendered

7

Motivation

- **f:viewParam lets you:**
 - Make links or buttons that pass parameters to pages
 - E.g., you could make different links to the same JSF page, but specify colors, languages, or simple variables
 - Make JSF pages accessible from forms in non-JSF apps
 - But the forms would normally need to send simple values, since you lose the strong association with bean properties in the forms, and you lose JSF’s validation capabilities
 - Bookmark results pages
 - Use POST-redirect-GET to expose data needed in results pages as GET parameters. Can save page in favorites.
- **f:viewParam does *not* let you:**
 - Make h:form use GET instead of POST
 - Access any random JSF page with GET

8

Setup

- **Declare the f:namespace**

- In the <html ...> start tag, you must declare the namespace for the f: tags
 - We saw the same requirement in other tutorial sections where we used f: tags (e.g., f:selectItems, f:ajax, etc.)

- **Put f:viewParam within f:metadata**

```
<f:metadata>
  <f:viewParam name="param1" value="#{bean.prop1}"/>
  <f:viewParam name="param2" value="#{bean.prop2}"/>
</f:metadata>
```

- By convention, this is placed at the top of the page, under the <html...> start tag

- **Use the bean in the page**

- You might directly use the properties that were passed in, or you might use properties derived from them.

9

Typical Page Structure

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
  <f:metadata>
    <f:viewParam name="param1" value="#{bean.prop1}"/>
    <f:viewParam name="param2" value="#{bean.prop2}"/>
  </f:metadata>
  <h:head>...</h:head>
  <h:body>
    Blah, blah, #{bean.prop1}
    Blah, blah, #{bean.prop2}
    Blah, blah, #{bean.derivedProp}
  </h:body>
</html>
```

If the page is called with page.jsf?param1=foo¶m2=bar, then "foo" and "bar" are passed to "setProp1" and "setProp2" before the page is rendered. If any of the parameters are null (i.e., no such request parameter exists), then the associated setter is not called at all, and the bean has its normal value for that property.

10



Example Scenario

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Example Application

- **Basic banking application (shown here)**
 - Input form collects customer ID, password, foreground color, background color
 - Customer ID and password used to find Customer that has first name, last name, and bank account balance
 - Colors stored in session-scoped bean that affects all future pages
 - Normal JSF validation used
- **Want to add several features (next sections)**
 - Make links to login page with predefined colors
 - Bookmark login page with preferred colors saved
 - Bookmark the page showing the balance
 - This last feature is problematic, as we will see

Color Preferences Bean

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

    public String getForeground() {
        return(foreground);
    }
    public void setForeground(String foreground) {
        if (!isEmpty(foreground)) {
            this.foreground = foreground;
        }
    }
    // getBackground and setBackground are similar

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

Using explicit checks for empty strings because later we will add the capability for the foreground to come in via a request parameter.

13

Color Preferences Bean

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

    public String getForeground() {
        return(foreground);
    }
    public void setForeground(String foreground) {
        if (!isEmpty(foreground)) {
            this.foreground = foreground;
        }
    }
    // getBackground and setBackground are similar

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

Using explicit checks for empty strings because later we will add the capability for the foreground to come in via a request parameter.

14

Input Form: Top (lookup1.xhtml)

```
...  
<h:body style="#{colorPreferences.style}">  
...  
<h:form prependId="false">  
  <label>  
    <font color="red">*</font> Customer ID:  
    <h:inputText value="#{bankForm1.customerId}"  
      required="true"  
      requiredMessage="Missing customer ID"  
      id="customerId"/>  
    <h:message for="customerId" styleClass="error"/>  
  </label><br/>
```

Next section will add ability to pass in request params that control these colors.

Using prependId makes it easier to refer to the input element ID in the action controller method when setting a custom error message.

If no user ID is entered, the form is redisplayed and "Missing customer ID" is shown. If an unknown ID is entered, the action controller method sets a custom FacesMessage and returns null so that the form is redisplayed and "Unknown ID" is shown. If a recognized ID is entered, will navigate to page showing name and balance (using preferred colors).

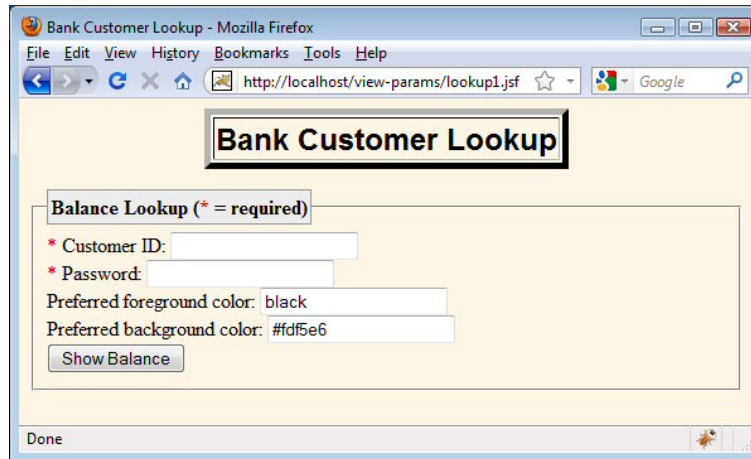
15

Input Form: Bottom (lookup1.xhtml)

```
<label>  
  <font color="red">*</font> Password:  
  <h:inputSecret value="#{bankForm1.password}"  
    required="true"  
    requiredMessage="Missing password"  
    id="password"/>  
  <h:message for="password" styleClass="error"/>  
</label><br/>  
<label>  
  Preferred foreground color:  
  <h:inputText value="#{colorPreferences.foreground}"/>  
</label><br/>  
<label>  
  Preferred background color:  
  <h:inputText value="#{colorPreferences.background}"/>  
</label><br/>  
<h:commandButton value="Show Balance"  
  action="#{bankForm1.findBalance}"/>  
</h:form> ...
```

16

Input Form (Initial Result)



17

Backing Bean: Properties

```
@ManagedBean
public class BankForm1 {
    protected String customerId, password;
    protected Customer customer = new Customer();
    protected static CustomerLookupService lookupService =
        new CustomerSimpleMap();

    public String getCustomerId() {
        return(customerId);
    }

    public void setCustomerId(String customerId) {
        this.customerId = customerId;
    }

    // getPassword and setPassword are similar
}
```

18

Backing Bean: Action Controller

```
public String findBalance() {
    customer = lookupService.findCustomer(customerId);
    FacesContext context =
        FacesContext.getCurrentInstance();
    if (customer == null) {
        String message =
            String.format("Unknown ID '%s'", customerId);
        context.addMessage("customerId", new FacesMessage(message));
    }
    if (!password.equals("secret")) {
        String message = "Incorrect password";
        context.addMessage("password", new FacesMessage(message));
    }
    if (context.getMessageList().size() > 0) {
        return(null);
    } else {
        return ("show-customer1");
    }
}
```

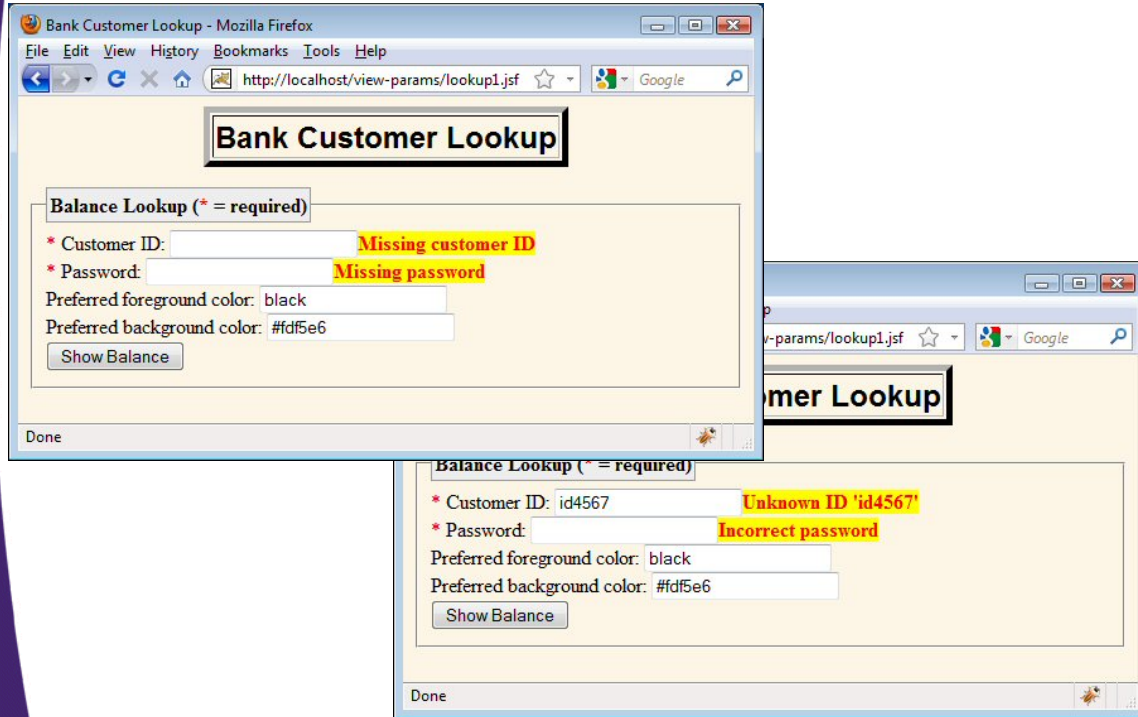
19

Results Page (show-customer1.xhtml)

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
<h:head> ...</h:head>
<h:body style="{colorPreferences.style}">
...
<p/>
<ul>
    <li>First name: #{bankForm1.customer.firstName}</li>
    <li>Last name: #{bankForm1.customer.lastName}</li>
    <li>ID: #{bankForm1.customer.id}</li>
    <li>Balance: $#{bankForm1.customer.balanceNoSign}</li>
</ul>
</h:body></html>
```

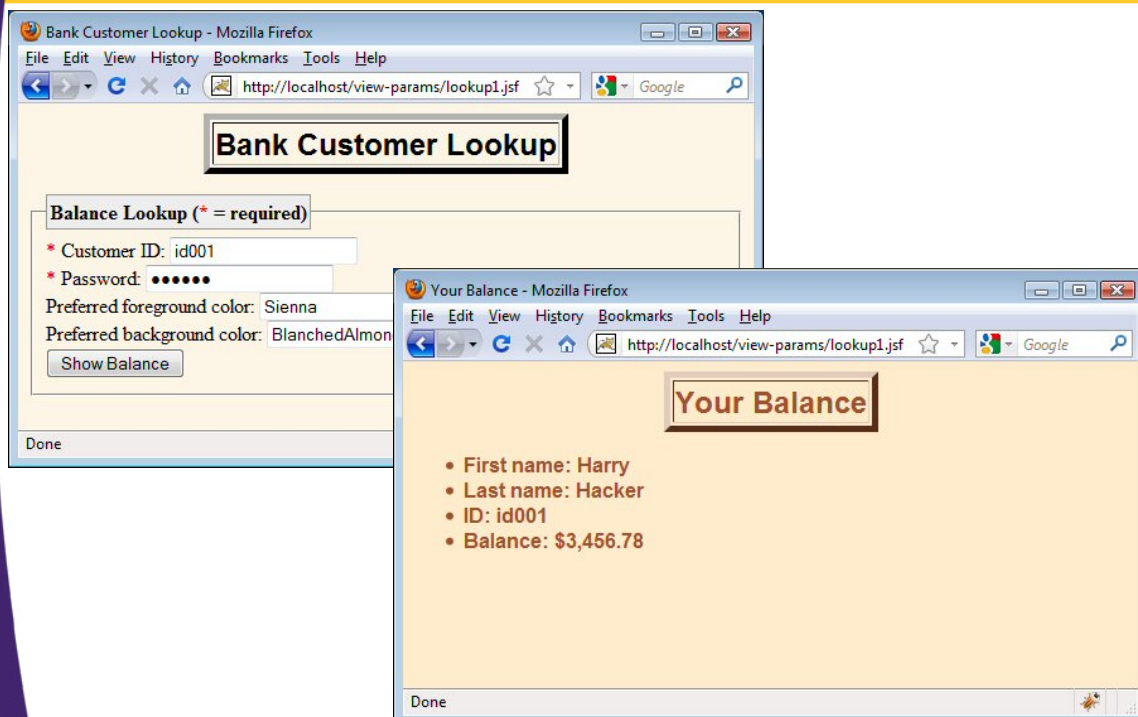
20

Results (Bad Data)



21

Results (Good Data)



22



Automatically Storing Request Params in Beans

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

f:viewParam

- **Basic syntax**
 - `<f:viewParam name="param1" value="#{bean.prop1}"/>`
 - Before page is rendered, find the value of the request parameter (query variable) called “param1”, and if non-null, pass it to the setProp1 method of bean.
 - The bean can be declared in any of the normal JSF ways.
- **Advanced options**
 - Additional attributes
 - required, requiredMessage, id, converterMessage
 - Can give messages for missing params, but page still rendered.
 - Converters
 - You can use standard JSF converters to convert input


```
<f:viewParam name="passed" value="#{exam.amountPassed}">
  <f:convertNumber type="percentage"/>
</f:viewParam>
```

Example

- **Color preferences bean**
 - Stores a foreground and background color
 - Has `getStyle` method that produces a CSS style string appropriate for the “style” attribute of `h:body`
- **Normal behavior via traditional JSF**
 - The colors are taken from session-scoped bean
- **Additional behavior via `f:viewParam`**
 - Colors can be overridden by supplying explicit request parameters.
 - Links containing these parameters can be bookmarked, so coming back to them later (when session is long expired) still uses the saved colors

25

New Input Form (lookup2.jsf)

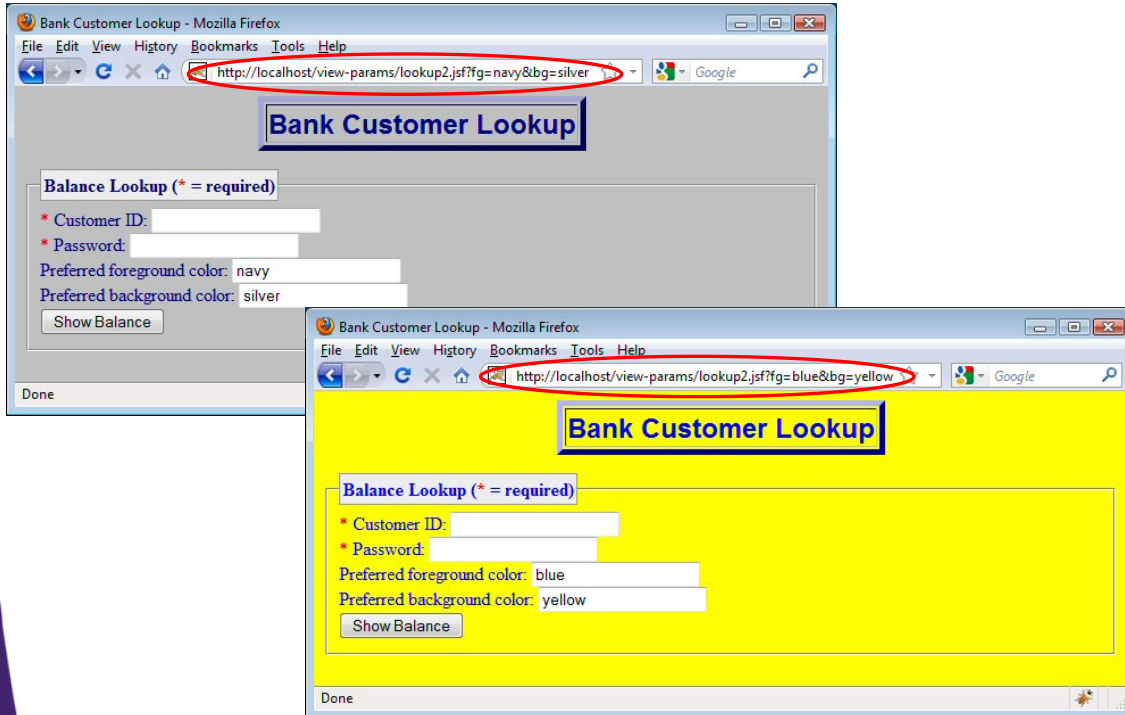
```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
  <f:metadata>
    <f:viewParam name="fg"
                value="#{colorPreferences.foreground}"/>
    <f:viewParam name="bg"
                value="#{colorPreferences.background}"/>
  </f:metadata>

  <!-- The rest is identical to lookup1.jsf -->

</html>
```

26

Results (Bookmarkable URLs)



27

© 2012 Marty Hall



Using `h:link` and `h:button`

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Idea

- **f:viewParam lets you *accept* request params**
 - So, your page can now accept “fg” and “bg” params
 - But how do you *send* those params?
- **h:link and h:button let you *send* params**
 - Basic syntax

```
<h:link outcome="blah?param1=val1&param2=val2"
value="Click here to go to blah page"/>
<h:button outcome="blah?param1=val1&param2=val2"
value="Click here to go to blah page"/>
```
 - Interpretation
 - When the user clicks on link or button, go to outcome “blah” (which could be blah.xhtml or could be mapped via a navigation rule in faces-config.xml) and pass along the param1 and param2 request params.
 - Note: to satisfy rules of XML, use “&”, not “&”

29

Using Value Expressions

- **Idea**
 - Either outcome or value can be `#{someBean.someProp}`
 - Note this is a shorthand for the `getSomeProp` method
 - This is not a method expression as with “action”
 - This lets you choose values dynamically
- **Understanding when code runs**
 - The getter methods are called when the page is loaded, not when the link is pressed
 - I.e., outcome is precalculated, not based on data in link
 - In fact, the main reason for doing this is to *put* data in link
 - So, it is not in any way equivalent to “action” for `h:commandLink` or `h:commandButton`

30

Examples: h:link (linkers.xhtml)

```
<ul>
  <li><h:link outcome="lookup2?fg=black&bg=white"
             value="Light"/>&nbsp; &nbsp; &nbsp;
</li>
  <li><h:link outcome="lookup2?fg=white&bg=black"
             value="Dark"/></li>
  <li><h:link outcome="lookup2?fg=dimgray&bg=lightgray"
             value="Medium"/></li>
  <li><h:link outcome="lookup2?fg=chocolate&bg=papayawhip"
             value="Foods"/></li>
  <li><h:link
             outcome="lookup2?fg=cornflowerblue&bg=cornsilk"
             value="Corny"/></li>
  <li><h:link
             outcome="lookup2?fg=#{colorUtils.randomForeground}&
             bg=#{colorUtils.randomBackground}" Line break added for readability.  
This is one long string in actual code.
             value="Random"/></li>
</ul> The random colors are computed when the page is loaded, not when the link is pressed.
```

31

Examples: h:button (linkers.xhtml)

```
<ul>
  <li><h:button outcome="lookup2?fg=black&bg=white"
              value="Light"/>&nbsp; &nbsp; &nbsp;
</li>
  <li><h:button outcome="lookup2?fg=white&bg=black"
              value="Dark"/></li>
  <li><h:button outcome="lookup2?fg=dimgray&bg=lightgray"
              value="Medium"/></li>
  <li><h:button outcome="lookup2?fg=chocolate&bg=papayawhip"
              value="Foods"/></li>
  <li><h:button
              outcome="lookup2?fg=cornflowerblue&bg=cornsilk"
              value="Corny"/></li>
  <li><h:button
              outcome="lookup2?fg=#{colorUtils.randomForeground}&
              bg=#{colorUtils.randomBackground}" Line break added for readability.  
This is one long string in actual code.
              value="Random"/></li>
</ul> The random colors are computed when the page is loaded, not when the button is pressed.
```

32

Examples: ColorUtils

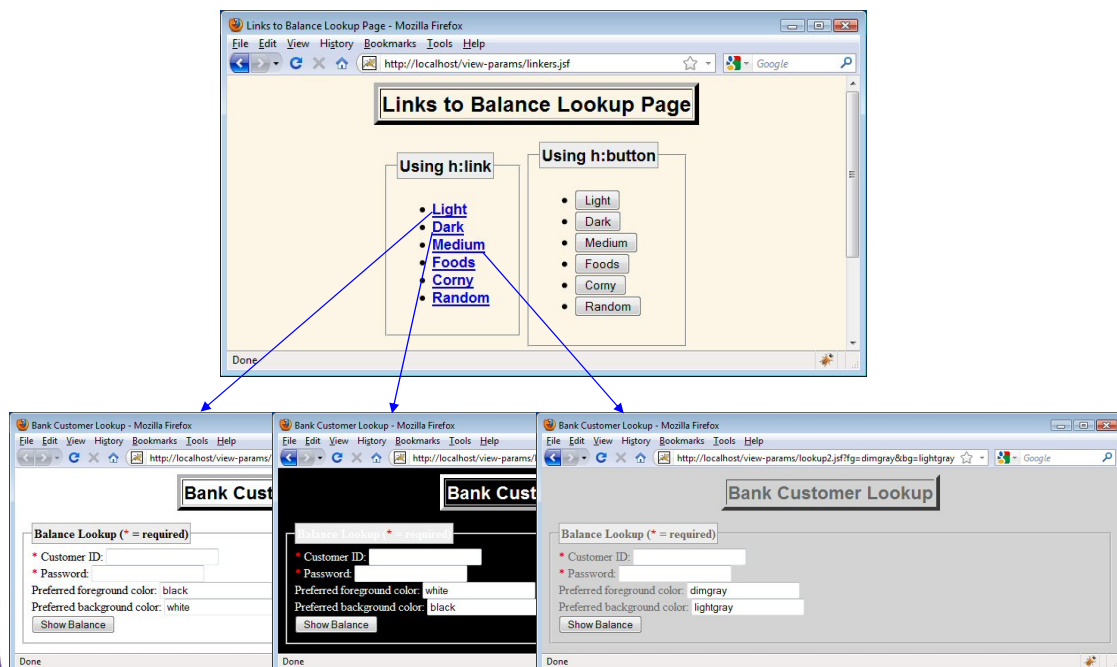
```
@ManagedBean
@ApplicationScoped
public class ColorUtils {
    private String[] foregrounds = {
        "DarkBlue", "DarkCyan", "DarkGoldenRod", "DarkGray",
        ... };
    private String[] backgrounds = {
        "LightBlue", "LightCyan", "LightGoldenRodYellow",
        ... };

    public String getRandomForeground() {
        return (RandomUtils.randomElement(foregrounds));
    }

    public String getRandomBackground() {
        return (RandomUtils.randomElement(backgrounds));
    }
}
```

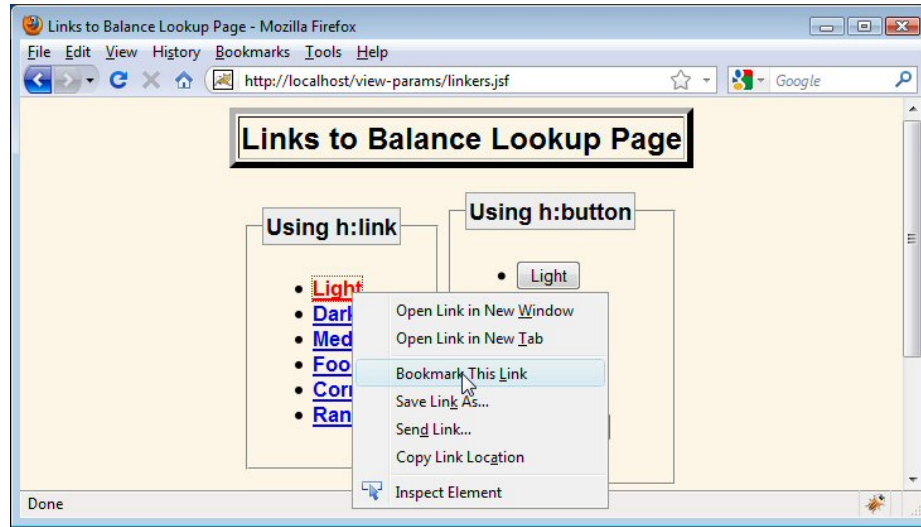
33

Results: h:link



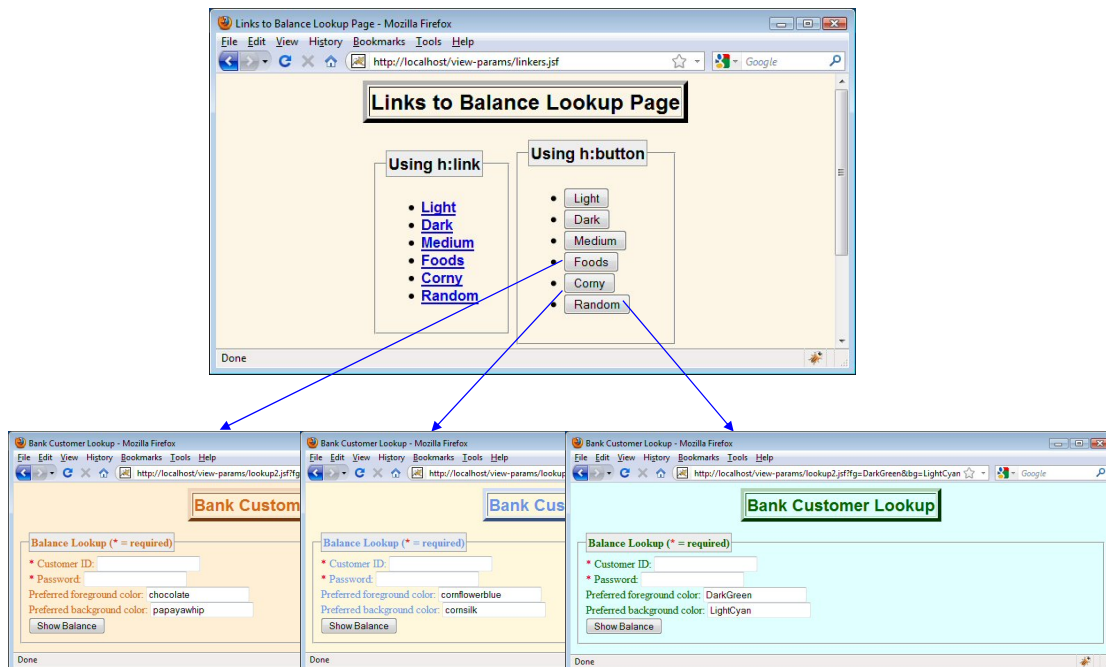
34

Results: h:link (Bookmarking)



35

Results: h:button



36



Sending Data to JSF from non-JSF Apps

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Idea

- **f:viewParam lets you *accept* request params**
 - Same syntax as shown earlier
- **Regular HTML forms let you *send* params**
 - In a JSF app, h:link and h:button are much more flexible
 - But, if you have a non-JSF app that needs to send data to the JSF app, you can just give names to the textfields to match the expected request parameter names.
 - However, note that the form will be missing many important JSF capabilities
 - Validation and redisplay with error messages
 - Prepopulation of form field values based on Java code
 - Rich components
 - Ajax

Normal HTML Form

```
<form action="lookup2.jsf">
```

Of course, a totally separate app would use an absolute URL for the action.

Foreground color:

```
<input type="text" name="fg"/><br/>
```

Background color:

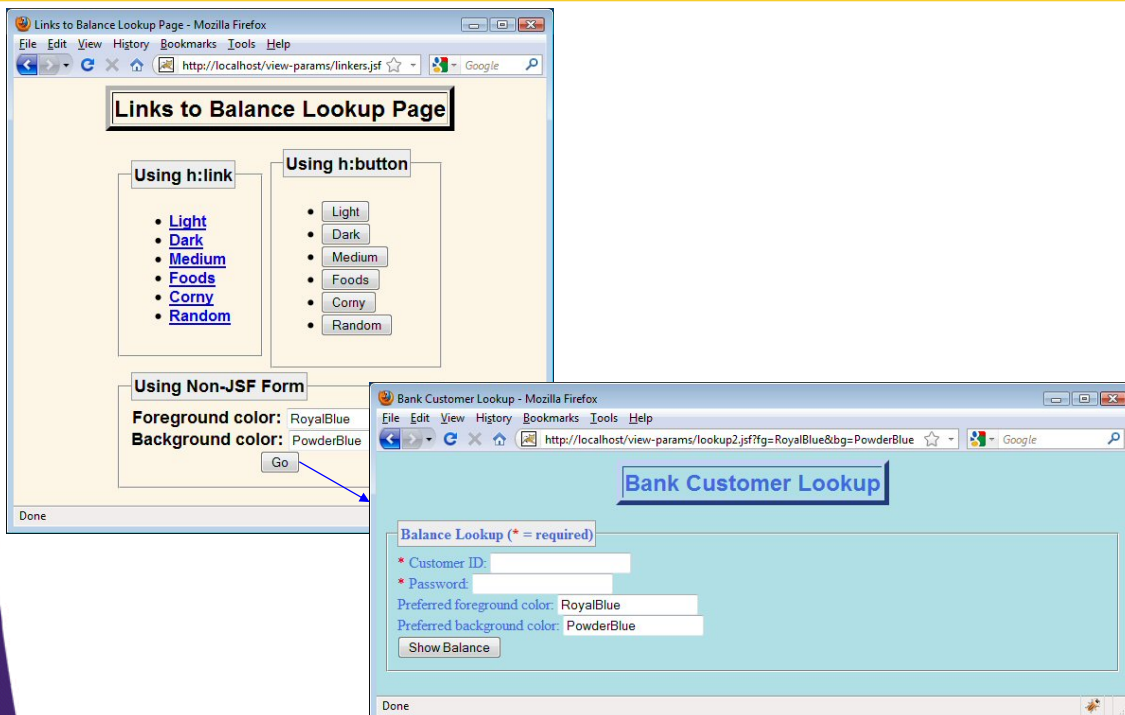
```
<input type="text" name="bg"/><br/>
```

```
<input type="submit" value="Go"/>
```

```
</form>
```

39

Results



40



POST-Redirect-GET

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Idea

- **JSF already lets you do redirects**
 - For implicit navigation
 - Add “faces-redirect=true” to end of outcome string
 - For explicit navigation rules in faces-config.xml
 - Add <redirect/> to the navigation-case
- **Normally done with session-scoped data**
 - Since request-scoped data not available on redirects
 - Because there are really two HTTP requests. First one gets a 302 status code and Location response header. Second HTTP request goes to location from the header.
- **You can do redirect with view params**
 - For implicit navigation
 - Add “includeViewParams=true” to end of outcome string
 - For explicit navigation rules in faces-config.xml
 - Add <redirect include-view-params="true"/> to nav case

Problems

- **Simple usage results in static linking**
 - View params are used are determined by the results page.
 - So, if results page has params named foo and bar linked to `#{bean.foo}` and `{bean.bar}`, JSF will call `getFoo` and `getBar` and pass them along as the foo and bar params.
 - Bookmarking explicit values is not useful if properties are dynamic
 - Bookmarking a link with explicit colors is useful
 - Bookmarking a link with bank account balance is not useful, since the balance amount is in the link. The user really wants to link to the balance for a particular ID.
- **Dynamic links require changing business logic**
 - For example, for the balance
 - You send the ID as a param, then `setId` needs to look up the associated customer. Not the way you would normally design it.
 - Be sure GET is even appropriate to the scenario!
 - Never use it for situations with passwords, credit cards, etc.

43

Static Linking: Bean

```
@ManagedBean
public class BankForm3 extends BankForm1 {
    public String findBalance() {
        customer = lookupService.findCustomer(customerId);
        FacesContext context = FacesContext.getCurrentInstance();
        if (customer == null) {
            String message =
                String.format("Unknown ID '%s'", customerId);
            context.addMessage("customerId", new FacesMessage(message));
        }
        if (context.getMessageList().size() > 0) {
            return(null);
        } else {
            String result =
                "show-customer3?faces-redirect=true" +
                "&includeViewParams=true";
            return(result);
        }
    }
}
```

44

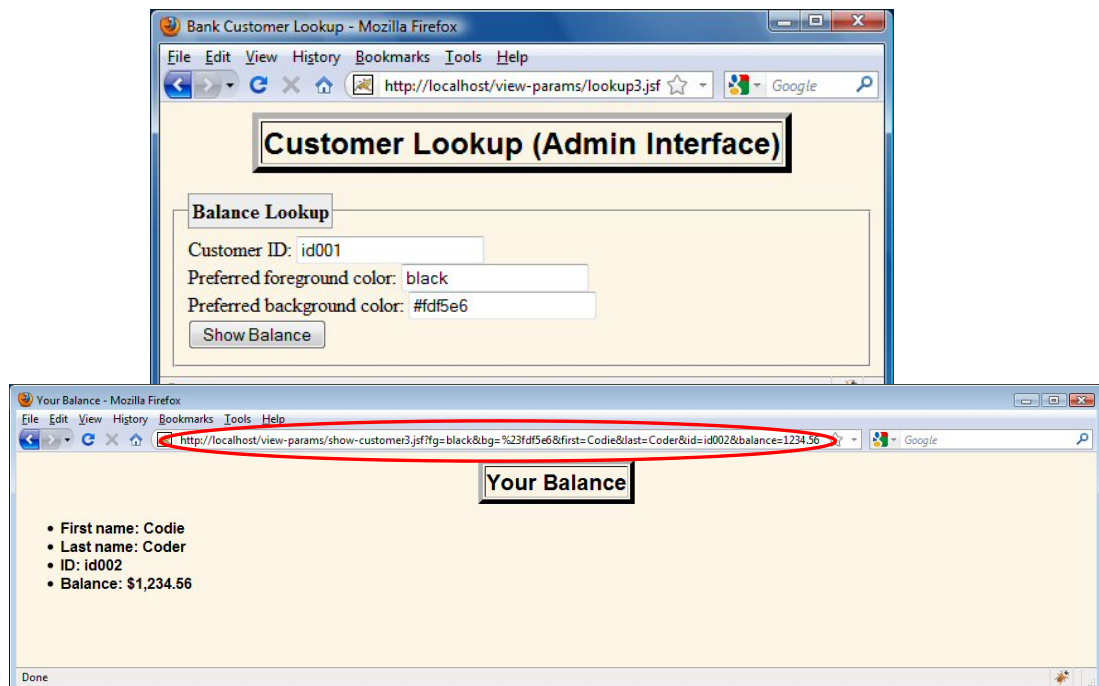
Static Linking: Results Page

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
  <f:metadata>
    <f:viewParam name="fg" value="#{colorPreferences.foreground}"/>
    <f:viewParam name="bg" value="#{colorPreferences.background}"/>
    <f:viewParam name="first" value="#{bankForm3.customer.firstName}"/>
    <f:viewParam name="last" value="#{bankForm3.customer.lastName}"/>
    <f:viewParam name="id" value="#{bankForm3.customer.id}"/>
    <f:viewParam name="balance" value="#{bankForm3.customer.balance}"/>
  </f:metadata>
  ...
  <ul>
    <li>First name: #{bankForm3.customer.firstName}</li>
    <li>Last name: #{bankForm3.customer.lastName}</li>
    <li>ID: #{bankForm3.customer.id}</li>
    <li>Balance: $#{bankForm3.customer.balanceNoSign}</li>
  </ul>
  ...

```

45

Static Linking: Results



46

Dynamic Linking: Bean (Customer ID Property)

```
@ManagedBean
public class BankForm4 extends BankForm1 {
    @Override
    public void setCustomerId(String customerId) {
        this.customerId = customerId;
        customer = lookupService.findCustomer(customerId);
    }
}
```

Setting the ID now sets the customer. There is no problem with this approach if you go through the normal business logic. However, this parameter is also set via a view parameter in the results page (which is accessible via GET). This means that a direct link or a bookmark could have an illegal customer ID that is not checked by the business logic.

47

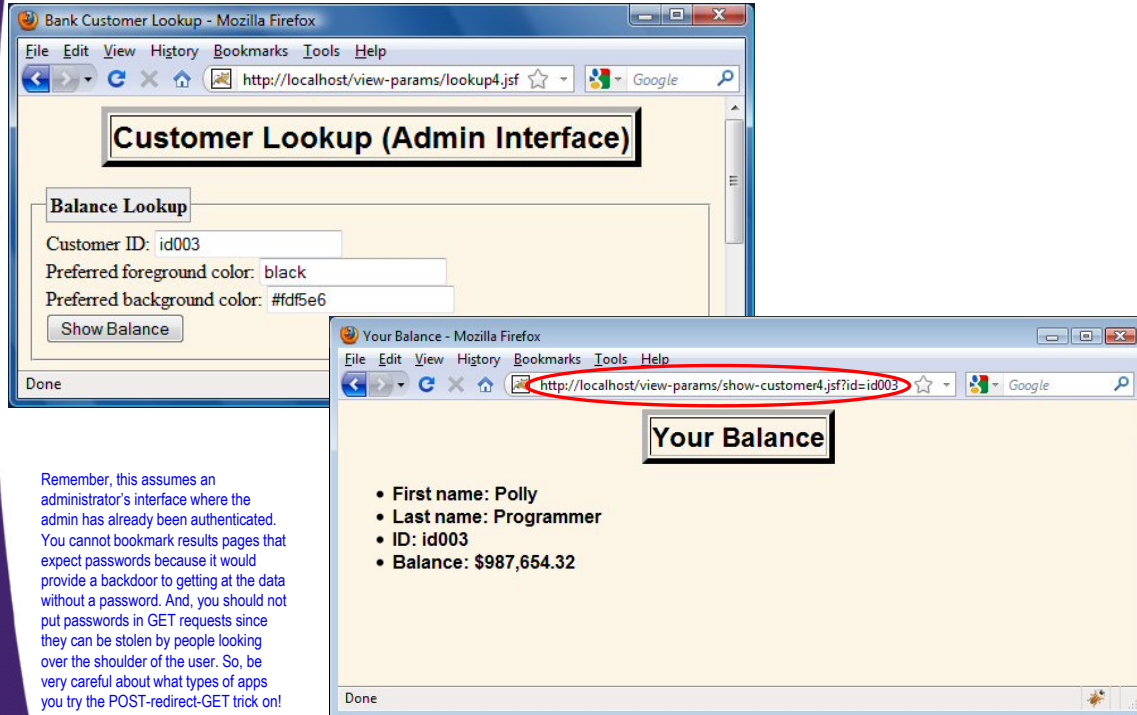
Dynamic Linking: Bean (Action Controller Method)

```
public String findBalance() {
    FacesContext context =
        FacesContext.getCurrentInstance();
    if (customer == null) {
        String message =
            String.format("Unknown ID '%s'", customerId);
        context.addMessage("customerId",
            new FacesMessage(message));
    }
    if (context.getMessageList().size() > 0) {
        return(null);
    } else {
        String result =
            "show-customer4?faces-redirect=true" +
            "&includeViewParams=true";
        return (result);
    }
}
```

The customer is already set by the time the action controller runs. Splitting the logic like this makes the code more difficult to maintain.

48

Dynamic Linking: Results

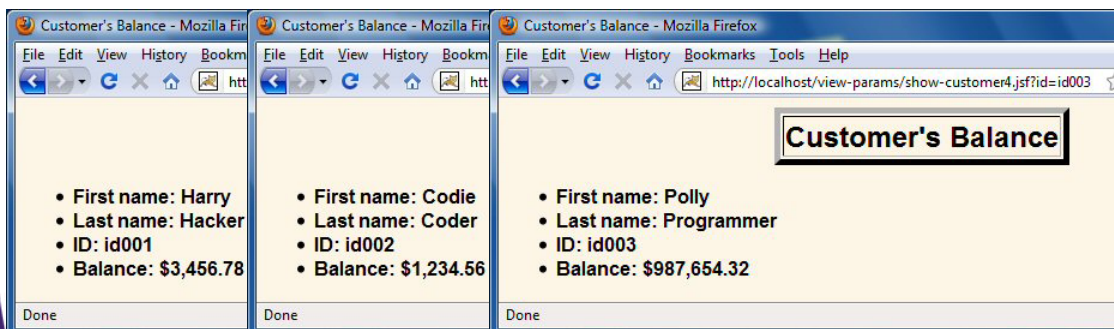


Remember, this assumes an administrator's interface where the admin has already been authenticated. You cannot bookmark results pages that expect passwords because it would provide a backdoor to getting at the data without a password. And, you should not put passwords in GET requests since they can be stolen by people looking over the shoulder of the user. So, be very careful about what types of apps you try the POST-redirect-GET trick on!

49

Dynamic Linking and Bookmarks

```
<ul>  
  <li><h:link outcome="show-customer4?id=id001"  
    value="Harry"/></li>  
  <li><h:link outcome="show-customer4?id=id002"  
    value="Polly"/></li>  
  <li><h:link outcome="show-customer4?id=id003"  
    value="Codie"/></li>  
</ul>
```



50



Wrap-up

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Summary

- **Read a request param and store in bean**

```
<f:viewParam name="param1" value="#{bean.prop1}"/>
```
- **Send request params to JSF pages**

```
<h:link outcome="blah?param1=val1&param2=val2"
value="Click here to go to blah page"/>
<h:button outcome="blah?param1=val1&param2=val2"
value="Click here to go to blah page"/>
<form action="blah">
  <input type="text" name="param1"/>
  <input type="text" name="param2"/> ...
</form>
```
- **POST-redirect-GET**
 - Add “redirect=true&includeViewParams=true”
 - Be careful about reorganizing business logic and exposing data that should not be accessible via GET



Questions?

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.