

Java Server Pages (JSP) and MySQL Internationalization, Localization, and Turkish Language Support

*Bulut F. Ersavaş, M.S.
bersavas@netscape.net*

21 February 2005

(This article has been published in www.teknoturk.org in Turkish.)

1. Introduction

There are two big challenges to build a web service that will provide content with multiple languages. These are (i) developing the application so that it supports many region and language specific elements such as various character sets and (ii) setting up the database so that it can handle those elements. Most basically, a multilingual web service should be able to receive, process, and display text including international characters.

A common problem is that it is not always clearly described how to make a collection of tools (e.g. application server, database server) and technologies (e.g. java, SQL) work together to support many languages and character encodings. This article, based on working models, explains how to make java applications work with MySQL databases with multiple language and character sets.

This article includes examples of an application that supports both English and Turkish interfaces (i.e. menus, links etc. are presented in a language preferred by user). The application includes dynamic content (stored in database) from both languages, therefore, should be able to support characters included in one language but not in the other (e.g. x and ö).

I will use UTF-8 as the default character encoding. This is what I recommend for those who would like to have more than one language support in their applications (especially Turkish or any other European Language). UTF-8, which stands for *Unicode Transformation Format-8*, is an 8-bit lossless encoding of Unicode characters. However, the information and sample code excerpts given in this article can be applied to any character encoding. Reader should replace any "utf-8" reference with his/her selection of encoding.

2. Using Emacs with International Character Sets

A developer, working with JSPs, Tomcat and MySQL, is very likely to use Emacs for editing and managing source files. This section briefly describes how to customize Emacs to support a specific character encoding.

To read and write all ".jsp" files using UTF-8 coding system, add the following line to your .emacs file (most likely in your home directory). You can repeat the same expression for different file extensions (e.g. .html) and coding systems (e.g. 'iso-8859-9).

```
(modify-coding-system-alist 'file "\\\\.jsp\\" 'utf-8)
```

Once you attach .jsp files to UTF-8 encoding, you can use Turkish characters in your source file without any problem.

In emacs, you can select one of many ways to input special characters of one language. To see the description of one input method, press "**Ctrl-h Shift-i**" or **M-x** and type **describe-input-method** and then select the language/input method (hit Tab to see a list of selections). To set the input method use "**M-x set-input-method**". To toggle between two input methods, you can use the "**Ctrl-`**" shortcut. You can set the default language preferences for your emacs by adding the following lines into your .emacs initialization file:

```
;; init file should only have one instance of custom-set-variables
(custom-set-variables
 '(case-fold-search t) ;; optional
 '(current-language-environment "Turkish")
 '(default-input-method "turkish-postfix")
 '(global-font-lock-mode t nil (font-lock)) ;; optional
 '(transient-mark-mode t) ;; optional
)
```

Generally, a standard emacs installation includes support for most of the character encodings, if not all. However, if this is not the case for your copy, please refer to GNU Emacs Manual [1] to install additional character encodings. Other useful commands to use in Emacs for character set support are as follows:

Note: M-x stands for meta-x and is the command prompt reached usually by pressing **left-Alt** key with **x**.

M-x prefer-coding-system: This command reads the name of a coding system from the minibuffer, and adds it to the front of the priority list, so that it is preferred to all others. [1]

M-x set-buffer-file-coding-system: If you want to write files from current buffer using a different coding system, you can specify it for the buffer using this command. [1]

M-x set-keyboard-coding-system: Use this command to set keyboard coding system to insert characters directly in this coding. [1]

3. Internationalization and Localization

Locale is a set of region specific elements such as character set, currency and time format represented in an application. Separating the locale dependencies from application's source code is called internationalization. Localization is adapting such an application to a specific locale. [2]

Java provides **java.util.Locale** class to represent a specific geographical, political and cultural region. To contain locale-specific objects, resource bundles are used (**java.util.ResourceBundle**). For more information on these classes please refer to [3]. A simple example of how to use Locale and ResourceBundle in your JSP code is given below.

```
// Get Turkish Resource Bundle:
Locale locale = new Locale("tr", "TR");
ResourceBundle messages = ResourceBundle.getResource("Message", locale);
// Get the string for "Hello World" translated to Turkish:
String myString =
messages.getString("messages.hello_world");
out.println(myString);
```

In the example above, the Turkish translation of the `hello_world` message is included in an ASCII file called bundle (e.g. `Message_tr_TR.properties`). `ResourceBundle`'s `getString` function retrieves the requested message from this file. For every locale selected, there must be one such file, otherwise, java uses the default bundle (i.e. any bundle with no country or language code). Here is a mapping for the bundle naming:

```
BundleName + "_" + localeLanguage + "_" + localeCountry + "_" + localeVariant
BundleName + "_" + localeLanguage + "_" + localeCountry
BundleName + "_" + localeLanguage
BundleName
```

Here is an example excerpt from a bundle named `Message_en_US.properties`:

```
movie.title=Title
movie.starring=Starring
movie.director=Director
```

A bundle written in native character encoding should be converted to ASCII format by using Java's `"native2ascii"` utility (located under `/<java-sdk-install-dir>/bin/`). This is necessary for the web application server (e.g. Tomcat) to correctly display the character set. A sample usage for this utility is as follows:

```
native2ascii -encoding ISO-8859-9 tr.src Message_tr_TR.properties
```

In this example the source bundle (`tr.src`) is written in ISO-8859-9 encoding. You can keep your bundles under `/<jakarta-install-dir>/webapps/<app-name>/WEB-INF/classes` directory.

Here is an excerpt from `Message_tr_TR.properties` bundle after running `native2ascii`. Notice that "İ" is converted to `\u0130` and "ö" is converted to `\u00f6`:

```
movie.title=\u0130sim
movie.starring=Oyuncular
movie.director=Y\u00f6netmen
```

3.1. How to Use Locale in JSP

You can set a variable in client's session to keep what locale he/she would like to use. Depending on your preference, you can also store user's language choice as a cookie or pass it as a request parameter between JSP pages. Here is an example with session:

```
Locale locale = (Locale) session.getValue("myLocale");

if (session.getValue("myLocale") == null)
{
    // Default Language Setting
    locale = new Locale("tr", "TR");
```

```
        session.putValue("myLocale", locale);
    }
```

Here is an example on how to use the ResourceBundle in your JSP code:

```
<%
ResourceBundle bundle =
    ResourceBundle.getBundle("Message", locale);
%>

<select name="searchColumn">
    <option value="name">
<%=bundle.getString("movie.title")%></option>
    <option value="starring"> <%=bundle.getString("movie.starring")%></option>
    <option value="director"> <%=bundle.getString("movie.director")%></option>
</select>
```

4. JSP Character Set Handling

Character handling can be split into two categories: displaying the characters and receiving the ones entered by the user. For JSP pages, setting the encoding for each category is done separately by using different directives and/or functions.

4.1. Displaying International Characters

To set the character encoding for JSP page display, use the standard "page" directive with "contentType" parameter as follows:

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
```

Page directive is used to control the structure of a servlet or a JSP by importing classes, customizing superclasses, and setting the content type, etc.

I also recommend using the following HTML tag for the web browser to load the correct character set:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

Meta tags with an http-equiv attribute are the same as HTTP headers. In general, they are used to control the action of browsers, and can be used to refine the information provided by the actual headers. [4] In the example given above, the HTTP content type is extended to specify the character set.

Avoid using Java String functions (or constructors) such as the following to convert character encoding of a string, because it is both inefficient and unnecessary. Once you set all the options mentioned in this article correctly, you will not need to use such a conversion.

```
str = new String(request.getParameter("value").getBytes("ISO-8859-1"), "UTF-8");
```

4.2. Retrieving Data from HTML Forms with International Characters

You need to set the request encoding in JSP with setCharacterEncoding method to be able to receive user inputs entered through the forms properly. Below is the call needed in the JSP file:

```
<% request.setCharacterEncoding("UTF-8"); %>
```

You can verify that the character encoding is set correctly for JSP request by printing out the result of the following function call:

```
<% request.setCharacterEncoding(); %>
```

This should return "UTF-8". If "setCharacterEncoding" does not work for you, you can also try to set the encoding directly in HTML form tag as follows: (see reference [6], section 17 - forms, for more details)

```
<FORM action="..." method="..." accept-charset="UTF-8"> ...form content... </FORM>
```

5. JSP – MySQL Connection Setup

When connecting to MySQL database from your JSP or Java Servlet class using mysql-connector-java or MySQL Connector/J driver (JDBC driver directly from MySQL) connect as follows (notice the useUnicode and characterEncoding argument) to correctly pass the right character encoding from your JSP pages to MySQL database server. (see reference [7] for details)

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
...
try {
    Connection conn = java.sql.DriverManager.getConnection(
    "jdbc:mysql://localhost/dbName?user=username&password=usersPassword&useUnicode
    =true&characterEncoding=UTF-8");

    // Use the Connection...

} catch (SQLException ex) {
    // handle errors, if any
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}
```

If you are using MySQL version 4.1 or later (highly recommended), you can also set the default collation for the connection with the following statements. These function-calls set the variables used by MySQL to define the default collation. You can also specify what collation to use in your SQL statements. Please see section 6 of this article and MySQL documentation [8] for details on collation.

```
conn.createStatement().execute("SET character_set_client=utf8");
conn.createStatement().execute("SET character_set_connection=utf8");
conn.createStatement().execute("SET character_set_results=utf8");
conn.createStatement().execute("SET character_set_database=utf8");
conn.createStatement().execute("SET character_set_server=utf8");

conn.createStatement().execute("SET collation_connection=utf8_turkish_ci");
conn.createStatement().execute("SET collation_database=utf8_turkish_ci");
conn.createStatement().execute("SET collation_server=utf8_turkish_ci");
```

It is optional to set these for connection, because you can specify character set / collation within your query or when you are creating your table. See next section for examples.

6. MySQL Character Sets and Collation

As a good practice, if you are starting to develop your application from scratch, use UTF-8 character encoding because this standard encoding covers most of the international character sets and is supported by recent versions of all major web browsers (e.g. Netscape, Internet Explorer). Although ISO-8859-9 Turkish encoding (i.e. latin5) sounds very appealing, it is better to use UTF-8 with MySQL for Turkish characters. Furthermore, if you will adapt your application to many other languages, you will not have to change your character set and collation for every new language you localize to (as long as they are supported by UTF8).

Starting from version 4.1, MySQL has extensive collation support. Collation is a collection of rules to compare characters in a character set. For example, when you are making a case-insensitive query, database should be able to match lowercase character (e.g. a) with the uppercase character (e.g. A). The real complication is with conflicting international characters. For example, in Turkish, uppercase for letter "i" is "İ" and lowercase for letter "I" is "ı". A database server that does not know about this property of the Turkish character (i.e. that uses a collation table for English) will return wrong results for case-insensitive queries that involve "i", "ı", "İ", and "I" characters. This problem is well addressed in MySQL versions starting from 4.1. Under MySQL, UTF8 character set supports one case-insensitive collation for Turkish called "utf8_turkish_ci" and latin5 (ISO 8859-9) supports two collations that are "latin5_bin" and "latin5_turkish_ci".

For the data entered into and received from MySQL database server, you can set the default character set and collation at five levels: (i) server, (ii) database, (iii) table, (iv) column, and (v) connection. More information and example for each follows:

- i. When you start the database server:

```
mysqld --default-character-set=utf8 \  
      --default-collation=utf8_turkish_ci
```

- ii. When you are creating the database (or with alter statement after creation):

```
CREATE DATABASE db_name  
DEFAULT CHARACTER SET utf8 COLLATE utf8_turkish_ci;
```

- iii. When you are creating the table (or with alter statement after creation):

```
CREATE TABLE tbl_name (column_list)  
DEFAULT CHARACTER SET utf8 COLLATE utf8_turkish_ci;
```

- iv. When you are describing the columns during table creation:

```
CREATE TABLE tbl_name  
(  
    c1m_name VARCHAR(5) CHARACTER SET utf8 COLLATE utf8_turkish_ci  
);
```

- v. As described in section 5.

My personal preference is to set the character set and the collation at table or column level. Depending on the structure and content of your application, you can establish these settings at the database or server level. For example, if your application supports many languages; you might want to create separate tables for certain entries in each language with different default character set and collation.

6.1. Using Collate in SQL Queries

MySQL uses an expression called introducer in its SQL queries to tell to the parser what the character set is for the string, which follows it. Please note that introducer is not used for any conversion, rather for specifying the character set. Here is an example of its use:

```
SELECT uid FROM users WHERE name = _utf8'Ümit' COLLATE utf8_turkish_ci;
```

In this example, `_utf8` that precedes `'Ümit'` is the introducer which tells parser that string `'Ümit'` is in character set `utf8`. Introducer is written as MySQL charset name preceded by an underscore. If the character set and collation is not specified by introducer and `COLLATE` clause in the query, then MySQL uses the ones set by `character_set_connection` and `collation_connection` system variables. `COLLATE` clause overwrites the default collation for the comparison and can be used in various parts of the SQL statements. Please see MySQL reference [8] (section 10.3.8 as of this article's release date) for examples.

7. Conclusion

This article describes how to develop and setup JSP applications and MySQL databases to support multiple languages and various character-sets, especially Turkish. Code examples given throughout the article are sufficient to initialize a web service going with different character encodings. Users who require more detailed information should refer to the resources given in corresponding sections.

References:

- [1] *GNU Emacs Manual*, Free Software Foundation, 2004, Boston, MA, USA.
(<http://www.gnu.org/software/emacs/manual/>)
- [2] *Designing Enterprise Applications with the J2EE Platform, Second Edition*, Greg Murray, Sun Microsystems, Inc., 2002, Upper Saddle River, NJ, USA
(http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/)
- [3] *Java™ 2 Platform, Standard Edition, v 1.4.2 API Specification*, Sun Microsystems, Inc., 2003, Santa Clara, CA, USA
(<http://java.sun.com/j2se/1.4.2/docs/api/>)
- [4] *A Dictionary of HTML META Tags*, 1996, Vancouver Webpages.
(<http://vancouver-webpages.com/META/>)
- [5] *Core Servlets and JavaServer Pages*, First Edition, Marty Hall, Sun Microsystems Press/Prentice Hall PTR, May 2000
- [6] *HTML 4.01 Specification, W3C Recommendation*, Copyright © 1994-2002, World Wide Web Consortium, (<http://www.w3.org/TR/html401/cover.html>)

[7] *MySQL Connector/J Documentation*, © 1995-2005 MySQL AB.,
(<http://dev.mysql.com/doc/connector/j/en/index.html>)

[8] *MySQL Reference Manual*, © 1995-2005 MySQL AB.,
(<http://dev.mysql.com/doc/mysql/en/index.html>)