

Internationalization Report:

Unicode Enabling Java Web Applications

From Browser to DB

Provided by:



LingoPort, Inc.
1734 Sumac Avenue
Boulder, Colorado 80304
Tel: +1.303.444.8020
Fax: +1.303.484.2447
<http://www.lingoport.com>

Sales contact:
Adam Asnes
aasnes@lingoport.com

Copyright, Lingoport Inc.
January 28, 2005

NOT FOR GENERAL DISTRIBUTION WITHOUT PERMISSION

Table of Contents

Java Web Applications and Unicode	3
UTF-8 Enabling	3
Application Server UTF-8 Enabling.....	3
JSP File UTF-8 Enabling.....	3
HTML File UTF-8 Enabling.....	4
JavaScript UTF-8 Enabling.....	4
Database Unicode Enabling	4
Oracle Data Types.....	5
Other Encoding Tasks.....	6
Convert Properties Files to Unicode Escapes	6
About Lingoport	8

Java Web Applications and Unicode

Java is a language well suited to internationalization because of its native Unicode character support and robust locale-specific functionality. There are some steps that need to be taken, however, to enable Unicode support in a Java web application, and these steps are dependent on the components being used (database, application server, etc.). This document describes the steps to be taken to set up a Java web application to use Unicode.

UTF-8 Enabling

UTF-8 is the preferred encoding form of Unicode to use with HTML, and by extension JSP, Servlets, etc. because ASCII markup remains in ASCII, which means smaller file sizes being transferred over the Internet (assuming the files are more markup than content in the case of Asian languages).

To enable a Java web application to use UTF-8, it is necessary to include this information in certain key places so that the application knows what encoding to use when delivering output or receiving input. There are five possible areas for this to be defined:

1. The application server (startup script/app config)
2. The JSP files (page directive)
3. The HTML files (content-type meta tag)
4. The JavaScript files/code blocks (charset attribute)
5. The database (client and server)

Application Server UTF-8 Enabling

By default, an application server uses the encoding of the system on which it is running, defined in the system property `file.encoding`. On English systems this property will likely be defined as either ISO-8859-1 (Latin-1, Unix-based systems) or Cp1252 (Windows Latin-1 charset).

In the event this value is not correct for the application (as is the case for our purposes), some application server solutions recommend redefining the `file.encoding` property during system startup. For example, the Apache group recommends changing the Tomcat startup script (*catalina.bat* or *catalina.sh*) to add the switch `-Dfile.encoding=UTF-8` to the startup call to the Java executable. This ensures that the http response encoding will default to UTF-8, though this can be overridden within the java Servlet code as needed.

The Orion Application Server, to provide another example, uses a different approach. Under Orion, each web application has an Orion-specific configuration file called *orion-web.xml*. This file can be found under the application deployment location (*ORION_HOME/application-deployments/<deploymentName>/<warname(.war)>/orion-web.xml*), or in the application's *WEB-INF* directory if not deployment-directory is specified in *server.xml*. This file contains a `default-charset` value (set to ISO-8859-1 by default) that should be changed to UTF-8.

JSP File UTF-8 Enabling

The encoding of a JSP file is defined by placing a page directive at the top of the file and including `pageEncoding` and/or `contentType` attributes. In its most verbose form, here is a page directive that defines UTF-8 as the encoding/charset:

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

It might seem redundant to have both the `pageEncoding` and `contentType` definitions contain UTF-8, but they are used for different purposes. The first, `pageEncoding`, is an indication to the web container of the file's encoding so that it is compiled using that encoding. The `contentType` attribute, on the other hand, becomes the Content-Type header sent in the response to the client's request.

Ideally, setting the default character set of the application/application server to UTF-8 would be enough to ensure that the application uses UTF-8 as its encoding throughout. In practice, however, it is a good idea to be as explicit as possible to help ensure that any issues arise aren't related to encoding. This means including this page directive in every JSP file.

HTML File UTF-8 Enabling

An HTML file defines its encoding similarly to JSP, but defining its content-type. In HTML this is done using a meta tag:

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

The meta tag should be defined before the title element in the head of the document. (Note that case is not important; both "utf-8" and "UTF-8" will work.)

It could be argued that this is not strictly necessary, since the server's response already contains a Content-Type header with charset. While this is true, the same logic to justify the JSP page directive can be used here, which is that the more explicit you are in defining the encoding, the less likely that issues will arise related to encoding.

JavaScript UTF-8 Enabling

It is possible to indicate the encoding of a JavaScript block or file by including the `charset` attribute in the script start tag, for example:

```
<script src="functions.js" type="text/javascript" charset="UTF-8"></script>
```

In practice this is probably unnecessary provided that JavaScript has been properly internationalized (strings externalized, etc.). However, if a JavaScript code block or file happens to contain an extended character, including this attribute isn't a bad idea. Of course, it will be necessary to verify that code or file is actually encoded as UTF-8, otherwise the extended characters will not display correctly.

Database Unicode Enabling

This section will discuss Unicode enablement of Oracle and SQL Server db platforms. Other platforms have different Unicode configurations Steps.

Oracle

When Oracle is the db platform, the decision must be made whether the db should store character data in the UTF-8 (multibyte) Unicode encoding or in the UTF-16 (double-byte) Unicode encoding, as Oracle supports both. When the db is likely to contain mostly ASCII

character data, UTF-8 is generally the best solution. Within UTF-8, ASCII data is encoded with its original, single-byte ASCII code point, meaning no extra space is required for the storage of UTF-8 data that falls within the ASCII range. Character data beyond the ASCII range is encoded with either two or three bytes.

When the majority of character data in the db is non-ASCII data (such as Japanese, etc.) it is a wiser decision to use the UTF-16 encoding, which encodes each character with two bytes with the exception of surrogate pairs. This proves a space saver when lots of Asian characters are stored, as UTF-8 encodes each Asian character with three bytes.

Oracle UTF-8

The steps, if any, to communicate with a UTF-8 encoded Oracle database depend on how it is being accessed.

If the database is being accessed through such utilities as SQLPLUS, it is necessary to define an environmental variable on the client that indicates the encoding of the database you are connecting to. The following command sets the NLS_LANG environmental variable to UTF-8 for US on a Unix machine in the bash shell:

```
$ export NLS_LANG=American America.AL32UTF8
```

If a client is accessing the database via Java using either the JDBC thin driver or the OCI driver, it is not necessary to do anything additional on that client since both drivers will determine database encoding and transparently perform any necessary conversions (UTF-8 to UTF-16 in this case). This is true both for querying from and updating to the database.

Oracle UTF-16

In order to store character data in the UTF-16 encoding, Oracle requires use of NCHAR, And NVARCHAR2 data types. As mentioned previously in this document, Oracle refers to this double-byte Unicode encoding as AL16UTF16.

Oracle Data Types

Within Oracle, the character encoding for a database is usually defined when the database is created. With the exception of the UTF-16 encoding, the character encoding is specified as the CHARACTER SET parameter inside the CREATE DATABASE statement.¹ If UTF-16_-_AL16UTF16 is the desired database encoding, the encoding is instead specified as the NATIONAL CHARACTER SET parameter in the CREATE DATABASE statement:

```
CREATE DATABASE sample

CONTROLFILE REUSE
LOGFILE
GROUP 1 ('diskx:log1.log', 'disky:log1.log') SIZE 50K,
GROUP 2 ('diskx:log2.log', 'disky:log2.log') SIZE 50K
MAXLOGFILES 5
MAXLOGHISTORY 100
MAXDATAFILES 10
MAXINSTANCES 2
ARCHIVELOG
CHARACTER SET WE8ISO8859P1
```

¹ Choosing a Character Set.

http://www.cs.utah.edu/classes/cs5530/oracle/doc/B10501_01/server.920/a96529/ch2.htm#104327

```
NATIONAL CHARACTER SET AL16UTF16  
DATAFILE  
'disk1:df1.dbf' AUTOEXTEND ON,  
'disk2:df2.dbf' AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED  
DEFAULT TEMPORARY TABLESPACE temp ts  
UNDO TABLESPACE undo ts  
SET TIME_ZONE = '+02:00';
```

Once AL16UTF16 UTF-16 has been passed in as the NATIONAL CHARACTER SET VALUE in the Create Database statement, Oracle will automatically store all text defined as SQL "N" data types (NCHAR, NVARCHAR2, NCLOB) in the UTF-16 encoding. Oracle will store UTF-16 characters into columns of these data types regardless of the value passed into the CHARACTER SET parameter. The CHARACTER SET encoding will continue to be used for other character data defined as VARCHAR and CHAR types. The NCHAR data types have been redefined in Oracle9i to be double-byte Unicode data types exclusively.²

SQL SERVER Data Types and Encoding

SQL Server 2000 employs only the UCS-2 Unicode encoding. Though UCS-2 is an earlier version of Unicode, Microsoft's implementation of UCS-2 in SQL Server is compatible with UTF-16.

The difference between UCS-2 and UTF-16 is the support for surrogate code units, a range of 16-bit code units reserved for use in UTF-16 to represent characters with Unicode values in the range U+10000 – U+10FFFF.

The surrogate code units are always used in pairs, with one high (range D800 – DBFF) and one low (range DC00 – DFFF) surrogate code unit.

Although SQL Server is not aware of surrogates, SQL Server will not corrupt the data in a database that contains them; it will treat the pair of surrogate code points as two separate (undefined) characters.

On Windows, both UCS-2 and UTF-16 are stored in a Little Endian manner.

In SQL Server, all fields that will store UCS-2 data, must be defined as "N" data types.³

Other Encoding Tasks

There one remaining task related to encoding that must be performed to ensure proper character handling by the Java web application: convert translated properties files to Unicode escapes.

Convert Properties Files to Unicode Escapes

Properties files do not provide a mechanism for indicating the files' encoding, and so the files must be encoded in a form that Java can interpret correctly as Unicode characters. This form is known as Unicode escapes. A Unicode escape indicates a Unicode character value and is interpreted by Java into that character. Some examples:

² Supporting Multilingual Databases with Unicode.

http://www.cs.utah.edu/classes/cs5530/oracle/doc/B10501_01/server.920/a96529/ch5.htm#1017484

³ Kaplan, Michael. "International Features in Microsoft SQL Server 2000." April, 2001.

<http://msdn.microsoft.com/library/default.asp?url=/library/enus/dnsq2k/html/intlfeaturesinsqlserver2000.asp>

"\u0041" = 'A' (capital letter A)
"\u263A" = '☺' (smiley face)
"\u65E5" = '日' (Chinese/Japanese character for "sun")

Properties files containing extended characters should be processed so that those characters are converted to Unicode escapes. This is accomplished on the command line using the `native2ascii` command, which takes an `-encoding` switch to indicate the encoding of the file, the name of the source file, and the name of the target file. For example, assuming you have a set of properties files with the base name "Messages," here is the command to convert the Japanese file from UTF-8 to Unicode escapes:

```
$ native2ascii -encoding UTF-8 Messages_ja.properties Messages_ja.properties
```

This example uses the same file name for the source and the target. The process can also be reversed, for example if you need to be able to edit the text in the properties files (likely at some point), by including the `-reverse` switch.

```
$ native2ascii -reverse -encoding UTF-8 Messages_ja.properties \  
> Messages_ja.properties
```

The `-reverse` switch changes the meaning of the `-encoding` switch from source encoding to target encoding.

In actuality, there is one other possible encoding to use with properties files: the default encoding of the system. So if the system uses a Latin-1 encoding such as ISO-8859-1 or Cp1252, properties files for Western European languages could be encoding in either of these without needing to convert to Unicode escapes. This is not recommended in a multilingual environment, however, since ideally all files should be encoded as UTF-8 during editing, and converted to Unicode escapes for deployment. The value of this approach is that all files are uniform, so there is no need to keep track of different encodings for different files.

About Lingoport

Lingoport provides internationalization services and its Globalyzer software, which combine to help our clients to become nimble and effective in providing high quality products for any worldwide locale. Services include internationalization architectural planning, internationalization code development and testing as well as internationalization training. Globalyzer helps teams of developers find and fix internationalization issues buried within large volumes of source code. You can find out more about Lingoport at <http://www.lingoport.com>.