

The NetBeans E-commerce Tutorial - Adding Language Support

Tutorial Contents

1. [Introduction](#)
2. [Designing the Application](#)
3. [Setting up the Development Environment](#)
4. [Designing the Data Model](#)
5. [Preparing the Page Views and Controller Servlet](#)
6. [Connecting the Application to the Database](#)
7. [Adding Entity Classes and Session Beans](#)
8. [Managing Sessions](#)
9. [Integrating Transactional Business Logic](#)
10. **Adding Language Support**
 - [Understanding Resource Bundles](#)
 - [Making Pages Multilingual](#)
 - [Implementing a Language Toggle](#)
 - [See Also](#)
11. [Securing the Application](#)
12. [Testing and Profiling](#)
13. [Conclusion](#)

The goal of this tutorial unit is to demonstrate how to enable language support for a web application. "Language support" here refers to the ability to display page views according to the customer-specified languages. Within the context of the `AffableBean` application, we have agreed to provide support for both English and Czech, as per the previously outlined [customer requirements](#).



In order to accomplish this, you rely on Java's support for internationalization. You create a *resource bundle* for each language and let the Java runtime environment determine the appropriate language for incoming client requests. You also implement a 'language toggle' to enable users to switch the languages manually.

The NetBeans IDE provides special support for localizing application content. This includes a Customizer dialog that enables you to add new locales to an existing resource bundle base name, as well as a special Properties editor that lets you view and edit key-value pairs for all locales in a table layout. These are both utilized in this tutorial.

You can view a live demo of the application that you build in this tutorial: [NetBeans E-commerce Tutorial Demo Application](#).

Software or Resource	Version Required
NetBeans IDE	Java bundle, 6.8 or 6.9
Java Development Kit (JDK)	version 6
GlassFish server	v3 or Open Source Edition 3.0.1
MySQL database server	version 5.1
AffableBean project	snapshot 8
AffableBean project	snapshot 9

Notes:

- The NetBeans IDE requires the Java Development Kit (JDK) to run properly. If you do not have any of the resources listed above, the JDK should be the first item that you download and install.
- The NetBeans IDE Java Bundle includes Java Web and EE technologies, which are required for the application you build in this tutorial.
- The NetBeans IDE Java Bundle also includes the GlassFish server, which you require for this tutorial. You could [download the GlassFish server independently](#), but the version provided with the NetBeans download has the added benefit of being automatically registered with the IDE.
- You can follow this tutorial unit without having completed previous units. To do so, see the [setup instructions](#), which describe how to prepare the database and establish connectivity between the IDE, GlassFish, and MySQL.

Understanding Resource Bundles

In Java, a resource bundle is a representation of the `java.util.ResourceBundle` class. As stated in the Javadoc,

Resource bundles contain locale-specific objects. When your program needs a locale-specific resource, a `String` for example, your program can load it from the resource bundle that is appropriate for the current user's locale. In this way, you can write program code that is largely independent of the user's locale isolating most, if not all, of the locale-specific information in resource bundles.

This allows you to write programs that can:

- *be easily localized, or translated, into different languages*
- *handle multiple locales at once*
- *be easily modified later to support even more locales*

From the Javadoc, you can also note that the `ResourceBundle` is parent to both `ListResourceBundle` and `PropertyResourceBundle`. In this tutorial we utilize the `PropertyResourceBundle`, which manages resources as text files that use the `.properties` extension and contain locale-specific information in the form of key-value pairs. With new each translation, a new version of the resource bundle is created by appending the locale identifier to the base name using an underscore ('_'). For example, snippets from two of the resource bundles you create in this tutorial look as follows:

messages_en.properties

```
meats=meats
bakery=bakery
```

messages_cs.properties

```
meats=maso
bakery=pečivo
```

In the above example, 'messages' represents the base name, and the locale identifier is the two-letter code which is appended using an underscore. (i.e., 'en' for English, 'cs' for Czech). The two-letter codes are derived from the international [ISO 639](#) standard, which lists codes that represent the names of languages. The ISO 639 standard is adopted by the [W3C Internationalization Activity](#) and is used by all major browsers (these are the codes understood in the `Accept-Language` HTTP header). It is also internalized in the `java.util.Locale` class.

Making Pages Multilingual

Returning to the `AffableBean` application, after continued discussions with the customer you've agreed on the following implementation details:

- The website initially displays based on the preferred language of the user's browser.
- If the browser's preferred language is neither English nor Czech, the site displays text in English.
- The user has the option of changing the language by means of a 'language toggle' in the page header.
- When using the language toggle to change the language, the user remains in the same page view.
- The language toggle should not appear for the confirmation page, as a user will already have selected his or her language prior to checkout.

In order to implement the above points, divide the task into two parts. Start by creating basic bilingual support for page views. Once bilingual support is in place, implement the language toggle that enables users to manually switch languages.



There are three basic steps that you need to follow to incorporate multilingual support into your web pages.

1. Create a resource bundle for each language you plan to support.
2. Register the resource bundle with the application by setting a context parameter in the `web.xml` deployment descriptor.
3. In page views, replace 'hard-coded' text with `<fmt:message>` tags that reference keys in the resource bundles.


The following exercise demonstrates how to integrate English and Czech language support into the AffableBean welcome page by applying the above three steps, and finishes by showing how to test for browser language support using Firefox.

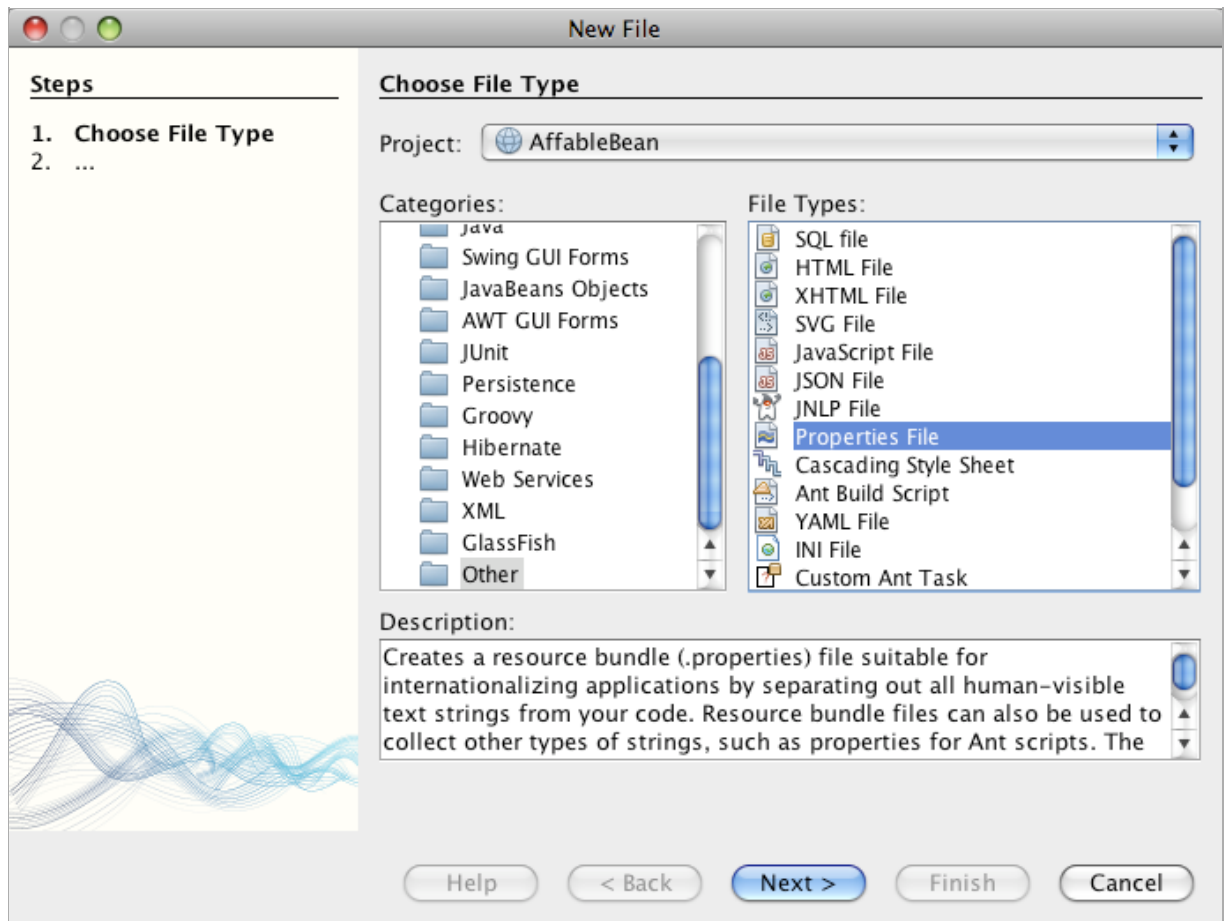
1. [Create Resource Bundles](#)
2. [Register the Resource Bundle with the Application](#)
3. [Replace 'Hard-Coded' Text with <fmt:message> Tags](#)
4. [Test Supported Languages](#)

Create Resource Bundles

1. Open the AffableBean project [snapshot 8](#) in the IDE. Click the Open Project () button and use the wizard to navigate to the location on your computer where you downloaded the project.
2. Click the Run Project () button to run the project and ensure that it is properly configured with your database and application server.

If you receive an error when running the project, revisit the [setup instructions](#), which describe how to prepare the database and establish connectivity between the IDE, GlassFish, and MySQL.

3. Begin by creating a default resource bundle to contain text used in page views. Click the New File () button in the IDE's toolbar. (Alternatively, press Ctrl-N; ⌘-N on Mac.)
4. Under Categories select Other, then under File Types select Properties File.

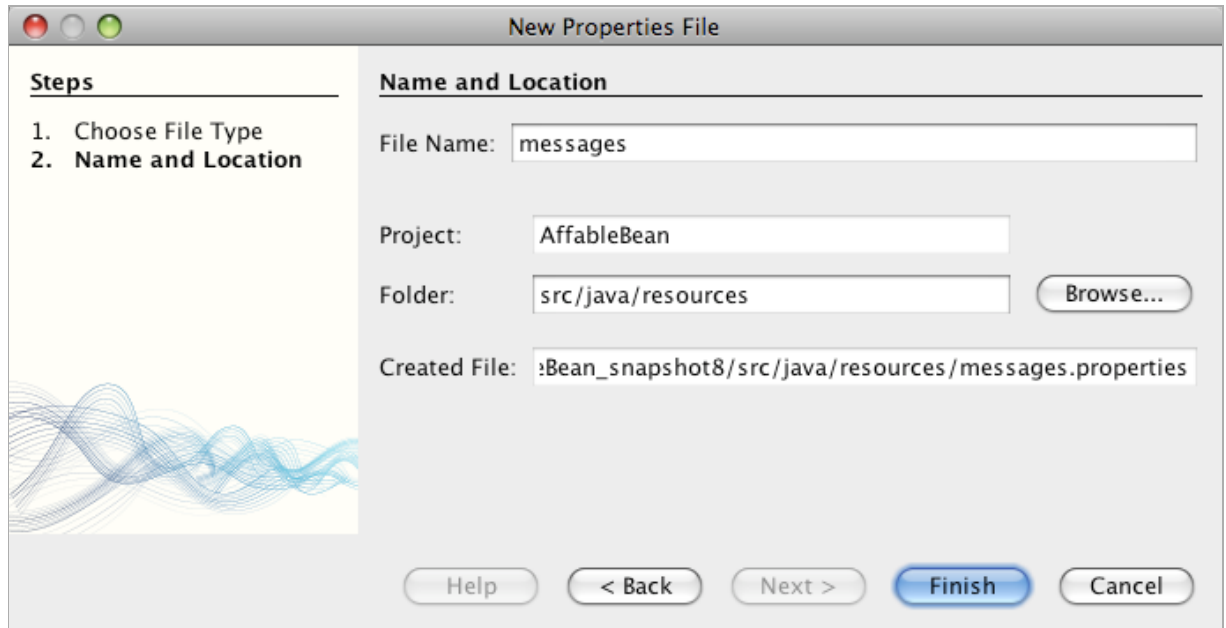


Note that the wizard provides a description for the selected file type:

Creates a resource bundle (.properties) file suitable for internationalizing applications by separating out all human-visible text strings from your code. Resource bundle files can also be used to collect other types of strings, such as properties for Ant scripts. The created resource bundle contains

only one locale, but you can add additional locales from the created file's contextual menu. The bundle can be edited in a text file (property-file format) for a specific locale or in a table that displays information for all locales.

5. Click Next. In the Name and Location step, name the file `messages` and type in `src/java/resources` in the Folder field. This will instruct the wizard to place the resource bundle in a new package named `resources`.



6. Click Finish. The `messages.properties` resource bundle is generated and opens in the editor.

Note that the new `messages.properties` file name does not have a language code appended to it, as was previously described. This is because this file will be used as the *default* resource bundle. The default resource bundle is applied when the Java runtime environment does not find a direct match for the requested locale.

7. Open the project's `index.jsp` file in the editor and note that the following text is currently used:

- **Greeting:** Welcome to the online home of the Affable Bean Green Grocer.
- **Introductory Message:** Enjoy browsing and learning more about our unique home delivery service bringing you fresh organic produce, dairy, meats, breads and other delicious and healthy items to your doorstep.

Also, note that we'll need language-specific names for the four categories that display when `index.jsp` renders in the browser. Since these names are currently taken from the database, we can use them as keys in the resource bundle.

Recall that one of the [implementation details](#) outlined above states that "if the browser's preferred language is neither English nor Czech, the site displays text in English." Therefore, the values that we apply to the `messages.properties` file will be in English.

8. In the `messages.properties` file, begin adding key-value pairs for the text used in the welcome page. Add the following content.

```
# welcome page
greeting=Welcome to the online home of the Affable Bean Green Grocer.
introText=Our unique home delivery service brings you fresh organic produce, dairy,
meats, breads and other delicious and healthy items direct to your doorstep.

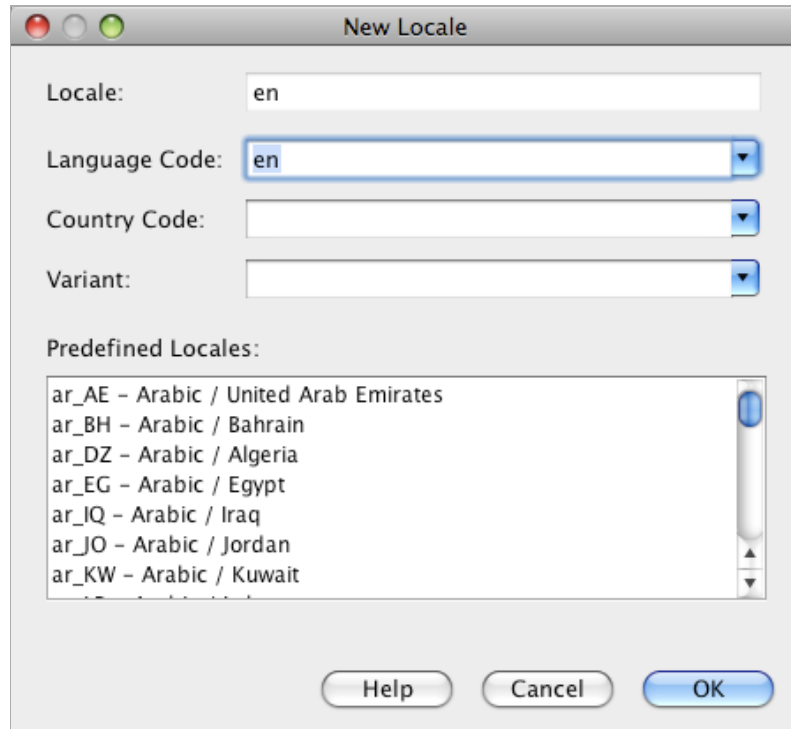
# categories
dairy=dairy
meats=meats
bakery=bakery
```

```
fruit\ &\ veg=fruit & veg
```

Comments are added using a number sign ('#'). Also, because the `fruit & veg` category name contains spaces, it is necessary to escape the space characters using a backslash ('\') in order to apply the name as a resource bundle key.

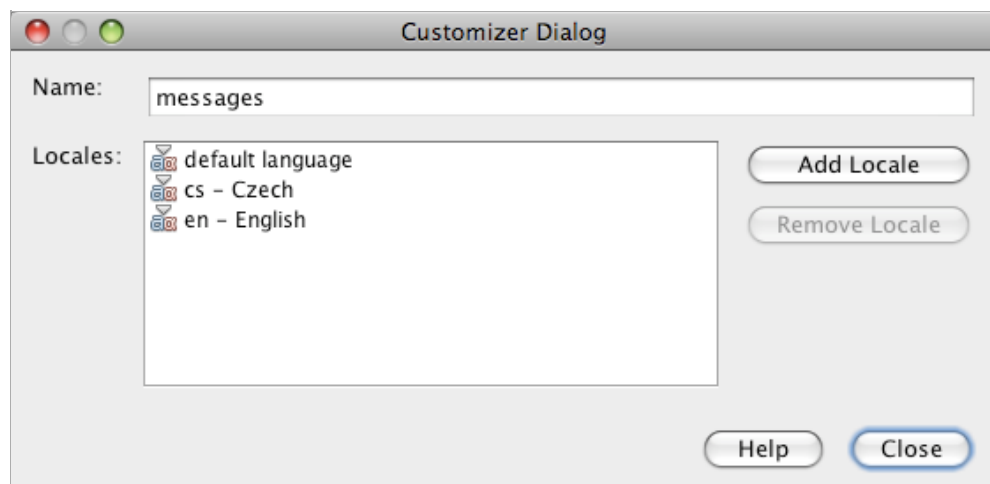
We are now finished with the default resource bundle for the application's welcome page. Let's continue by creating resource bundles for the customer-specified languages.

9. In the Projects window, expand the Source Packages node, then right-click the `resources > messages.properties` file node and choose Customize. The Customizer dialog opens.
10. In the Customizer dialog, click the Add Locale button. In the New Locale dialog that displays, enter 'en' in the Language Code combo box, then click OK.

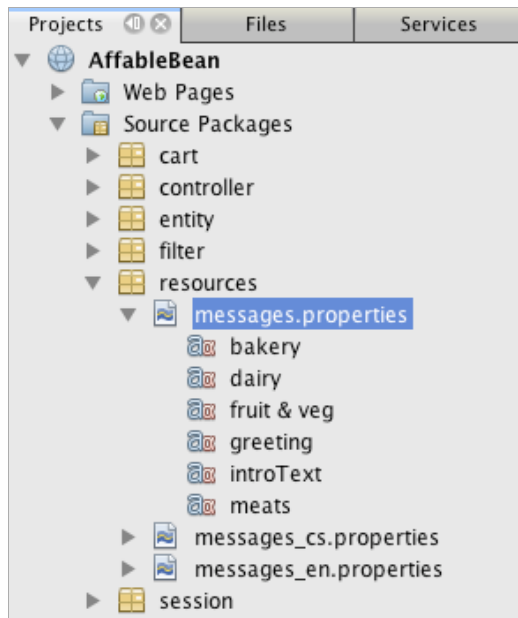


A *locale* can be defined by both a language and a geographic region. The optional country code which can be used to specify the region can be applied to define formatting for dates, time, numbers, and currency. For more information, see the technical article, [Understanding Locale in the Java Platform](#).

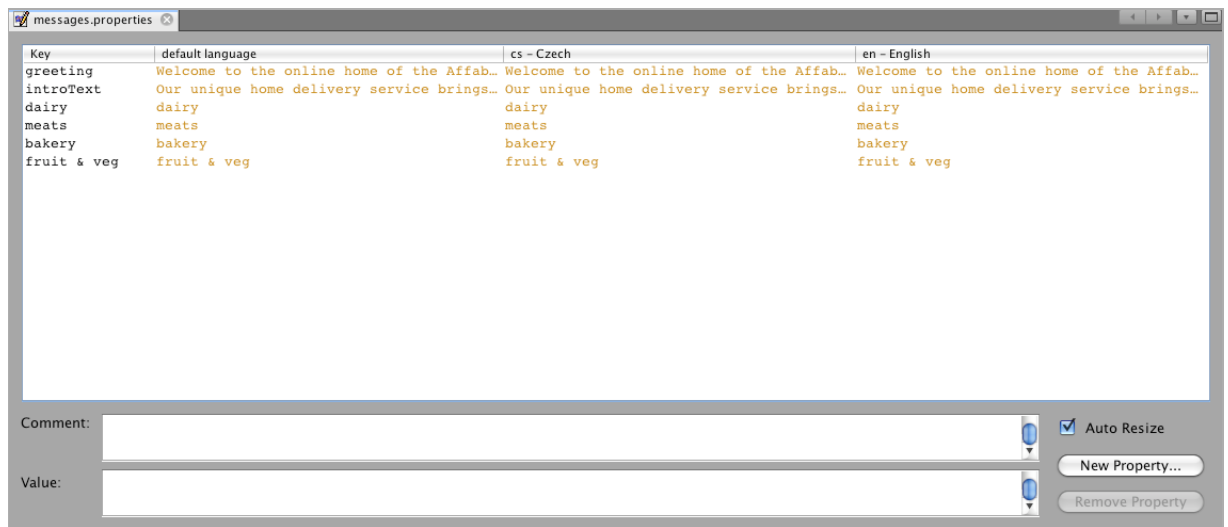
11. Click the Add Locale button again, then enter 'cs' in the Language Code combo box and click OK. The Customizer dialog displays as follows.



12. Click Close. In the Projects window, note that your resource bundles look as follows. You can expand a resource bundle to view the keys it contains.



13. Right-click any of the three resource bundles and choose Open. The Properties editor opens, enabling you to view and edit key-value pairs for all locales in a table layout.



Press Shift-Esc to maximize the window in the IDE.

Note that when you add a new locale using the Customizer dialog, as you did for English and Czech in the previous steps, the keys and values of the default resource bundle are copied to the new locale.

14. Modify the values for the Czech resource bundle. You can do this by *either* clicking into the table cells for each row and typing your entries directly *or* selecting the cell you want to edit and typing into the **Value** field located at the bottom of the Properties editor.
- **greeting:** Vítejte v našem domácím on-line obchodě Affable Bean Green Grocer.
 - **introText:** Naše jedinečná dodávková služba Vám zajistí dopravu čerstvých organických produktů, mléčných výrobků, uzenin, pečiva a dalších delikates a zdravých výroků až ke dveřím.
 - **dairy:** mléčné výrobky
 - **meats:** maso
 - **bakery:** pečivo

- **fruit & veg:** ovoce a zeleniny

You can also add a comment to each key-value pair. Any text you enter into the **Comment** field in the Properties editor is added to the resource bundle text file above the key-value pair as a comment (i.e., following a '#' sign).

15. Double-click the `messages_cs.properties` file node in the Projects window. Note that the text file has been updated according to your changes in the Properties editor.

```
# welcome page
greeting=Vítejte v našem domácím on-line obchodě Affable Bean Green Grocer.
introText=Naše jedinečná dodávková služba Vám zajistí dopravu čerstvých organických
produktů, mléčných výrobků, uzenin, pečiva a dalších delikates a zdravých výroků až
ke dveřím.

# categories
dairy=mléčné výrobky
meats=maso
bakery=pečivo
fruit\ &\ veg=ovoce a zeleniny
```

We now have the following resource bundles defined:

- default (English)
- Czech
- English

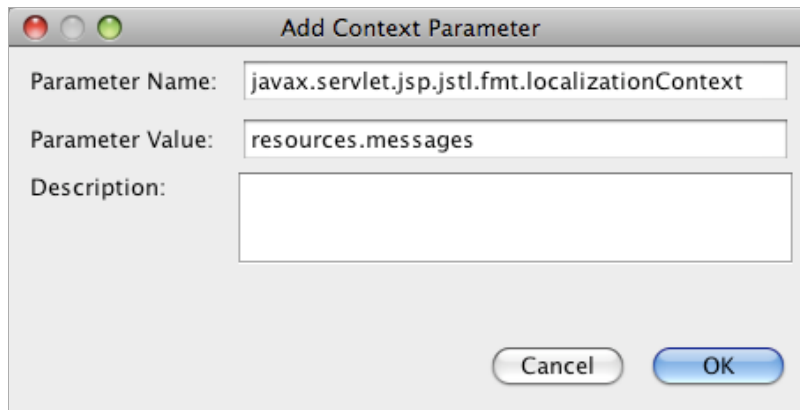
You might assume that if the default bundle is in English, then there is no need to create a resource bundle explicitly for English. However, consider the following scenario: a client browser's list of preferred languages includes both Czech and English, with English taking precedence over Czech. If the application doesn't provide a resource bundle for English but does for Czech, pages sent to that browser will be in Czech (since a Czech bundle was defined). This is clearly not the desired behavior for that browser.

Register the Resource Bundle with the Application

The purpose of this step is to inform JSTL's `format` (i.e., `fmt`) tag library where it can locate any resource bundles existing in the application. You accomplish this by instructing the application to create a `LocalizationContext` using the existing resource bundles. This can be done by setting a context parameter in the application's `web.xml` deployment descriptor.

The topic of setting context parameters is also covered in [Connecting the Application to the Database](#).

1. In the Projects window, expand the Configuration Files node, then double-click `web.xml` to open it in the editor.
2. Under the deployment descriptor's General tab, expand the Context Parameters category.
3. Click the Add button, then in the Add Context Parameter dialog enter the following values.
 - **Parameter Name:** `javax.servlet.jsp.jstl.fmt.localizationContext`
 - **Parameter Value:** `resources.messages`



The `LocalizationContext` class belongs to the `javax.servlet.jsp.jstl.fmt` package. You can verify this by viewing the [JSTL 1.1 API Reference](#) online.

4. Click OK. The new context parameter is added to the table of existing context parameters under the General tab.
5. Click the deployment descriptor's XML tab. Note that the following entry has been added to the file:

```
<context-param>
  <param-name>javax.servlet.jsp.jstl.fmt.localizationContext</param-name>
  <param-value>resources.messages</param-value>
</context-param>
```

Replace Hard-Coded Text with `<fmt:message>` Tags

In order to apply the localized text of resource bundles to your web pages, you reference the keys from the key-value pairs you created. You can reference the keys using JSTL's `<fmt:message>` tags.

1. Open the project's `index.jsp` page in the editor. (If already opened, press Ctrl-Tab to switch to the file.)
2. Delete instances of hard-coded text that display in the page's left column, and in their place enter `<fmt:message>` tags using the `key` attribute to specify the resource bundle key. The page's left column will look as follows.

```
<div id="indexLeftColumn">
  <div id="welcomeText">
    <p style="font-size: larger"><fmt:message key='greeting' /></p>

    <p><fmt:message key='introText' /></p>
  </div>
</div>
```

3. Add `<fmt:message>` tags for the four category names, but use the `${category.name}` expression as the value for the `key` attribute. Since the category name is also used as the value for the `` tag's `alt` attribute, follow the same procedure. The page's right column will look as follows.

```
<div id="indexRightColumn">
  <c:forEach var="category" items="${categories}">
    <div class="categoryBox">
      <a href="<c:url value='category?${category.id}' />">
        <span class="categoryLabel"></span>
        <span class="categoryLabelText"><fmt:message key='${category.name}' />
      </span>

      "
        class="categoryImage">
```



```

        </a>
    </div>
</c:forEach>
</div>

```

4. Finally, ensure that you have the `fmt` tag library declared in the web page. Enter the following at the top of the file:

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

Note: Here you add the tag library declaration to the top of the `index.jsp` file. However, when you begin using `<fmt>` tags elsewhere in the project, it may make more sense to remove the tag library declaration from individual page views, and add it to the header (`header.jspf`) file. This practice is adopted in [snapshot 9](#) (and later snapshots).

You've now completed the tasks necessary for providing bilingual support for the application's welcome page. The following step demonstrates how to test the language support in your browser.

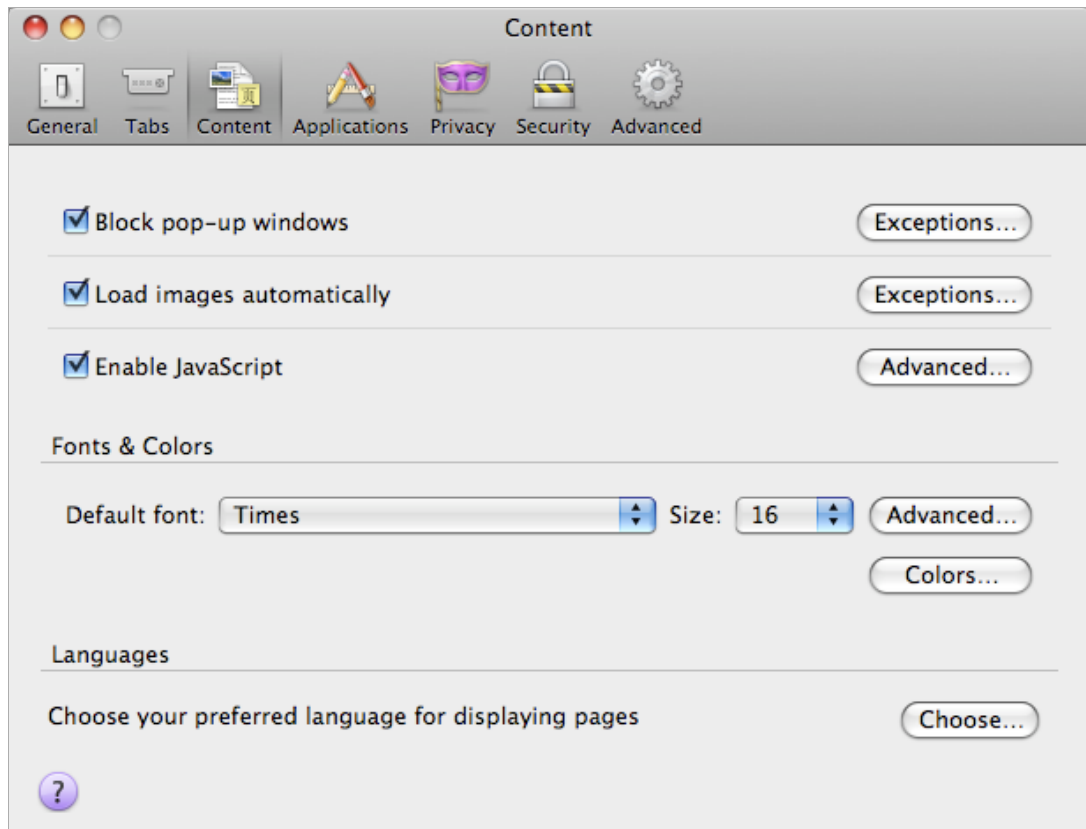
Test Supported Languages

You could theoretically test for the following scenarios involving the application's supported languages, as well as an unsupported language (e.g., Korean):

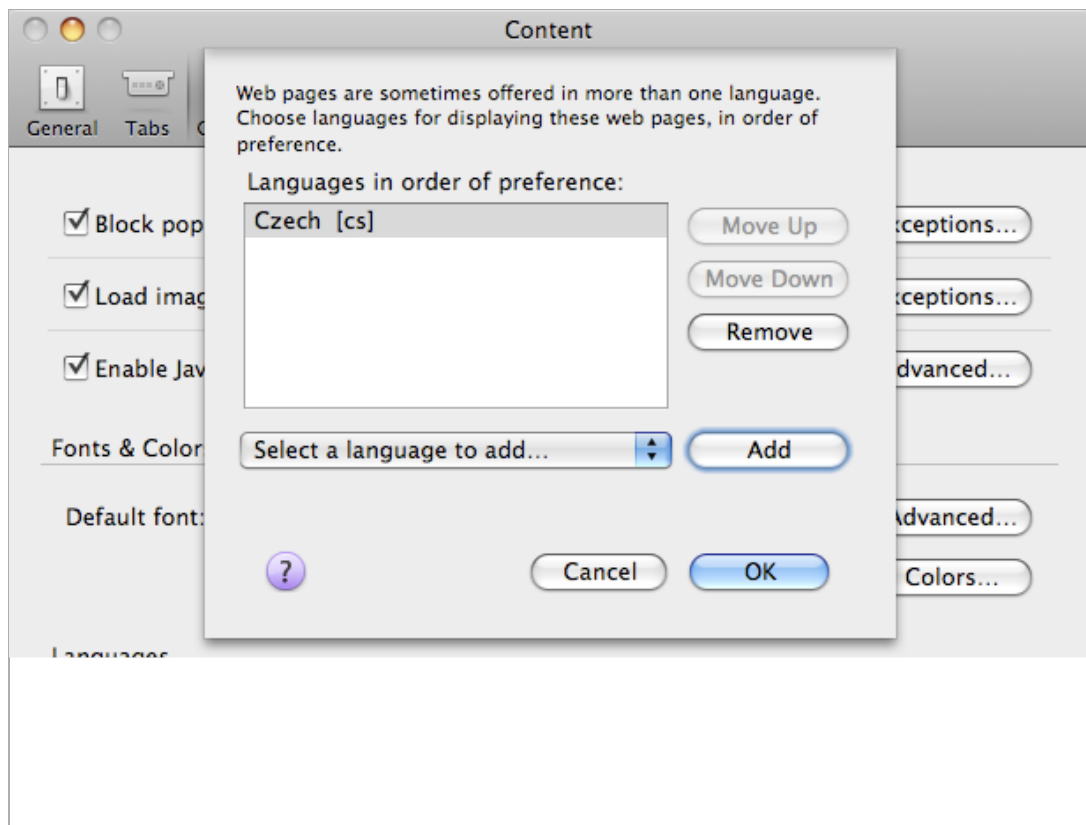
Use-case	Outcome
1. Browser has no preferred language	English displays
2. Browser prefers only English	English displays
3. Browser prefers only Czech	Czech displays
4. Browser prefers only Korean	English displays
5. Browser prefers Korean and English; Korean takes precedence	English displays
6. Browser prefers Korean and English; English takes precedence	English displays
7. Browser prefers Korean and Czech; Korean takes precedence	Czech displays
8. Browser prefers Korean and Czech; Czech takes precedence	Czech displays
9. Browser prefers English and Czech; English takes precedence	English displays
10. Browser prefers English and Czech; Czech takes precedence	Czech displays
11. Browser prefers, in the following order, English, Czech, Korean	English displays
12. Browser prefers, in the following order, English, Korean, Czech	English displays
13. Browser prefers, in the following order, Czech, English, Korean	Czech displays
14. Browser prefers, in the following order, Czech, Korean, English	Czech displays
15. Browser prefers, in the following order, Korean, English, Czech	English displays
16. Browser prefers, in the following order, Korean, Czech, English	Czech displays


Rather than stepping through all 16 scenarios, we'll demonstrate how to examine scenario 3 above, in which the browser's preferred language is Czech, using the Firefox browser.

1. In Firefox, choose Tools > Options (Firefox > Preferences on Mac). In the window that displays, click the Content tab.



2. Under the Languages heading, click Choose.
3. Select any language that is currently listed in the provided text area, then click Remove. (You should remember your language list and reinstate languages after completing this tutorial.)
4. Click the 'Select Language to Add' drop-down and select Czech [cs]. Then click the Add button. The Czech language is added to the text area.



5. Click OK, then press Esc to close Firefox' Options window.
6. Run the project (). When the welcome page opens in your browser, note that text is displayed in Czech.



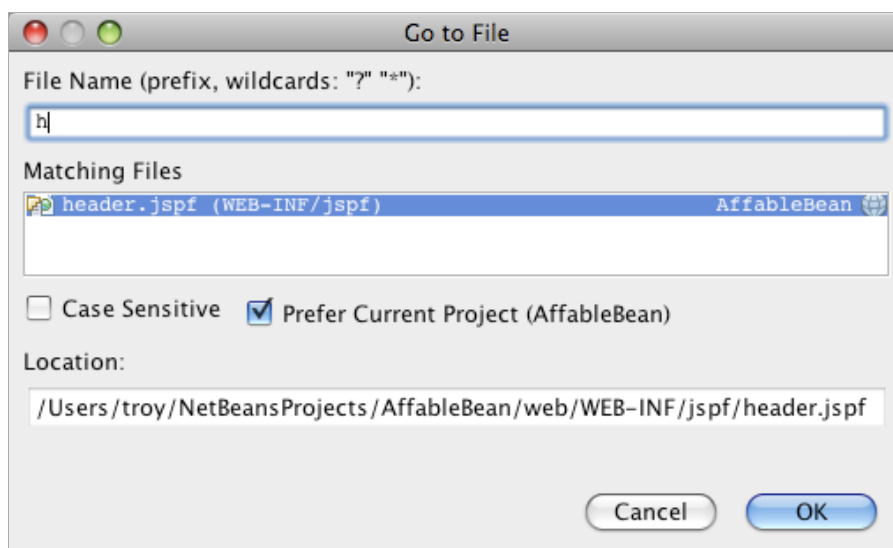
Implementing a Language Toggle

Now that basic Czech-English language support is in place, continue by implementing the language toggle in the application's page views. We can divide this task into three parts:

- [Create Toggle Display and Synchronize with the Browser's Preferred Language](#)
- [Implement Functionality to Handle a Request from the Language Toggle](#)
- [Enable the Application to Keep Track of the Originating Page View](#)

Create Toggle Display and Synchronize with the Browser's Preferred Language

1. Use the Go to File dialog to open the header JSP fragment in the editor. Press Alt-Shift-O (Ctrl-Shift-O on Mac), then type 'h' in the dialog and click OK.



2. In the header.jspf file, locate the first <div class="headerWidget"> tag (line 56), and replace the [

language toggle] placeholder text with the following HTML markup.


```
<div class="headerWidget">

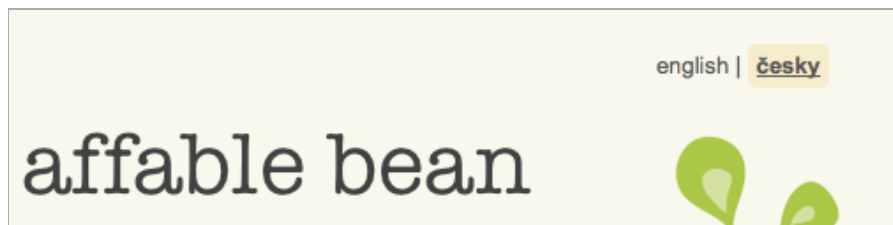
    <!-- language selection widget -->
    english | <div class="bubble"><a href="chooseLanguage?language=cs">česky</a>
</div>
</div>
```

This markup implements the language toggle's appearance when English is the displayed language. In other words, the toggle provides a link allowing the user to select the Czech (i.e., 'česky') option. The link is used to send a request for `chooseLanguage`, and creates a query string (`?language=cs`) that specifies the requested language code.

Note: Recall that in Unit 5, [Preparing the Page Views and Controller Servlet](#), you set the `ControllerServlet` to handle the `/chooseLanguage` URL pattern.

Snapshot 8 includes the [jQuery](#) JavaScript library and takes advantage of various UI effects to enhance the appearance and behavior of the website. Aside from a [jQuery plugin for client-side validation](#) (discussed in the [previous tutorial unit](#)), the snapshot implements an easing effect for category headings in the welcome page, as well as for category buttons in the category page. Configuration is included in `header.jspf` of the project snapshot. Rounded corners are implemented using CSS3's [border-radius](#) property (applied in `affablebean.css`).

3. Run the project () to see what the toggle looks like in the browser.



Currently, the language toggle appears as in the above image regardless of what language the page displays in. In the next step, you integrate JSTL logic into the toggle so that it renders according to the language displayed on the page.

4. Modify the toggle implementation as follows.

```
<div class="headerWidget">

    <!-- language selection widget -->
    <c:choose>
        <c:when test="${pageContext.request.locale.language ne 'cs'}">
            english
        </c:when>
        <c:otherwise>
            <c:url var="url" value="chooseLanguage">
                <c:param name="language" value="en"/>
            </c:url>
            <div class="bubble"><a href="${url}">english</a></div>
        </c:otherwise>
    </c:choose> |

    <c:choose>
        <c:when test="${pageContext.request.locale.language eq 'cs'}">
            česky
        </c:when>
        <c:otherwise>
            <c:url var="url" value="chooseLanguage">
```

```

        <c:param name="language" value="cs"/>
    </c:url>
    <div class="bubble"><a href="${url}">česky</a></div>
</c:otherwise>
</c:choose>
</div>

```

In the above implementation, you rely on conditional tags from JSTL's `core` tag library to display the left and right portions of the toggle according to the language used by the request locale. What is the "language used by the request locale"? When a request is made, the browser passes a list of preferred locales in the `Accept-Language` HTTP header. The Java runtime environment on the server reads the list and determines the best match based on the locales defined by the application's resource bundles. This match is then recorded in the `ServletRequest` object, and can be accessed using the `getLocale` method. For example, you could access the preferred locale from a servlet with the following statement.

```
request.getLocale();
```

You can use the IDE's HTTP Monitor (Window > Debugging > HTTP Server Monitor) to examine HTTP headers for client requests. In order to use the HTTP Monitor, you need to first activate it for the server you are using. Unit 8, [Managing Sessions](#) provides a demonstration under the sub-section, [Examining Client-Server Communication with the HTTP Monitor](#).

To determine the language of the preferred locale, you use the `Locale` class' `getLanguage` method. Again, from a servlet you could access the language of the client request's preferred locale with the following.

```
request.getLocale().getLanguage();
```

Returning to the code you just added to the `header.jspf` fragment, you utilize the `pageContext.request` implicit object to access the `ServletRequest` for the given client request. Using dot notation, you then proceed to call the same methods as you would from a servlet. In the above example, accessing the "language used by the request locale" is as simple as:

```
${pageContext.request.locale.language}
```

Note: The above implementation uses `<c:url>` tags to set up the toggle link. This is done in order to properly encode the request URL in the event that URL rewriting is used as a means for session tracking. Unit 8, [Managing Sessions](#) provides a brief explanation of how the `<c:url>` tags can be used.

5. Add a basic language test to the `header.jspf` file. This will enable us to check whether the toggle is properly rendering according to the client request's preferred language. Enter the following after the page's `<body>` tag.


```

<body>

    <!-- Language test -->
    <p style="text-align: left;"><strong>tests:</strong>
        <br>
        <code>\${pageContext.request.locale.language}</code>:
    <code>\${pageContext.request.locale.language}</code>
    </p>

    <div id="main">

```

6. Ensure that you have set Czech as your browser's preferred language. (If you are following this tutorial unit sequentially, you've already done this. If not, refer to the steps outlined above in [Test Supported Languages](#).)
7. Run the project () and examine the application welcome page in the browser.

tests:

```
${pageContext.request.locale.language}: CS
```



If your browser's preferred language is set to Czech, you can note the following:

- The test that we introduced in the previous step indicates that 'cs' is the preferred language.
- Czech text is displayed in the page.
- The language toggle provides a link enabling the user to select English.

Implement Functionality to Handle a Request from the Language Toggle

Now that the toggle is in place and it appears according to the language displayed in the page, let's continue by adding code to the `ControllerServlet` that handles the request sent when a user clicks the link in the language toggle.

As indicated in the current language toggle implementation from [step 4](#) above, the requested URL with query string looks as follows:

- **English:** `chooseLanguage?language=en`
- **Czech:** `chooseLanguage?language=cs`

Our goal is to register the language choice, and then display both the page view and language toggle based on the chosen language. We can accomplish this by extracting the `language` parameter from the query string and creating a session-scoped language attribute that remembers the language selected by the user. Then we'll return to the `header.jspf` fragment and apply the `<fmt:setLocale>` tag to set the page language based on the user's choice. With the `<fmt:setLocale>` tag we can manually switch the language used in the page display. We'll also modify the language toggle so that if the language attribute has been set, the toggle's appearance is determined according to the language attribute's value.

1. Open the `ControllerServlet` in the editor. Use the Go To File dialog - press Alt-Shift-O (Ctrl-Shift-O on Mac), then type 'controller' and click OK. In the opened file, locate the portion of the `doGet` method that handles the

chooseLanguage request (line 126).

2. Delete the `// TODO: Implement language request` comment and enter code to extract the language parameter from the request query string.

```
// if user switches language
} else if (userPath.equals("/chooseLanguage")) {

    // get language choice
    String language = request.getParameter("language");
}
```

3. Place the language parameter in the request scope. Add the following.

```
// if user switches language
} else if (userPath.equals("/chooseLanguage")) {

    // get language choice
    String language = request.getParameter("language");

    // place in request scope
    request.setAttribute("language", language);
}
```

4. As a temporary measure, have the application forward the response to the `index.jsp` welcome page when the language toggle link is clicked. Add the following code.

```
// if user switches language
} else if (userPath.equals("/chooseLanguage")) {

    // get language choice
    String language = request.getParameter("language");

    // place in request scope
    request.setAttribute("language", language);

    // forward request to welcome page
    try {
        request.getRequestDispatcher("/index.jsp").forward(request, response);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return;
}
```

Naturally, forwarding the user to the welcome page regardless of what page he or she is on is not an ideal way to handle the language toggle's behavior. We'll return to this matter in the next sub-section, [Enable the Application to Keep Track of the Originating Page View](#). For the meantime however, this will allow us to examine the results of the current language toggle implementation when running the project.

5. Switch to the `header.jspf` fragment (If the file is already opened in the editor, press Ctrl-Tab and choose the file.) and apply the `<fmt:setLocale>` tag to set the page language based on the new language variable. Add the following.

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

```

<%-- Set language based on user's choice --%>
<c:if test="${!empty language}">
    <fmt:setLocale value="${language}" scope="session" />
</c:if>

<%@page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

```

Since the language variable is only created when the user clicks the link in the language toggle, you perform a test using `<c:if>` tags to determine whether the variable exists before attempting to set the language. When applying the `<fmt:setLocale>` tag, you set its scope to `session` as you want the user-selected language to take precedence for the remainder of his or her session on the website. Also, since this is the first time the `fmt` library is used in the header, you declare the tag library.

You can read the EL expression `${!empty language}` as, "False if the language variable is null or an empty string." See the [Java EE 5 Tutorial: Examples of EL Expressions](#) for other available examples.

6. Modify the language toggle implementation so that if a value has been set by the `<fmt:setLocale>` tag, the toggle displays according to the language specified by that value. (You can determine this value using the `sessionScope['javax.servlet.jsp.jstl.fmt.locale.session']` expression.)

Enclose the current implementation within `<c:choose>` tags, and create logic similar to the current implementation in the event that the locale has been manually set. (Changes are displayed in **bold**.)

```

<div class="headerWidget">

    <%-- language selection widget --%>
    <c:choose>
        <%-- When user hasn't explicitly set language,
             render toggle according to browser's preferred locale --%>
        <c:when test="${empty
sessionScope['javax.servlet.jsp.jstl.fmt.locale.session']}">
            <c:choose>
                <c:when test="${pageContext.request.locale.language ne 'cs'}">
                    english
                </c:when>
                <c:otherwise>
                    <c:url var="url" value="chooseLanguage">
                        <c:param name="language" value="en"/>
                    </c:url>
                    <div class="bubble"><a href="${url}">english</a></div>
                </c:otherwise>
            </c:choose> |

        <c:choose>
            <c:when test="${pageContext.request.locale.language eq 'cs'}">
                český
            </c:when>
            <c:otherwise>
                <c:url var="url" value="chooseLanguage">
                    <c:param name="language" value="cs"/>
                </c:url>
                <div class="bubble"><a href="${url}">česky</a></div>
            </c:otherwise>
        </c:choose>
    </c:when>

```



```

<!-- Otherwise, render widget according to the set locale -->
<c:otherwise>
  <c:choose>
    <c:when test="\${sessionScope['javax.servlet.jsp.jstl.fmt.locale.session']} ne
'cs' }">
      english
    </c:when>
    <c:otherwise>
      <c:url var="url" value="chooseLanguage">
        <c:param name="language" value="en"/>
      </c:url>
      <div class="bubble"><a href="\${url}">english</a></div>
    </c:otherwise>
  </c:choose> |

  <c:choose>
    <c:when test="\${sessionScope['javax.servlet.jsp.jstl.fmt.locale.session']} eq
'cs' }">
      český
    </c:when>
    <c:otherwise>
      <c:url var="url" value="chooseLanguage">
        <c:param name="language" value="cs"/>
      </c:url>
      <div class="bubble"><a href="\${url}">český</a></div>
    </c:otherwise>
  </c:choose>
</c:otherwise>
</c:choose>

</div>



```

7. Before examining the project in a browser, add another test that displays the value set by the `<fmt:setLocale>` tag. Add the following code beneath the test you created earlier.

```

<p style="text-align: left;"><strong>tests:</strong>
  <br>
  <code>\${pageContext.request.locale.language}</code>:
  ${pageContext.request.locale.language}
  <br>
  <code>\${sessionScope['javax.servlet.jsp.jstl.fmt.locale.session']}</code>:
  ${sessionScope['javax.servlet.jsp.jstl.fmt.locale.session']}
</p>

```

`javax.servlet.jsp.jstl.fmt.locale.session` is the *string literal* key for the `Locale` set by the `<fmt:setLocale>` tag. You can verify this by clicking in the editor's left margin to set a breakpoint () on the new test, then running the debugger () on the project. When you click the toggle link to change languages in the browser, examine the Variables window (Alt-Shift-1; Ctrl-Shift-1 on Mac) when the debugger suspends on the breakpoint.


Variables		
Name	Type	Value
▼ Implicit Objects		
▶ request	MonitorRequestWrapper	... #8837
▶ response	MonitorResponseWrapper	... #8839
▶ pageContext	PageContextImpl	... #8840
▼ session	StandardSessionFacade	... #8485
▼ session	StandardSession	... #8631
▼ attributes	ConcurrentHashMap	... "size = 3"

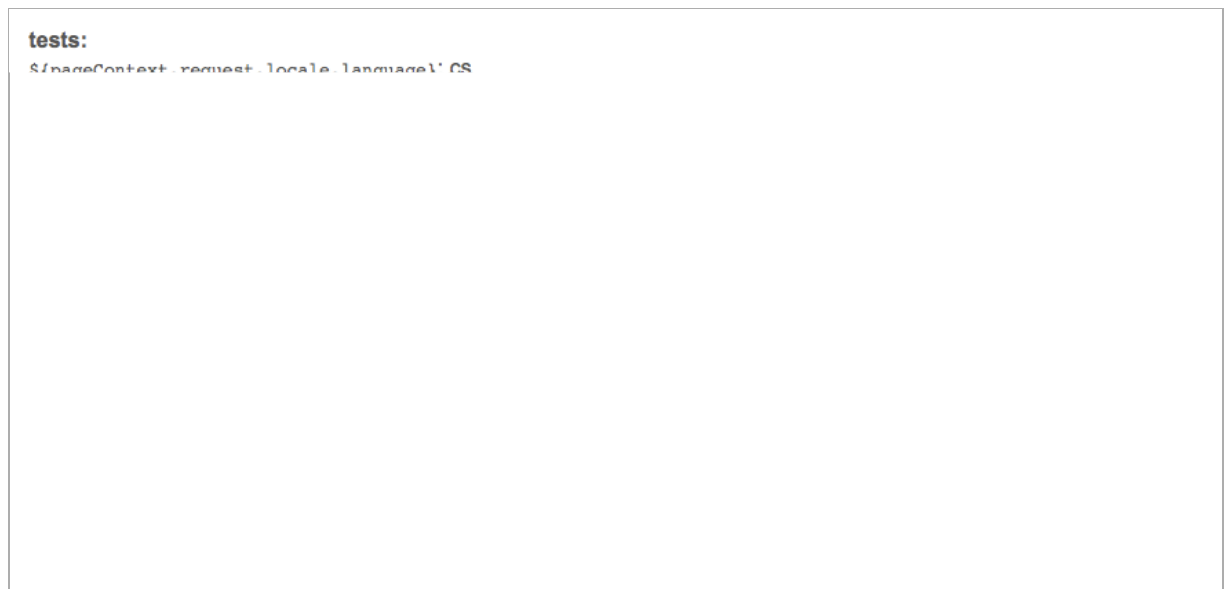
EL expressions presented in this tutorial primarily use dot (.) notation. The format depicted in the expression above is known as *bracket* ([]) notation whereby you enter the string literal key within quotes in order to extract the object's value:

```
${sessionScope['javax.servlet.jsp.jstl.fmt.locale.session']}
```

Numerous EL resolver classes exist for the purpose of resolving expressions. For example, when the above expression is encountered at runtime, the `ImplicitObjectResolver` first returns a `Map` that maps session-scoped attribute names to their values. (In the above image of the Variables window, you can verify that session attributes are maintained in a `ConcurrentHashMap`.) In order to resolve the remainder of the expression, the `MapELResolver` is used to get the value of the key named `'javax.servlet.jsp.jstl.fmt.locale.session'`.

For more information, refer to the Java EE 5 Tutorial: [Unified Expression Language: Resolving Expressions](#).

8. Run the project () and examine the application welcome page in the browser.



In the above image, the server identifies Czech (cs) as the browser's preferred language from the Accept-Language HTTP header. This is indicated from the first test. The page displays in Czech, and the language toggle enables the user to choose English. The second test remains blank as the `<fmt:setLocale>` tag has not yet been called.

9. Click the toggle link for English.

When clicking the toggle link, the default Czech language is overridden by means of the `<fmt:setLocale>` tag implemented in the `header.jspf` file. Although the browser's preferred language remains Czech, you see that the page now displays according to the new language made available by the language toggle.

10. Click the toggle link for Czech.

Changing the language back to the browser's preferred language works as expected, however note that the deciding factor is no longer the language detected from the `Accept-Language` HTTP header, but is the language specified from the `<fmt:setLocale>` tag.

11. Before continuing, remove the tests you added to the `header.jspf` file. (Deleted code in ~~strike-through~~ text.)

```
<body>

<%-- Language tests --%>
<p style="text-align: left;"><strong>tests:</strong>
<br>
<code>\${pageContext.request.locale.language}</code>:-
\${pageContext.request.locale.language}
<br>
<code>\${sessionScope['javax.servlet.jsp.jstl.fmt.locale.session']}</code>:-
\${sessionScope['javax.servlet.jsp.jstl.fmt.locale.session']}
</p>



<div id="main">
```

Enable the Application to Keep Track of the Originating Page View

One of the [implementation details](#) which you have agreed on with the Affable Bean staff is that when the language toggle is used to change the language, the user remains in the same page view. In our current implementation, the welcome page is returned whenever the language toggle is clicked. A more user-friendly approach would be to provide the application with a means of tracking the request page view, and forwarding the request to that page view when the language toggle link is clicked.

We can accomplish this by setting a session-scoped `view` attribute within each of the page views, then referencing this attribute in the `ControllerServlet` in order to determine where to forward the request. There are however several caveats to consider when dealing with the language toggle in the confirmation page. These are discussed and dealt with in steps 7-11 below.

Begin this exercise with [snapshot 9](#) of the `AffableBean` project. This snapshot includes completed English and Czech resource bundles for all page views, all page views have been modified to use text from the resource bundles, and the language toggle is presented in a state corresponding to this point in the tutorial.

1. Open [snapshot 9](#) in the IDE. Click the Open Project () button and use the wizard to navigate to the location on your computer where you downloaded the project.
2. Click the Run Project () button to run the project. When navigating through the site, note that when you click the language toggle from any of the page views, you are returned to the application's welcome page.

If you receive an error when running the project, revisit the [setup instructions](#), which describe how to prepare the database and establish connectivity between the IDE, GlassFish, and MySQL.

3. Use `<c:set>` tags to set a session-scoped `view` attribute for each of the page views. Open each of the page views in the editor and add the following code to the top of each file.

index.jsp

```
<%-- Set session-scoped variable to track the view user is coming from.
```

```

        This is used by the language mechanism in the Controller so that
        users view the same page when switching between English and Czech. --%>
<c:set var='view' value='/index' scope='session' />

```

category.jsp

```

<%-- Set session-scoped variable to track the view user is coming from.
      This is used by the language mechanism in the Controller so that
      users view the same page when switching between English and Czech. --%>
<c:set var='view' value='/category' scope='session' />

```

cart.jsp

```

<%-- Set session-scoped variable to track the view user is coming from.
      This is used by the language mechanism in the Controller so that
      users view the same page when switching between English and Czech. --%>
<c:set var='view' value='/cart' scope='session' />

```

checkout.jsp

```

<%-- Set session-scoped variable to track the view user is coming from.
      This is used by the language mechanism in the Controller so that
      users view the same page when switching between English and Czech. --%>
<c:set var='view' value='/checkout' scope='session' />

```

Based on customer-agreed [implementation details](#), we do not need to provide a means of switching languages on the confirmation page view. From a usability perspective, a user will have already selected his or her preferred language prior to checkout. From an implementation perspective, recall that we destroy the user session upon a successfully completed order. (Refer back to the final paragraph in [Managing Sessions](#), which describes how to apply the `invalidate` method to explicitly terminate a user session.) If the Affable Bean staff were to insist on allowing customers to view their orders bilingually, you would need to consider the following scenarios, dependent on whether you destroy the user session upon displaying the confirmation page:

1. **Session destroyed:** Would be necessary to take extra measures to ensure that a `chooseLanguage` request from the confirmation page refers to the appropriate order, and can display customer-sensitive details in a secure fashion.
 2. **Session maintained:** Would risk enabling users to mistakenly place double orders on their shopping cart. Also, by not terminating user sessions when they are no longer needed, an unnecessary load may be placed on the server.
4. Open the `ControllerServlet` in the editor. (If already opened, press `Ctrl-Tab` and choose the file.) In the opened file, locate the portion of the `doGet` method that handles the `chooseLanguage` request (line 126).

Note that currently `chooseLanguage` requests are forwarded to the `index.jsp` welcome page.

```

// if user switches language
} else if (userPath.equals("/chooseLanguage")) {

    // get language choice
    String language = request.getParameter("language");

    // place in session scope
    session.setAttribute("language", language);

    // forward request to welcome page

```

```

        try {
            request.getRequestDispatcher("/index.jsp").forward(request, response);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return;
    }
}

```

5. Use the view session attribute to forward the request back to the originating page view. Make the following changes (in **bold**).

```

// if user switches language
} else if (userPath.equals("/chooseLanguage")) {

    // get language choice
    String language = request.getParameter("language");

    // place in request scope
    request.setAttribute("language", language);

    String userView = (String) session.getAttribute("view");

    if ((userView != null) &&
        (!userView.equals("/index"))) {           // index.jsp exists outside 'view' folder
                                                    // so must be forwarded separately

        userPath = userView;
    } else {

        // if previous view is index or cannot be determined, send user to welcome
page
        try {
            request.getRequestDispatcher("/index.jsp").forward(request, response);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return;
    }
}

```

In the above implementation, you extract the value of the view attribute and, provided that the view:

- can be identified (i.e., the value is not null),
- does not originate from the welcome page (index.jsp does not reside in the same location as other page views, and therefore cannot be resolved using the doGet method's way of forwarding requests)


...you set it to the doGet method's userPath variable, and forward the request using the method's existing RequestDispatcher:

```

// use RequestDispatcher to forward request internally
String url = "/WEB-INF/view" + userPath + ".jsp";

try {
    request.getRequestDispatcher(url).forward(request, response);
} catch (Exception ex) {
    ex.printStackTrace();
}

```

6. Run the project () to test it in the browser. When you navigate to the category, cart or checkout pages, switch languages using the language toggle. When you do so, you now remain within the same page view.
7. In the browser, complete an order so that the application forwards you to the confirmation page. When you click the language toggle from the confirmation page, note that you are sent back to the website's welcome page.

Implementation-wise, you may consider this to be sufficient. However, the Affable Bean staff have explicitly asked you to remove the language toggle from this page view. One way to accomplish this is to perform a test to determine whether the request *servlet path* contains `'/confirmation'`.

Switch to the `header.jspf` file in the editor and surround the language toggle with the following test. You can use JSTL's functions (i.e., `fn`) library to perform string operations.

```
<div class="headerWidget">

    <!-- If servlet path contains '/confirmation', do not display language toggle -->
    <c:if test="${!fn:contains(pageContext.request.servletPath, '/confirmation')}">

        <!-- language selection widget -->
        <c:choose>

            ...
        </c:choose>

    </c:if>
</div>
```

Examine the above code snippet and note the following points:

- The servlet path can be accessed from the `HttpServletRequest` using the `getServletPath` method. Because we use a `RequestDispatcher` to forward the request to the confirmation page (`ControllerServlet`, line 158), the servlet path becomes:

```
/WEB-INF/view/confirmation.jsp
```

- Using the `pageContext.request.servletPath` EL expression is comparable to calling `request.getServletPath()` from a servlet.
- The `fn:contains()` function allows you to test if an input string contains the specified substring.
- The `fn` tag library has already been declared for you at the top of in the `header.jspf` file in snapshot 9:

```
<%@taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

8. Run the project again and step through to the confirmation page. Note that the page no longer displays the language toggle.
-

9. In the browser, step through to the confirmation page but switch languages once along the way using the language toggle. Note that when you complete an order, the confirmation page inadvertently switches back to the originally displayed language. You may rightly identify the cause: upon a successfully completed order, the `ControllerServlet` destroys the user session and consequently the session-scoped locale that was set using the `<fmt:setLocale>` tag is also lost.

To remedy this, open the `ControllerServlet` and locate the `invalidate()` method which is used to destroy user sessions (approximately line 259).

Use the editor's quick search facility: press `Ctrl-F` (`⌘-F` on Mac) and type in `'invalidate'`.

10. Add code that extracts the session-scoped locale value prior to destroying the user session and resets the request-scoped language attribute to the locale value after the session has been destroyed. (Changes in **bold**.)

```
// if order processed successfully send user to confirmation page
if (orderId != 0) {

    // in case language was set using toggle, get language choice before destroying
    session
    Locale locale = (Locale)
    session.getAttribute("javax.servlet.jsp.jstl.fmt.locale.session");
    String language = "";

    if (locale != null) {

        language = (String) locale.getLanguage();
    }

    // dissociate shopping cart from session
    cart = null;

    // end session
    session.invalidate();

    if (!language.isEmpty()) { // if user changed language
        using the toggle, // reset the language attribute

        - otherwise // language will be switched on
            request.setAttribute("language", language); // language will be switched on
            confirmation page!
    }

    // get order details
    Map orderMap = orderManager.getOrderDetails(orderId);

    ...
    userPath = "/confirmation";
}
```

11. Run the project and again, step through to the confirmation page but switch languages once along the way using the language toggle. Note that when you complete an order, the confirmation page now displays in the language you selected.

You have now successfully integrated language support into the `AffableBean` application according to customer specification. You've factored out all text from page views, placed it into resource bundles, and have applied JSTL's `fmt` tag library to use resource bundle content based on the user's preferred language. You also implemented a language toggle that enables users to switch between English and Czech, and override their browser's default language choice. Download and examine [snapshot 10](#) to compare your work with the state of the project at the end of this tutorial unit.

[Send Us Your Feedback](#)

See Also

NetBeans Resources

- [Introduction to Java EE Technology](#)
- [Getting Started with Java EE Applications](#)

- [Keyboard Shortcuts & Code Templates Card](#)
- [Java EE & Java Web Learning Trail](#)

External Resources

- [The Java Tutorials: Internationalization](#)
- [Java EE 5 Tutorial: Internationalizing and Localizing Web Applications](#)
- [Developing Multilingual Web Applications Using JavaServer Pages Technology](#)
- [Internationalization: Understanding Locale in the Java Platform](#)
- [Java Internationalization: Localization with ResourceBundles](#)
- [A JSTL primer, Part 3: Presentation is everything](#)
- [Java Internationalization](#) [Technology Homepage]
- [Internationalization and localization](#) [Wikipedia]
- [ISO 639-2 Language Code List](#) [Library of Congress]
- [W3C Internationalization Activity: Articles, best practices & tutorials: Language](#)
- [jQuery](#)