© Copyright 2003, AudioAudit, Inc.

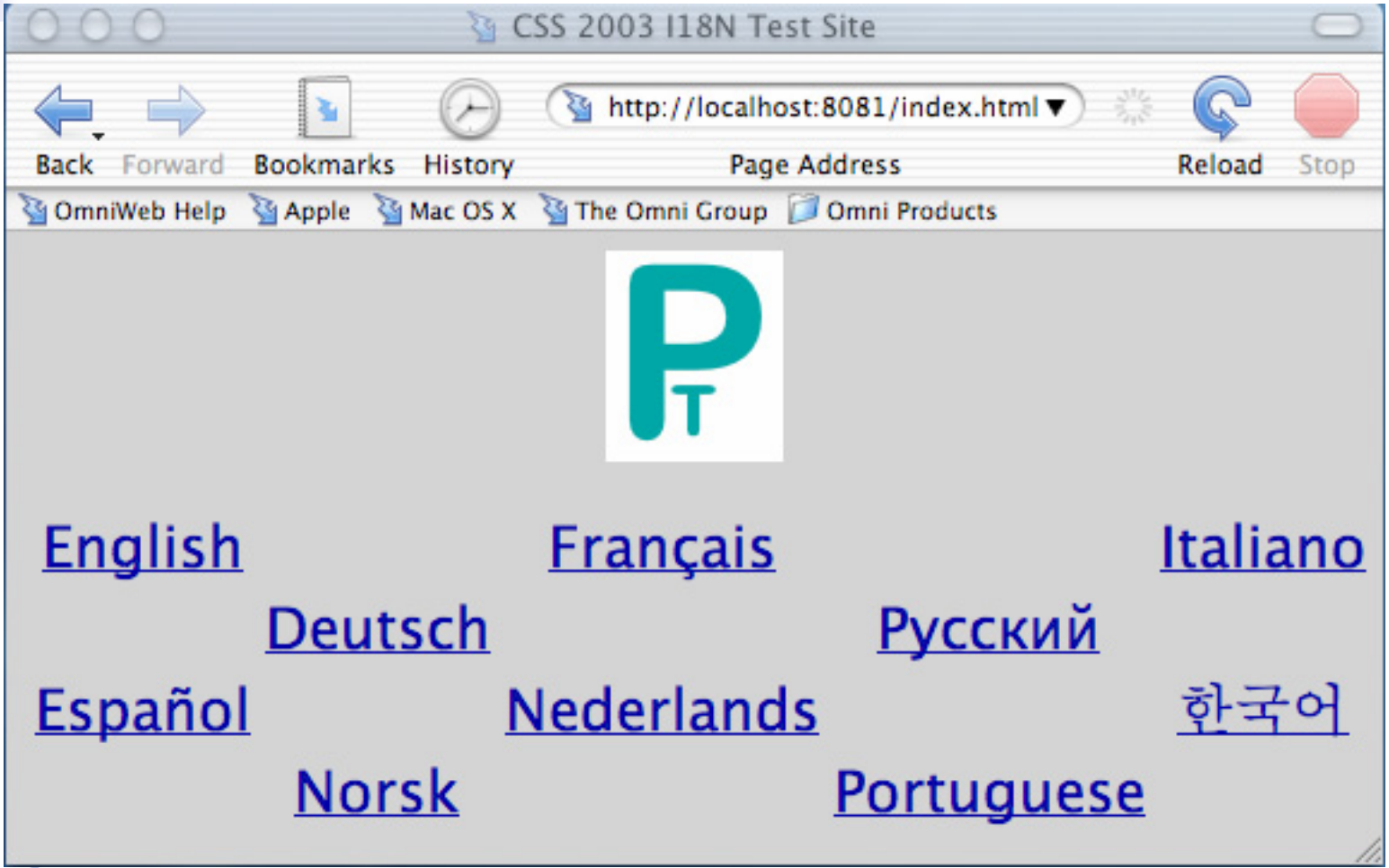
Localizing and Customizing JavaServer Pages

Paul Tremblett

AudioAudit, Inc.

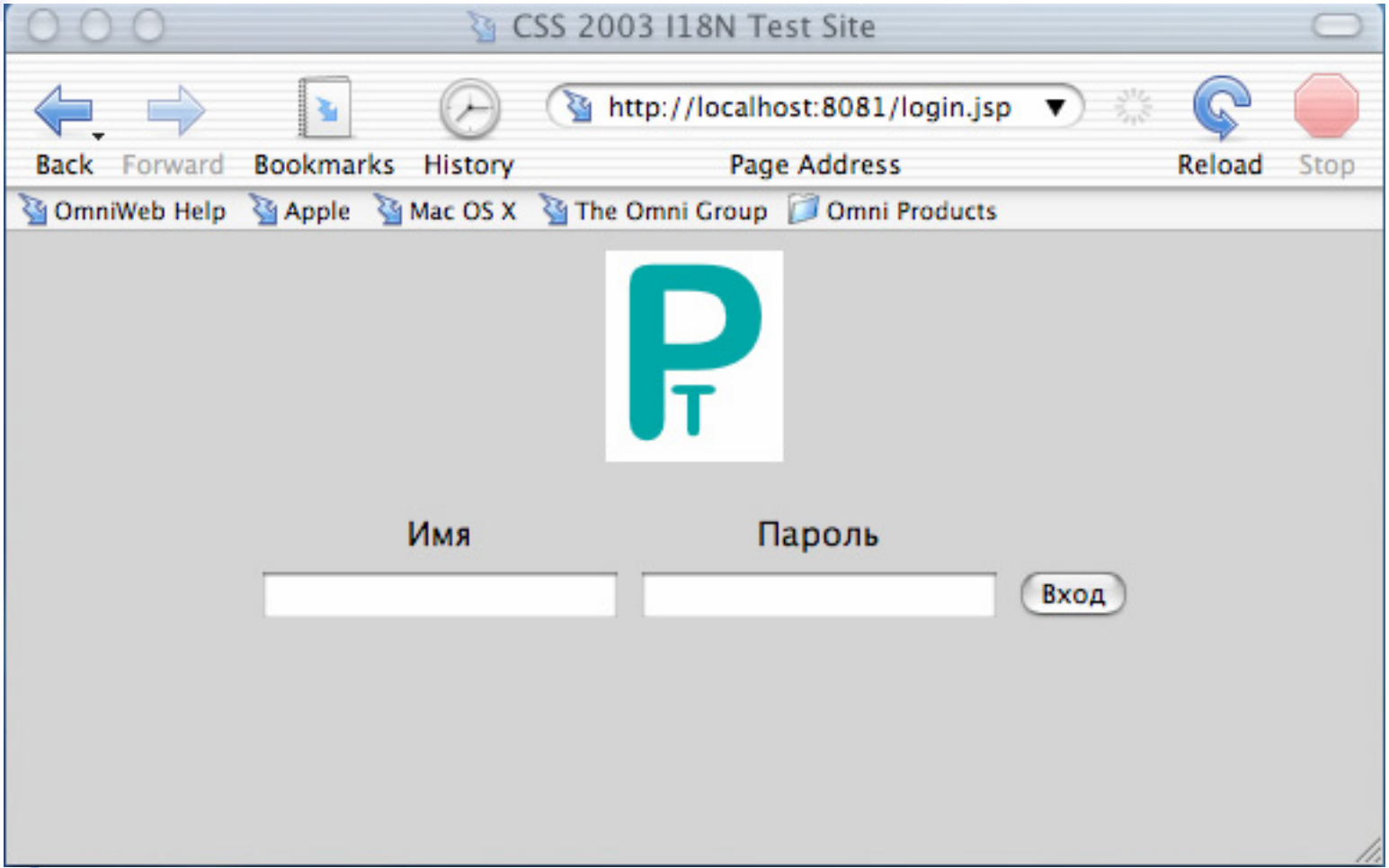
© Copyright 2003, AudioAudit, Inc.

A Simple Web Site (1)



© Copyright 2003, AudioAudit, Inc.

A Simple Web Site (2)



© Copyright 2003, AudioAudit, Inc.

Internationalization

- The process of designing software that can be adapted to various languages and regions without recompilation
- Called i18n for short
  - There are 18 letters between the leading **i** and the terminating **n**

© Copyright 2003, AudioAudit, Inc.

Characteristics

- A single executable can be run worldwide
- Textual elements are not hard-coded
- Addition of a new language does not require recompilation
- The format of dates, currencies and other culturally dependent data conforms to the user's region and language
- Can be localized quickly

© Copyright 2003, AudioAudit, Inc.

Localization

- The implementation of a specific language for an internationalized application
  - The creation of localized objects according to a specific region's rules
- Called l10n for short

© Copyright 2003, AudioAudit, Inc.

The Relationship Between i18n and l10n

- i18n provides a common component which is used by l10n to construct specific components
- The two are mutually interdependent

© Copyright 2003, AudioAudit, Inc.

Starting at the Ground Level

- Yes, this session is about JavaServer Pages, but…
  - The JSP author's task, as we will see, is easy
  - It depends, however, upon understanding and using some basic internationalization and localization techniques
- Therefore…
  - We will start at the ground level and build our way up

© Copyright 2003, AudioAudit, Inc.

Hello World!

- Let's start with something we all know

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello world!");
  }
}
```

- What's good about this program?
  - Everything, if it will only be run by users who speak English
  - Nothing, as we will shortly discover, if you have to provide multiple versions of it in different languages

© Copyright 2003, AudioAudit, Inc.

The Brute Force Approach

```
public class HelloFrance {
    public static void main(String[] args) {
        System.out.println("Bonjour La France !");
    }
}


public class HelloRussia {
    public static void main(String[] args) {
        System.out.println("Привет Россия!");
    }
}
```

- An endless effort
- A maintenance nightmare

© Copyright 2003, AudioAudit, Inc.

The Properties File

- Contains entries of the form:
  - key=value
- File name has the format:
  - BaseName_<lc>_<cc>.properties where:
    - lc = language code
    - cc = country code
  - Some examples are:
    - MyMessages_en_US.properties
    - MyMessages_fr_FR.properties

© Copyright 2003, AudioAudit, Inc.

Translated Properties Files

```
MyMessages.properties contents:

greeting = Hello.
```

```
MyMessages_en_US.properties contents:

greeting = Hello.
```

```
MyMessages_fr_FR.properties contents:

greeting = Bonjour.
```

Notes:

*key remains constant and is usually in the default language

*value is translated

© Copyright 2003, AudioAudit, Inc.

The Locale Object

- Identifies a particular language and country
- Constructor takes two arguments:
  - Language code (defined by ISO-639)
  - Country code (defined by ISO-3166)
- Only an identifier; actual locale-specific work is performed by object that uses the Locale object
- Example: `Locale here = new Locale("en", "US");`

© Copyright 2003, AudioAudit, Inc.

The ResourceBundle Object

- Contains locale-specific objects
- Used to isolate locale-sensitive data
- Belongs to a family whose members share a common base name
- Base name is modified by components that identify the locale
- Loaded using the static method **`getBundle()`** of class **`ResourceBundle`**

© Copyright 2003, AudioAudit, Inc.

Resource Bundle

- Contains keys and values
- Keys are always Strings
- Values can be any type of object
- Values are retrieved using the appropriate getter method:
  - `getString()`
  - `getStringArray()`
  - `getObject()`

© Copyright 2003, AudioAudit, Inc.

InternationalHello

```
if (args.length != 2) {
  language = new String("en");
  country = new String("US");
}
else {
  language = new String(args[0]);
  country = new String(args[1]);
}
Locale currentLocale;
ResourceBundle messages;
currentLocale = new Locale(language, country);
messages =
ResourceBundle.getBundle(”MyMessages",currentLocale);
System.out.println(messages.getString(”greeting"));
```

© Copyright 2003, AudioAudit, Inc.

Running The Program

```
gribzoid~/CSS2003/I18N$ java InternationalHello

Hello.


gribzoid~/CSS2003/I18N$ java InternationalHello en US

Hello.


gribzoid~/CSS2003/I18N$ java InternationalHello fr FR

Bonjour.
```

© Copyright 2003, AudioAudit, Inc.

So That's It?

- If it were that simple, there would not be entire books and papers devoted to internationalization and localization
- The devil is in the details

© Copyright 2003, AudioAudit, Inc.

Dates and Times

- A **`Date`** object represents an instant in time with millisecond precision and is unaware of locale
- The **`toString()`** method of the **`Date`** class always displays the date/time in a fixed format
- The **`DateFormat`** class provides predefined, locale-sensitive formatting styles

© Copyright 2003, AudioAudit, Inc.

The DateFormat Class

- Provides formatters for both dates and times
  - `getDateInstance()`
  - `getDateTimeInstance()`
  - `getTimeInstance()`
    - There are three forms of each of these methods:
      - ✓ `()`
      - ✓ `(int style)`
      - ✓ `(int style, Locale locale)`

© Copyright 2003, AudioAudit, Inc.

Formatting Styles

- The following fields control the length of a formatted date/time
  - **DEFAULT**
  - **SHORT**
  - **MEDIUM**
  - **LONG**
  - **FULL**
- The exact format of each is locale-specific

© Copyright 2003, AudioAudit, Inc.

FormatDateTime

```
Date now = new Date();
System.out.println("For locale: " + args[0] + "_" + args[1]);
System.out.println("\tDATE (DEFAULT) = " +
  DateFormat.getDateInstance(DateFormat.DEFAULT, locale).format(now));
System.out.println("\tDATE (LONG) = " +
  DateFormat.getDateInstance(DateFormat.LONG, locale).format(now));
System.out.println("\tDATE (FULL) = " +
  DateFormat.getDateInstance(DateFormat.FULL, locale).format(now));
System.out.println("\tTIME (DEFAULT) = " +
  DateFormat.getTimeInstance(DateFormat.DEFAULT, locale).format(now));
```

© Copyright 2003, AudioAudit, Inc.

FormatDateTime Output

```
For locale: en_US
        DATE (DEFAULT) = Aug 18, 2003
        DATE (SHORT) = 8/18/03
        DATE (MEDIUM) = Aug 18, 2003
        DATE (LONG) = August 18, 2003
        DATE (FULL) = Monday, August 18, 2003
        TIME (DEFAULT) = 7:14:18 PM
        TIME (SHORT) = 7:14 PM
        TIME (MEDIUM) = 7:14:18 PM
        TIME (LONG) = 7:14:18 PM EDT
        TIME (FULL) = 7:14:18 PM EDT
For locale: en_CA
        DATE (DEFAULT) = 18-Aug-2003
        DATE (SHORT) = 18/08/03
        DATE (MEDIUM) = 18-Aug-2003
        DATE (LONG) = August 18, 2003
        DATE (FULL) = Monday, August 18, 2003
        TIME (DEFAULT) = 7:21:51 PM
        TIME (SHORT) = 7:21 PM
        TIME (MEDIUM) = 7:21:51 PM
        TIME (LONG) = 7:21:51 EDT PM
        TIME (FULL) = 7:21:51 o'clock PM EDT
```

© Copyright 2003, AudioAudit, Inc.

Numbers and Currencies

- Objects that represent numbers understand only magnitude and direction – not locale
- The **`toString()`** method of an object that represents a number always displays the number in a fixed format
- The **`NumberFormat`** class provides predefined, locale-sensitive formatting styles

© Copyright 2003, AudioAudit, Inc.

The NumberFormat Class

- Provides formatters for numbers, percentages and currencies
  - **`getCurrencyInstance()`**
  - **`getIntegerInstance()`**
  - **`getNumberInstance()`**
  - **`getPercentInstance()`**
    - There are two forms of each of these methods:
      - ✓ **`()`**
      - ✓ **`(Locale locale)`**

© Copyright 2003, AudioAudit, Inc.

FormatNumbers

```
Integer wholeNumber = new Integer(48);
Double fractionalNumber = new Double(6.0238);
Double percent = new Double(0.2345);
Double moneyAmount = new Double(298.45);
NumberFormat nf = NumberFormat.getIntegerInstance(locale);
System.out.println("whole number looks like: " +
nf.format(wholeNumber));
nf = NumberFormat.getNumberInstance(locale);
System.out.println("fractional number looks like: " +
  nf.format(fractionalNumber));
nf =  NumberFormat.getPercentInstance(locale);
nf.setMaximumFractionDigits(2);
System.out.println("percent looks like: " +
  nf.format(percent));
nf = NumberFormat.getCurrencyInstance(locale);
System.out.println("currency looks like: " +
  nf.format(moneyAmount));
```

© Copyright 2003, AudioAudit, Inc.

FormatNumbers Output

```
gribzoid~/css2003/i18n$ java FormatNumbers en US

for Locale: en_US
  whole number looks like: 48
  fractional number looks like: 6.024
  percent looks like: 23%
  currency looks like: $298.45


gribzoid~/css2003/i18n$ java FormatNumbers fr FR

for Locale: fr_FR
  whole number looks like: 48
  fractional number looks like: 6,024
  percent looks like: 23%
  currency looks like: 298,45 €
  gribzoid~/css2003/i18n$
```

© Copyright 2003, AudioAudit, Inc.

Sentence Structure

- Sentence structure is not the same in all languages
  - I saw a brown dog
  - J'ai vu un chien brun.
  - I have read “Der Zauberberg”.
  - Ich habe “Der Zauberberg” gelesen.

© Copyright 2003, AudioAudit, Inc.

Compound Messages

- Internationalizing a compound message is accomplished in five steps:
  - Separate variable content and static content
  - Create a ResourceBundle
  - Set the message arguments
  - Create a message formatter
  - Format the message
    - We will examine each of these steps in detail

© Copyright 2003, AudioAudit, Inc.

Separating Content

- Identify those elements of the sentence that represent variable data
  - At [time] on [date], the total was [n].

© Copyright 2003, AudioAudit, Inc.

Creating a ResourceBundle

```
#SumLog_en_US.properties
template = At {0,time,medium} on {0,date,long}, the total was \
{1,number,integer}.

#SumLog_de_DE.properties
template = Um {0,time,medium} am {0,date,long} war die Betrage \
{1,number,integer}.
```

© Copyright 2003, AudioAudit, Inc.

Setting Message Arguments

```
ResourceBundle messages =
  ResourceBundle.getBundle(”SumLog",currentLocale);

Object[] messageArguments = {
  new Date(),
  new Integer(total)
};
```

© Copyright 2003, AudioAudit, Inc.

Creating Message Formatter

```
MessageFormat formatter = new MessageFormat("");
formatter.setLocale(currentLocale); };
```

© Copyright 2003, AudioAudit, Inc.

Formatting the Message

```
formatter.applyPattern(messages.getString("template"));
System.out.println(formatter.format(messageArguments));
```

© Copyright 2003, AudioAudit, Inc.

CompoundMessage

```
gribzoid~/CSS2003/I18N$ java CompoundMessage en US
For locale: en_US

        At 4:53:25 PM on August 19, 2003, the total was 0.
        At 4:53:35 PM on August 19, 2003, the total was 5.
        At 4:53:45 PM on August 19, 2003, the total was 10.
        At 4:53:55 PM on August 19, 2003, the total was 15.


gribzoid~/CSS2003/I18N$ java CompoundMessage de DE
For locale: de_DE

        Um 16:54:14 am 19. August 2003 war die Betrage 0.
        Um 16:54:24 am 19. August 2003 war die Betrage 5.
        Um 16:54:34 am 19. August 2003 war die Betrage 10.
        Um 16:54:44 am 19. August 2003 war die Betrage 15.
```

© Copyright 2003, AudioAudit, Inc.

The Problem with Plurals

- In many languages, you use a different word for a single item than you use for multiple items
- Verbs often have a different plural form
- This can result in generalized messages that contain incorrect grammar

© Copyright 2003, AudioAudit, Inc.

Dealing with Plurals

- Plurals can be handled properly by following these seven steps:
  - Represent the message as a pattern
  - Create a ResourceBundle
  - Create a message formatter
  - Create a choice formatter
  - Apply the pattern
  - Assign the formats
  - Set the arguments and format the message
    - We will examine each of these steps in detail

© Copyright 2003, AudioAudit, Inc.

Creating a Pattern

dirname	contains	no files
dirname	contains	one file
dirname	contains	n files

variable data

Pattern: {1} {2} {3}.

© Copyright 2003, AudioAudit, Inc.

Creating a ResourceBundle

```
#FilesBundle_en_US.properties
noFiles = no files
oneFile = one file
multipleFiles = {2} files
verbForNone = contains
verbForSome = contains
pattern = {0} {1} {2}.

#FilesBundle_fr_FR.properties
noFiles = pas de fichier
oneFile = un fichier
multipleFiles = {2} fichiers
verbForNone = ne contient
verbForSome = contient
pattern = {0} {1} {2}.
```

© Copyright 2003, AudioAudit, Inc.

Creating a MessageFormatter

```
MessageFormat messageFormat = new MessageFormat("");
messageFormat.setLocale(locale);
```

© Copyright 2003, AudioAudit, Inc.

Creating a ChoiceFormatter

```
double [] verbLimits = {0,1};
String[] verbForms = {
  rb.getString("verbForNone"),
  rb.getString("verbForSome")
};
ChoiceFormat verbChoiceFormat =
  new ChoiceFormat(verbLimits, verbForms);

double[] fileLimits = {0,1,2};
String [] fileQuantities = {
  rb.getString("noFiles"),
  rb.getString("oneFile"),
  rb.getString("multipleFiles")
};
ChoiceFormat fileChoiceFormat =
  new ChoiceFormat(fileLimits, fileQuantities);
```

© Copyright 2003, AudioAudit, Inc.

Applying the Pattern

```
String pattern = rb.getString("pattern");
messageFormat.applyPattern(pattern);
```

© Copyright 2003, AudioAudit, Inc.

Assigning the Formats

```
Format[] formats = {null, verbChoiceFormat, fileChoiceFormat};
messageFormat.setFormats(formats);
```

© Copyright 2003, AudioAudit, Inc.

Setting Arguments and Formatting Message

```
Object[] messageArguments = {"'home'",null, null};

for (int fileCount = 0; fileCount < 4; fileCount++) {
  messageArguments[1] = new Integer(fileCount);
  messageArguments[2] = new Integer(fileCount);
  String result = messageFormat.format(messageArguments);
  System.out.println(result);
}
```

© Copyright 2003, AudioAudit, Inc.

FormatSingularOrPlural

```
gribzoid~/CSS2003/I18N$ java FormatSingularOrPlural en US
For locale: en_US
'home' contains no files.
'home' contains one file.
'home' contains 2 files.
'home' contains 3 files.

gribzoid~/CSS2003/I18N$ java FormatSingularOrPlural fr FR
For locale: fr_FR
'home' ne contient pas de fichier.
'home' contains un fichier.
'home' contains 2 fichiers.
'home' contains 3 fichiers.
```

© Copyright 2003, AudioAudit, Inc.

Finally, the Web!

- For command-line and Swing applications, you must bring the application to the customer
  - This means you have plenty of time
- With a Web application, the customer comes to you
  - This means you had better be prepared or ready to adapt quickly

© Copyright 2003, AudioAudit, Inc.

So, Do We See Internationalized JSPs Now?

- When you first learn to use a hammer, you are inclined to view everything as a nail
- Just because you know how to use i18n and l10n tools does not necessarily mean they are appropriate for every task
- Make some intelligent decisions up front

© Copyright 2003, AudioAudit, Inc.

When Are JSPs Appropriate?

- Remember that JSPs are all about dynamic data
- If your site content is static, it makes no sense to incur the overhead of performing message translation every time a page is requested
- The cost of translation is constant

© Copyright 2003, AudioAudit, Inc.

If You Do Decide on JSPs

- An internationalized application should follow the same design rules as any other well-designed application
- A servlet acting as a controller is often appropriate

© Copyright 2003, AudioAudit, Inc.

The Best Tool for the Job

- Tap into what others before you have learned:
  - Scriptlets are:
    - Ugly
    - Prone to unintentional changes by content provider
    - Difficult for the programmer to maintain
  - The real power of JSPs is recognized when you use custom tags

© Copyright 2003, AudioAudit, Inc.

JSTL to the Rescue

- The Internationalization Tags included in the Java Standard Tag Library provide mechanisms to:
  - Specify locales
  - Format messages
  - Format numbers and dates

© Copyright 2003, AudioAudit, Inc.

Using i18n Tags

- Include a <taglib> directive

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

- Use the tags just as you would any other custom tag

```
<fmt:setLocale value="${param.locale}" scope="page"/>
```

© Copyright 2003, AudioAudit, Inc.

The <fmt:locale> Action

- Specifies the locale used by resource bundle lookups and formatting actions

- Syntax:

```
<fmt:locale value="locale"
   [variant="variant"]
   [scope="{page|request|
     session|application}"]/>
```

© Copyright 2003, AudioAudit, Inc.

The <fmt:bundle> Action

- Loads a resource bundle

- Syntax:

```
<fmt:bundle basename="basename"
   [prefix="prefix"]
   [var="varName"]
   [scope="{page|request|
     session|application}"]>
  body content
</fmt:bundle>
```

© Copyright 2003, AudioAudit, Inc.

The <fmt:message> Action(1)

- Looks up a localized message in a resource bundle

- Syntax 1 – without body content:

```
<fmt:message key="messageKey"
   [bundle="resourceBundle"]
   [var="varName"]
   [scope="{page|request|
     session|application}"]/>
```

© Copyright 2003, AudioAudit, Inc.

The <fmt:message> Action(2)

- Syntax 2 – with a body to specify message parameters:

```
<fmt:message key="messageKey"
    [bundle="resourceBundle"]
    [var="varName"]
    [scope="{page|request|
      session|application}"]>
    <fmt:param> subtags
</fmt:message>
```

© Copyright 2003, AudioAudit, Inc.

The <fmt:message> Action(3)

- Syntax 3 – with a body to specify key and optional message parameters:

```
<fmt:message [bundle="resourceBundle"]
    [var="varName"]
    [scope="{page|request|session|
      application}"]>
    key
    optional <fmt:param> subtags
</fmt:message>
```

© Copyright 2003, AudioAudit, Inc.

The <fmt:param> Action(1)

- Supplies a single parameter for parametric replacement to a containing <fmt:message>

- Syntax 1 – value specified *via* attribute "value":

```
<fmt:param value="messageParameter"/>
```

© Copyright 2003, AudioAudit, Inc.

The <fmt:param> Action(2)

- Syntax 2 – value specified *via* body content:

```
<fmt:param>
    body content
  </fmt:param>
```

© Copyright 2003, AudioAudit, Inc.

The <fmt:requestEncoding> Action

- Sets the request's character encoding

- Syntax:

```
<fmt:requestEncoding
        [value="charsetName"]/>
```

© Copyright 2003, AudioAudit, Inc.

Code Is Worth 1000 Slides

- The remainder of the presentation will be spent examining code that uses the internationalization tags
- Demo examples are included in the Java Web Services Developer Pack (JWSDP)
  - Download from java.sun.com

© Copyright 2003, AudioAudit, Inc.

But Before We Get Started…

- The JavaServer Faces technology also deals with i18n and l10n issues
- For more information on JavaServer Faces, you should attend Stephen Stelting's session entitled "About JavaServer Faces"