Validating User Input
Part 2: Advanced Features
JSF 2.2 Version

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

Customized Java EE Training: http://courses.coreservlets.com/
Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at your location.

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
Basics from Previous Section

- **Example code**
  ```html
  <h:inputText value="..." required="true" requiredMessage="..."
  converterMessage="..." validatorMessage="..." id="someId">
    <f:validateBlah.../>
  </h:inputText>
  <h:message for="someId"/>
  ```

- **Strategies**
  - Always supply custom error messages
    - Use `requiredMessage` when `required="true"
    - Use `converterMessage` with non-String bean properties
    - Use `validatorMessage` when using `f:validateBlah`
  - Use `h:panelGrid` to keep error messages next to fields
  - Put “See Below” warning at top if form is very long

Topics in This Section

- **Manual validation**
  - Checking in the action controller method

- **Using a custom validator method**
  - `<h:inputBlah ... validator="#{someBean.someMethod}"/>

- **Localizing (internationalizing) error messages**
  - `blahMessage="#{msgs.messageName}"`

- **Using the Apache MyFaces validators**
  - For URLs, email addresses, credit cards, cross-field comparisons

- **Using the Apache MyFaces validators with the PrimeFaces extended GUI components**
  - If using 3rd party libraries anyhow, usually use both
Manual Validation

Overview

• **Big idea**
  – Check for problems in the action controller method

• **Motivation**
  – When you need to compare two fields (main motivation)
  – When validation requires a result computed only in action controller

• **Steps**
  – If any values are missing or invalid
    • Store error messages using FacesContext.addMessage
    • Return null to cause form to be redisplayed
  – If all values are OK
    • Do business logic
    • Return normal navigation string
Detailed Steps

• **Have setter methods take strings**
  - So you are sure the action controller is reached
  - Do explicit conversion to other types in the code
    - Use try/catch blocks to handle illegal data

• **Check values in action controller**
  - If values are valid
    - Return normal outcomes
  - If values are missing or invalid
    - Store error messages using FacesContext.addMessage
    - Return null to cause form to be redisplayed

• **Display error messages in input form**
  - Normally use h:messages to put messages at top of form
    - If messages were stored with an ID, can also use<br>      <h:message for="theID"/>. Less common.

Creating Error Messages

• **Basic steps**
  
  ```java
  FacesMessage errorMessage = new FacesMessage("...");
  errorMessage.setSeverity(FacesMessage.SEVERITY_ERROR);
  ```

• **Severity**
  - SEVERITY_ERROR normally used for this type of error message
  - SEVERITY_INFO (default) used for informational messages, as in the dialog box example at the end
  - h:messages has options to use different style sheet names depending on the severity (errorClass, infoClass, etc.)

• **Summary and details**
  - FacesMessage constructor also has options for summary message (default) and detailed message
    - h:messages has options for which to show (showDetail, etc.)
Storing Error Messages

• Basic steps
  FacesContext.getCurrentInstance().addMessage(null, errorMessage);
  – null means that it is a global error message, not associated with the ID of any particular field. You always use h:messages to display these general messages.

• Associating error message with an ID
  – Instead of null, you can supply a String of the form "formID:fieldID".
    • Since form ID is normally generated automatically, you should use the id attribute of h:form.

Checking if Messages Are Set

• Normal approach
  – If at least one error message set, store FacesMessage and return null
  – If no error messages have been set, do business logic and return normal navigation string

• Option 1: check value of message
  – See if String has been set
    • Usually used if there is only one potential message

• Option 2: check length of message list
  – See if context.getMessageList().size() > 0
    • Usually used if there are several potential messages
    • This is essentially what we did in last tutorial section when we put conditional “Fix Errors Below” warning at top
Manual Validation: Example

- Idea
  - Gather checkin and checkout dates for reservation at the world-famous JSF Resort

- Validation tests
  - Dates must be supplied
    - Uses required and requiredMessage as done earlier
  - Entries must be legal dates
    - Uses converterMessage with f:convertDateTime
  - Checkin date must be today or later
    - Uses manual validation
  - Checkout day must be after checkin day
    - Uses manual validation

Managed Bean: Properties

```java
@ManagedBean
public class ReservationBean {
    private Date startDate, endDate;

    public Date getStartDate() {
        if (startDate == null) {
            startDate = new Date();
        }
        return(startDate);
    }

    public void setStartDate(Date startDate) {
        this.startDate = DateUtils.nextDay(startDate);
    }

    public Date getEndDate() {
        if (endDate == null) {
            endDate = DateUtils.nextDay(getStartDate());
        }
        return(endDate);
    }

    public void setEndDate(Date endDate) {
        this.endDate = DateUtils.nextDay(endDate);
    }
}
```
Managed Bean: 
Action Controller

```java
public String register() {
    String messageText = null;
    Date today = new Date();
    if (startDate.before(today)) {
        messageText = "Arrival cannot be in the past";
    } else if (!startDate.before(endDate)) {
        messageText = "Departure must be after arrival";
    }
    if (messageText != null) {
        startDate = null;
        endDate = null;
        FacesMessage errorMessage = new FacesMessage(messageText);
        errorMessage.setSeverity(FacesMessage.SEVERITY_ERROR);
        FacesContext.getCurrentInstance().addMessage(null,
            errorMessage);
        return(null);
    } else {
        putReservationInDatabase();
        return("show-dates-1");
    }
}
```

Helper Class: DateUtils

```java
public class DateUtils {
    public static String formatDay(Date date) {
        if (date == null) {
            return("");
        } else {
            return(String.format("%tA, %tB %te, %tY",
                date, date, date, date));
        }
    }

    public static Date nextDay(Date day) {
        long millisPerDay = 24 * 60 * 60 * 1000;
        return(new Date(day.getTime() + millisPerDay));
    }

    public static boolean between(Date testDate,
        Date startDate,
        Date endDate) {
        return(testDate.after(startDate) && testDate.before(endDate));
    }
    ...
}
```
The approach of using requiredMessage and converterMessage is covered in first tutorial section. Only when this fails is manual validation attempted.

f:convertDateTime converts Dates to Strings on display and Strings to Dates on submission. The format of the pattern is given in Javadoc for java.text.SimpleDateFormat.

getStartDay and getEndDay call DateUtils.formatDate. This is (arguably) easier and more reusable than doing the formatting directly in the results page.
Results: Missing or Malformed Dates

Results: Arrival in the Past
Results: Departure Before Arrival

Register for the JSF Resort

Example Using Manual Validation

Start date (mm/dd/yyyy): 12/15/2014
End date (mm/dd/yyyy): 12/14/2014
Register

Results: Good Data

Reservation Confirmed

- Arrival: Thursday, December 24, 2015
- Departure: Sunday, December 27, 2015
Using Custom Validator Method

Customized Java EE Training: http://courses.coreservlets.com/
Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at your location.

Making Custom Checks

- **Checking for missing data**
  - Use built-in required and requiredMessage

- **Checking illegal format of standard types**
  - Use non-String properties and built-in converterMessage

- **Checking ranges and regular expressions**
  - Use f:validateBlah and built-in validatorMessage

- **Comparing fields to each other**
  - Use manual validation in action controller method

- **Checking value not covered by f:validateBlah, but for single field only**
  - Write custom converter method
Steps: Outline

- **Java method**
  - Throw ValidatorException with a FacesMessage if validation fails. Do nothing if validation succeeds.
  - Method arguments:
    - FacesContext (the context)
    - UIComponent (the component being validated)
    - Object (the submitted value, but primitives use wrappers)

- **Input form**
  - For input component, specify method name explicitly
    - `<h:inputText id="someID" validator="#{someBean.someMethod}"/>
  - Use h:message or h:messages as before
    - `<h:message for="someID"/>

- **Warning**
  - Don’t use f:validateBlah & custom method on same field

Validator Method: Template

```java
public void validateSomeProperty
    (FacesContext context,
     UIComponent componentToValidate,
     Object value)
    throws ValidatorException {
    double submittedValue = (Double)value;
    if (someTestWith(submittedValue)) {
        String messageText = "...";
        FacesMessage errorMessage =
            new FacesMessage(messageText);
        errorMessage.setSeverity(FacesMessage.SEVERITY_ERROR);
        throw new ValidatorException(errorMessage);
    }
}
```

This example assumes the bean property is of type double or Double. Do similar typecasts for other bean property types.
Do not Mix f:validate with Custom Validation Method

- **f:validate** tag
  ```
  <h:inputText ... validatorMessage="warning">
    <f:validateLength minimum="6"/>
  </h:inputText>
  ```
  - The error message comes from validatorMessage

- **Custom validator method**
  ```
  <h:inputText validator="#{someBean.someMethod}">
  ```
  - The error message comes from someMethod

- **If you have both, validatorMessage wins**
  - And thus the error message from the validator method is never displayed. So, you cannot use f:validate and a custom validation method in the same input element
    - However, the Apache MyFaces validators have planned for this, and let you have multiple validators on the same field without using the validatorMessage attribute. See upcoming slide.

Validator Method: Example

- **Idea**
  - Due to unsanitary conditions, the health department shut down all rooms except 2 and 7. Enforce this.

- **Validator method (validateRoomNumber)**
  - Cast to Integer. Throw exception if value is anything other than 2 or 7.
  - Do nothing if validation passes

- **Input form**
  ```
  <h:inputText ... validator="#{reservationBean2.validateRoomNumber}"/>
  ```
Managed Bean: Properties and Action Controller

@ManagedBean
public class ReservationBean2 extends ReservationBean1 {
    private Integer roomNumber = 2;

    public Integer getRoomNumber() {
        return(roomNumber);
    }

    public void setRoomNumber(Integer roomNumber) {
        this.roomNumber = roomNumber;
    }

    ...

    @Override
    public String register() {
        String result = super.register();
        if (result != null) {
            result = "show-dates-2";
        }
        return(result);
    }
}

Managed Bean: Validator Method

public void validateRoomNumber(FacesContext context, 
                                        UIComponent componentToValidate, 
                                        Object value) 
    throws ValidatorException {
    int roomNumber = (Integer)value;
    if (((roomNumber != 2) && (roomNumber != 7)) {
        String messageText =
            String.format("Room %s temporarily unavailable", 
                           roomNumber);
        FacesMessage errorMessage = new FacesMessage(messageText);
        errorMessage.setSeverity(FacesMessage.SEVERITY_ERROR);
        throw new ValidatorException(errorMessage);
    }
}
Input Form  
(resort-registration-2.xhtml)

```xml
<h:form>
  <h:messages styleClass="error"/>
  <h:panelGrid columns="2" styleClass="formTable">
    <!-- Same date fields and tests as last example -->
    Requested room number:
    <h:inputText value="#{reservationBean2.roomNumber}" required="true"
      requiredMessage="Room number required"
      converterMessage="Illegal room number"
      validator="#{reservationBean2.validateRoomNumber}"/>
  </h:panelGrid>
  <h:commandButton value="Register"
                    action="#{reservationBean2.register}"/>
</h:form>
```

Results Page  
(show-dates-2.xhtml)

```xml
...<h1 class="title">Reservation Confirmed</h1>
<ul>
  <li>Arrival: #{reservationBean2.startDay}</li>
  <li>Departure: #{reservationBean2.endDay}</li>
  <li>Room number: #{reservationBean2.roomNumber}</li>
</ul>
...```
Results: Unavailable Room

Register for the JSF Resort

Example Using Custom Validator Method

- Start date (mm/dd/yyyy): 12/21/2015
- End date (mm/dd/yyyy): 12/27/2015
- Requested room number: 3

Register

Results: Available Room

Register for the JSF Resort

Example Using Custom Validator Method

- Room 3 temporarily unavailable
- Start date (mm/dd/yyyy): 12/24/2015
- End date (mm/dd/yyyy): 12/27/2015
- Requested room number: 3

Register

Reservation Confirmed

- Arrival: Thursday, December 24, 2015
- Departure: Sunday, December 27, 2015
- Room number: 2
Validator Components vs. Validator Methods

• **Validator components and methods similar**
  – Both let you do custom checks for single field

• **Method**
  – Can access other methods in main backing bean
  – Not easily reusable in forms that use other beans
  – Can have any method name
  – Specified with `<h:inputText … validator="…"/>

• **Component**
  – Cannot access methods in backing bean
  – More easily reusable in forms that use other beans
  – Method is named validate (same method signature)
  – Specified with `<f:validator validatorId="…"/>

Component Outline

• **Java class**
  @FacesValidator("someld")
  public class SomeValidator implements Validator
    @Override
    public void validate(…) … {
      // Same signature and approach as validator method
    }
  }

• **Input form**
  `<h:inputText …>
  <f:validator validatorId="someld"/>
  </h:inputText>`
Localizing (Internationalizing) Error Messages

Overview

- Error messages can come from properties
  - requiredMessage="#{msgs.someName}""
- Page can set a Locale
  - <f:view locale="#{localeBean.currentLocale}">
- Multiple versions of file can be defined
  - messages.properties, messages_es.properties,
    messages_es_MX.properties, messages_jp.properties, etc.
- Covered in section on properties files
  - Once you know that the error messages accept EL entries
    instead of only static strings, *everything* in this example is
    just an application of the techniques covered in earlier
    tutorial section on properties and I18N.
Steps for Localizing Validation Messages

- Create multiple similarly named files
  - blah.properties, blah_es.properties, blah_es_MX.properties
- Use f:view and locale attribute
  - `<f:view locale="#{localeBean.currentLocale}"/>
  - LocaleBean should be session scoped
  - getCurrentLocale returns value user set earlier
- Declare in faces-config with resource-bundle
  - base-name gives base file name
    - Version(s) matching Locale will be used automatically
  - var gives scoped variable (Map) that will hold result
- Refer to name in validation messages
  - requiredMessage="#{...}", converterMessage="#{...}", etc.

Quick Review of Chained Loading

- messages.properties
  - siteName=mysite.example.com
  - buttonLabel=Click here
- messages_es.properties
  - buttonLabel=Haga clic aquí
- messages_fr.properties
  - buttonLabel=Cliquez ici
- messages_fr_CA.properties
  - buttonLabel=Se il vous plaît, cliquez ici

Recall from lecture on properties file that we do not need to repeat entries from earlier files if they do not change. So, siteName is not repeated.
Quick Review:
faces-config.xml

```xml
<?xml version="1.0"?>
<faces-config ...>
  <application>
    <resource-bundle>
      <base-name>messages</base-name>
      <var>msgs</var>
    </resource-bundle>
  </application>
</faces-config>
```

Reminder: file is assumed to be in CLASSPATH and have a .properties extension, so “messages” above means src/messages.properties in Eclipse (WEB-INF/classes/messages.properties at deploy time)

Also recall that we refer to base file name only (messages), not the localized names (messages_es, etc.)

---

Quick Review: Facelets Page

```html
<!DOCTYPE ...>
<html ... xmlns:f="http://xmlns.jcp.org/jsf/core">
  <f:view locale="#{localeBean.currentLocale}">
    ... 
    #{msgs.siteName} 
    ... 
    #{msgs.buttonLabel} 
    ... 
  </f:view></html>
```
Quick Review:
If Current Locale is...

• Spanish (es, es_MX, es_PR, etc.)
  – siteName is “mysite.example.com”
  – buttonLabel is “Haga clic aquí”

• Canadian French (fr_CA)
  – siteName is “mysite.example.com”
  – buttonLabel is “Se il vous plaît, cliquez ici”

• Any other French (fr, fr_FR, fr_HT, etc.)
  – siteName is “mysite.example.com”
  – buttonLabel is “Cliquez ici”

• Anything else
  – siteName is “mysite.example.com”
  – buttonLabel is “Click here”

Example:
Page to Compute Sums

• Idea
  – User can enter two numbers
  – Results page shows the sum
  – All text in input form and results page come from properties file.
    • This includes requiredMessage and converterMessage for the two number fields

• I18N
  – Page comes up in English by default
  – User can click on Spanish flag to switch Locale to es
    • Input form and results page then come up in Spanish
    • Validation error messages are also in Spanish
    • Bean is session scoped, so choice is retained for later
Example: Main Managed Bean (Backing Bean)

@ManagedBean
class CalculatorBean {
    private int firstNumber, secondNumber;

    public int getFirstNumber() { return(firstNumber); }

    public void setFirstNumber(int firstNumber) {
        this.firstNumber = firstNumber;
    }

    public int getSecondNumber() { return(secondNumber); }

    public void setSecondNumber(int secondNumber) {
        this.secondNumber = secondNumber;
    }

    public int getSum() { return(firstNumber + secondNumber); }

    public String computeSum() {
        return("show-sum");
    }
}

Example: Locale Bean

@ManagedBean
@SessionScoped
class LocaleBean implements Serializable {
    private final static Locale ENGLISH = new Locale("en");
    private final static Locale SPANISH = new Locale("es");
    private Locale currentLocale=ENGLISH;

    public Locale getCurrentLocale() {
        return(currentLocale);
    }

    public String setEnglish() {
        currentLocale=ENGLISH;
        return null;
    }

    public String setSpanish() {
        currentLocale=SPANISH;
        return null;
    }
}
Example: Properties Files

- **src/messages.properties**
  - inputPageTitle=Calculator
  - legend=I18N Example
  - num1Prompt=First number:
  - num2Prompt=Second number:
  - numRequiredMessage=Number required
  - numConverterMessage=Must be whole number
  - buttonLabel=Compute Sum
  - resultPageTitle=Sum

- **src/messages_es.properties**
  - inputPageTitle=Calculadora
  - legend=Ejemplo de I18N
  - num1Prompt=Primero número:
  - num2Prompt=Segundo número:
  - numRequiredMessage=Número requerido
  - numConverterMessage=Debe ser número entero
  - buttonLabel=Calcula Suma
  - resultPageTitle=Suma

Example: Input Form (calculator.xhtml)

```html
...<f:view locale="#{localeBean.currentLocale}">...
<h:form>
  <h:commandLink action="#{localeBean.setEnglish}">
    <img src="images/en_flag.png"/>
  </h:commandLink> &nbsp;
  <h:commandLink action="#{localeBean.setSpanish}">
    <img src="images/es_flag.png"/>
  </h:commandLink>
  <h:panelGrid columns="3" styleClass="formTable">
    #{msgs.num1Prompt}
    <h:inputText value="#{calculatorBean.firstNumber}" required="true"
                 requiredMessage="#{msgs.numRequiredMessage}"
                 converterMessage="#{msgs.numConverterMessage}"
                 id="num1"/>
    <h:message for="num1"/>
  ... ...
  </h:panelGrid>
</f:view></html>
```
Example: Results Page (show-sum.xhtml)

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:f="http://xmlns.jcp.org/jsf/core"
     xmlns:h="http://xmlns.jcp.org/jsf/html">
    <f:view locale="#{localeBean.currentLocale}">
        <h:head><title>#{msgs.resultPageTitle}</title>
            <link href='./css/styles.css' rel='stylesheet' type='text/css'/>
        </h:head>
        <h:body>
            <h1 class='title'>#{msgs.resultPageTitle}</h1>
            <h2>#{calculatorBean.firstNumber} + #{calculatorBean.secondNumber} = #{calculatorBean.sum}</h2>
        </h:body>
    </f:view></html>
Results: Spanish

Using the Apache MyFaces Extended Validators
Overview

• **Standard JSF**
  – Validators easy to use
  – Easy to write your own validator methods
  – Very sparse set of builtin validators

• **PrimeFaces**
  – Zillions of great-looking GUI components
  – No additional validators

• **Apache MyFaces Commons Validators**
  – Additional validators based on Apache Commons
    • Validates URLs, email addresses, credit card numbers, and does cross-field validation (eliminating the need for most manual validation!)

Apache MyFaces Commons Validators

• **Documentation**
  – Tag documentation
    • http://myfaces.apache.org/commons20/myfaces-validators20/tagdoc.html
  – Latest Apache Commons validation code
    • http://commons.apache.org/proper/commons-validator/

• **Downloads**
  • http://myfaces.apache.org/myfaces-commons/download.html

• **Many JAR files needed**
  • http://myfaces.apache.org/commons20/myfaces-validators20/dependencies.html
  – This list is a bit hard to follow, and sadly you need many separate JAR files: commons-beanutils-xyz.jar, commons-collections-xy.jar, commons-digester-xy.jar, commons-logging-xy.jar, commons-validator-xy.jar, myfaces-commons-utils-xyz.jar, myfaces-validators-xyz.jar, oro-xy.jar, xml-apis-xyz.jar
Most Useful New Validators and Their Most Common Attributes

- **mcv:validateCompareTo**
  - for: id of the field to compare this field to
  - operator: one of a list of possible comparison operators

- **mcv:validateCreditCard**
  - Usually used without attributes, but you can restrict the type of credit cards accepted
    - For practicing with this validator, use 4111111111111111

- **mcv:validateEmail**
  - Usually used without attributes

- **mcv:validateUrl**
  - Usually used without attributes

- **Specifying the error message**
  - Use validatorMessage of h:inputBlah unless you have multiple validators on the same field, in which case use message of mcv:validateBlah. See next slide.

The message Attribute

- **Normal usage**
  - Use validatorMessage attribute to give error message
    
    ```
    <h:inputText ... validatorMessage="Malformed Email Address">
    <mcv:validateEmail/>
    </h:inputText>
    ```

- **Usage when multiple validators on same field (e.g., two email entries must match)**
  - Use message attribute of MyFaces tag for error message
    
    ```
    <h:inputText ... >
    <mcv:validateEmail message="Malformed Email Address"/>  
    <mcv:validateCompareTo operator="eq" forId="email1" message="Two email entries must match"/> 
    </h:inputText>
    ```
Example: Page to Report Website Errors

• Idea
  – Let users report mistakes on commercial Web site. Reward verified reports with $10.

• Fields and validation tests
  – Name for crediting report on Web site
    • Non-empty
  – URL that has the problem
    • Non-empty. Legal URL.
  – Email address to which to send verification
    • Non-empty. Legal email address.
  – Credit card number that will get $10 (entry 1)
    • Non-empty. Legal credit card number.
  – Credit card number (repeated)
    • Non-empty. Matches previous entry.
  – Report description
    • Non-empty.

Example: JAR Files Loaded

The PrimeFaces JAR is not used in this example, but is used in the next example (which also uses the extended validators).
**Example: Managed Bean**

```java
@ManagedBean
public class ErrorReportBean1 {
    private String name, emailAddress, url, problemDescription,
        cardNumber="", cardNumberConfirmation;

    // Getters and setters for all fields

    public String getAbbreviatedCardNumber() {
        // Returns last four digits of card number
    }

    public String sendReport() {
        storeReportInDatabase();
        return("report-confirmation");
    }
}
```

**Example: Input Form (bug-report-1.xhtml) Part 1**

```xml
<!DOCTYPE html ...>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://xmlns.jcp.org/jsf/core"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:mcv="http://myfaces.apache.org/commons/validators"
    ...>
```
Example: Input Form (bug-report-1.xhtml) Part 2

```xml
<h:form>
<h:panelGrid columns="3" styleClass="formTable">
   Name or nickname:
   <h:inputText value="#{errorReportBean1.name}" required="true"
requiredMessage="Name required (for crediting report)"
   id="name"/>
   <h:message for="name"/>

   URL containing the problem:
   <h:inputText value="#{errorReportBean1.url}" required="true"
requiredMessage="URL required (page that has error)"
   validatorMessage="Invalid URL"
   id="url">
   <mcv:validateUrl/>
   </h:inputText>
   <h:message for="url"/>
</h:panelGrid>
</h:form>
```

Example: Input Form (bug-report-1.xhtml) Part 3

```xml
Email address:
<h:inputText value="#{errorReportBean1.emailAddress}" required="true"
requiredMessage="Email address required (for confirmation)"
   validatorMessage="Invalid email address"
   id="email">
   <mcv:validateEmail/>
   </h:inputText>
   <h:message for="email"/>
```
Example: Input Form (bug-report-1.xhtml) Part 4

Credit card number:
<h:inputSecret value="#{errorReportBean1.cardNumber}"
    required="true"
    requiredMessage=
        "Credit card required (for $10 'thank you')"
    validatorMessage="Invalid credit card number"
    id="card1">
    <mcv:validateCreditCard/>
</h:inputSecret>
<h:message for="card1"/>

Reenter card number:
<h:inputSecret value="#{errorReportBean1.cardNumberConfirmation}"
    required="true"
    requiredMessage=
        "Credit card required (for $10 'thank you')"
    id="card2">
    <mcv:validateCreditCard message="Invalid credit card number"/>
    <mcv:validateCompareTo operator="eq" forId="card1"
        message="Must match first card entry"/>
</h:inputSecret>
<h:message for="card2"/>

Example: Input Form (bug-report-1.xhtml) Part 5

Problem description:
<h:inputTextarea
    value="#{errorReportBean1.problemDescription}"
    required="true"
    requiredMessage=
        "At least brief description required"
    id="description">
</h:message for="description"/>
</h:panelGrid>
<h:commandButton value="Submit Report"
    action="#{errorReportBean1.sendReport}"/>
Example: Results Page
(report-confirmation.xhtml)

...<h1 class="title">Report Summary</h1><p>Here is the data you sent:</p><ul><li>Name: #{errorReportBean1.name}</li><li>URL: #{errorReportBean1.url}</li><li>Email address: #{errorReportBean1.emailAddress}</li><li>Credit card number:<i>xxxx</i>-#{errorReportBean1.abbreviatedCardNumber}</li><li>Problem description:<i>#{errorReportBean1.problemDescription}</i></li></ul><p>A $10 thank you will be credited to your card upon confirmation of the issue.</p>...

Results: Bad Data 1
Results: Bad Data 2

```
Web Site Bug Reports

Thanks for alerting us to an error on our Web site. All confirmed reports are given a $10 cash 
“thank you.” Your name or nickname will be credited with the report on our bug-fix page.

Example Using the MyFaces Common Validators:

- Name or nickname: Canned Casey
- URL containing the problem: http://yoursite.com
- Email address: casey@example.com
- Credit card number: ********
- Recruit card number: ***
- Problem description: Misspelled company name

Submit Report
```

Results: Good Data

```
Report Summary

Here is the data you sent:

- Name: Tim Cook
- URL: http://windows.microsoft.com/en-us/windows/home
- Email address: timmee2@apple.com
- Credit card number: xxxx-1111
- Problem description: I don’t like Windows.

A $10 thank you will be credited to your card upon confirmation of the issue.
```
Using the Apache MyFaces Extended Validators with PrimeFaces

Customized Java EE Training: http://courses.coreservlets.com/
Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at your location.

Overview

• Big ideas
  – Using third-party validators does not interfere in any way with using a third-party GUI components that were developed by a different organization
  – Most people that use one (validators) also use the other (GUIs). PrimeFaces is arguably the most popular choice.

• Example
  – Previous example will be redone with PrimeFaces, just to show the look. Little or no explanation is provided.
  • But all techniques and components used are covered in the tutorial sections on PrimeFaces

• Tutorial sections on PrimeFaces
Example: Managed Bean

@ManagedBean
public class ErrorReportBean2 extends ErrorReportBean1 {
    @Override
    public String sendReport() {
        super.sendReport();
        String confirmationMessage = String.format(
            "%s (%s) reported "%s" for %s. On confirmation " +
            "of problem, $10 will be credited to xxxx-%s."
        ,
            getName(), getEmailAddress(), getProblemDescription(),
            getUrl(), getAbbreviatedCardNumber());
        FacesContext context = FacesContext.getCurrentInstance();
        FacesMessage fMessage = new FacesMessage(confirmationMessage);
        context.addMessage(null, fMessage);
        return(null);
    }
}

The idea is that the main page will always pop up a dialog box with p:messages as the body of the dialog box. If there was a
validation error, the text in the dialog box will be in the "error" color and will show the validation messages, and the action
controller method will never be called. If validation passes, then this method will be called, the text in the dialog box will be in
the "info" color and will show the above text.

Example: Input Form

(bug-report-2.xhtml) Part 1

<h:form>
    <h:panelGrid columns="2" styleClass="formTable">
        Name or nickname:
        <p:inputText value="#{errorReportBean2.name}"
            required="true"
            requiredMessage="Name required (for crediting report)"
            id="name"/>
        
        URL containing the problem:
        <p:inputText value="#{errorReportBean2.url}"
            required="true"
            requiredMessage="URL required (page that has error)"
            validatorMessage="Invalid URL"
            id="url">
            <mcv:validateUrl/>
        </p:inputText>
    </h:panelGrid>
</h:form>
Example: Input Form (bug-report-2.xhtml) Part 2

Email address:
<p:inputText value="#{errorReportBean2.emailAddress}" required="true"
    requiredMessage="Email address required (for confirmation)"
    validatorMessage="Invalid email address"
    id="email">
    <mcv:validateEmail/>
</p:inputText>

Credit card number:
<p:password value="#{errorReportBean2.cardNumber}" required="true"
    requiredMessage="Credit card required (for $10 'thank you')"
    validatorMessage="Invalid credit card number"
    id="card1">
    <mcv:validateCreditCard/>
</p:password>

Reenter card number:
<p:password value="#{errorReportBean2.cardNumberConfirmation}" required="true"
    requiredMessage="Second credit card required (to confirm match)"
    validatorMessage="The two credit card numbers must match"
    id="card2">
    <mcv:validateCompareTo operator="eq" forId="card1"/>
</p:password>

Problem description:
<p:inputTextarea value="#{errorReportBean2.problemDescription}" required="true"
    requiredMessage="At least brief problem description required"
    id="description"/>
</h:panelGrid>
The button uses Ajax, since that is the default for PrimeFaces buttons. If validation fails, the action controller method (sendReport) is never called, so the text shown by p:messages gives the validation errors. If validation passes, then the action controller method (sendReport) is called, so the text shown by p:messages is the informational text set in sendReport.
Results: Good Data

Web Site Bug Reports

Thanks for alerting us to an error on our Web site. All confirmed reports are given a $10 cash "thank you". Your name or nickname will be credited with the report on our bug-fix page.

Example Using the MyFaces Commons Validators with PrimeFaces

Report Summary

Tim Cook (timmie2@apple.com) reported "I don't like Windows" for http://windows.microsoft.com/en-us/windows/home. On confirmation of problem, $10 will be credited to xxxx:1111.

Submit Report

Wrap-Up

Customized Java EE Training: http://courses.coreservlets.com/
Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at your location.
Summary

• **Manual validation (in action controller method)**
  – For comparing *two or more* fields to each other
    
    ```java
    FacesMessage errorMessage = new FacesMessage("...");
    errorMessage.setSeverity(FacesMessage.SEVERITY_ERROR);
    FacesContext.getCurrentInstance().addMessage(null, errorMessage);
    return(null);
    ```

• **Using a custom validator method**
  – For checking *single* field for something not done with f:validateBlah
    
    ```html
    <h:inputBlah ... validator="#{someBean.someMethod}"/>
    ```

• **Localizing (internationalizing) error messages**
  – `blahMessage="#{msgs.messageName}"`

• **Using the Apache MyFaces Commons validators**
  – For URLs, email addresses, credit cards, cross-field comparisons
  – Typically used with PrimeFaces as well